Automated Theorem Proving 3/4: Clause Sets and Resolution

A.L. Lamprecht

Course Program Semantics and Verfication 2020, Utrecht University

September 28, 2020

Lecture Notes

"Automated Reasoning" by Gerard A.W. Vreeswijk.
Available for download on the course website.
My slides are largely based on them.

In This Course

- Propositional theorem proving (last Monday), Chapter 2 of the lecture notes
- First-order theorem proving (last Wednesday), Chapter 3 of the lecture notes
- Clause sets and resolution (today), Chapters 4 and 5 of the lecture notes
- Satisfiability checkers, SAT/SMT (Wednesday), Chapter 6 of the lecture notes, additional material

Recap: First-Order Theorem Proving

- Reduction Rules for FOL
- Herbrand Domains
- FOL theorem proving with no functions and no equality
- FOL theorem proving with functions and no equality
- Skolem functions (postponed substitution)
- Unification
- FOL theorem proving with functions and equality
- Sound- and Completeness
- Complexity

Clause Sets

- Tableaux and sequent calculi are quite intuitive, but also quite slow.
- Resolution-based techniques are somewhat less intuitive, but faster.
- Thus, resolution is a practical and therefore important ATP technique.
- Resolution-based theorem provers operate on clauses only.
- Clause sets are set-representations of conjunctive normal forms.

Conjunctive Normal Forms

A *conjunctive normal form* (CNF) is a conjunction of disjunctions of literals, for example $(\neg p \vee q) \wedge (r \vee s) \wedge (t \vee u)$.
Properties:

1. A particular example of a CNF is the *empty* CNF, which is one with zero terms.
2. Because a CNF consists of conjunctions, a CNF is true if and only if all its terms are true.
3. By (1) and (2), the empty CNF is vacuously true.
4. Because every term of a CNF is a disjunction, a term of a CNF is true if and only if it contains two complementary literals.
5. By (2) and (4), a CNF is a tautology if and only if all terms possess complementary literals. It is therefore easy to check (in polynomial time) whether a CNF is a tautology.

Every proposition can be written in CNF.

Clause Sets

It is advantageous to write CNFs as *clause sets*, in which each clause represents a disjunction of literals.

Example:

$$\begin{aligned}(p \vee \neg q) \wedge (\neg p \vee q \vee \neg r) &\equiv \{p \vee \neg q, \neg p \vee q \vee \neg r\} \\ &\equiv \{\{p, \neg q\}, \{\neg p, q, \neg r\}\}\end{aligned}$$

The *empty clause* is denoted by $\Box$. Since a clause represents a disjunction, and since a disjunction is true if and only if one of its components is true, we have $\Box \equiv \mathrm{false}$. Similarly, if $\emptyset$ is regarded as an empty clause set, we have $\emptyset \equiv \mathrm{true}$.

Complexity of Rewriting into Normal Form

- Conceptually simple, but computationally hard.
- Depending on the structure of the formula, exponential blow-up may occur.
- In fact, impossible to rewrite an arbitrary formula into an equivalent CNF in polynomial time.
- See the lecture notes for further details.

Conversion to $\leq$3CNF in Linear Time

- Observation: The essential invariant of resolution is satisfiability (not logical equivalence).
- Possible to produce a clause set in linear time that is satisfiable if and only if ϕ is satisfiable!
- Computation done by the *Tseitin-derivative* of ϕ:
  - a conjunction of equivalences
  - LHS is a proposition letter corresponding to a sub-formula
  - RHS is a conjunction, negation, implication or disjunction of other sub-formulas
  - conjunctions in the Tseitin-derivative correspond to conjunctions in ϕ, etc.
- Tseitin derivation process linearly depends on the size of ϕ.

Tseitin-Derivative (Example)

Let $\phi = (\neg q) \wedge (p \supset ((\neg q)) \vee p\,)$.

Make a table of sub-formulas, associate with proposition letters:

Subformula	*New proposition letter*
ϕ	r
$\neg q$	s
$p \supset ((\neg q)) \vee p\,)$	t
$(\neg q) \vee p$	u
$\neg q$	v

The Tseitin-derivative of ϕ is now:

$$TS(\phi) = r \wedge (r \equiv s \wedge t) \wedge (s \equiv \neg q) \wedge (t \equiv p \supset u) \wedge (u \equiv v \vee p\,) \wedge (v \equiv \neg q)$$

Tseitin-Derivative (Example)

Equivalences are reduced to $\leq$3CNF clause sets as follows:

Subformula	*CNF (in clause set notation)*
$r \equiv s \wedge t$	$\{\{r, \neg s, \neg t\}, \{\neg r, s\}, \{\neg r, t\}\}$
$s \equiv \neg q$	$\{\{s, q\}, \{\neg s, \neg q\}\}$
$t \equiv p \supset u$	$\{\{p, t\}, \{t, \neg u\}, \{\neg p, \neg t, u\}\}$
$u \equiv v \vee p$	$\{\{u, \neg v\}, \{u, \neg p\}, \{\neg u, v, p\}\}$
$v \equiv \neg q$	$\{\{v, q\}, \{\neg v, \neg q\}\}$

Thus:

$$\begin{aligned} TS(\phi) = \{\{r\}\} \cup \{&\{r, \neg s, \neg t\}, \{\neg r, s\}, \{\neg r, t\}, \\ &\{s, q\}, \{\neg s, \neg q\}, \\ &\{p, t\}, \{t, \neg u\}, \{\neg p, \neg t, u\}, \\ &\{u, \neg v\}, \{u, \neg p\}, \{\neg u, v, p\}, \\ &\{v, q\}, \{\neg v, \neg q\}\} \end{aligned}$$

Exercise

Rewrite the following formulas as $\leq$3CNF clause sets.

1. $a \wedge b \wedge c \wedge d \wedge e$
2. $a \vee b \vee c \vee d \vee e$

Solution (1)

Fair enough: $\{\{a\},\{b\},\{c\},\{d\},\{e\}\}$

Solution (2)

Definitely harder than the first one! First, write ϕ as $a \vee (b \vee (c \vee (d \vee e)))$. (Alternatively you may write ϕ as $(((a \vee b) \vee c) \vee d) \vee e$, or $(((a \vee b)) \vee (c \vee d))) \vee e$, or whatever. In each case you end up with another Tseitin derivative.) Then decompose ϕ in subformulas where each formula receives its own letter:

$$\phi \equiv q \wedge (q \equiv (a \vee r)) \wedge (r \equiv (b \vee s)) \wedge (s \equiv (c \vee t)) \wedge (t \equiv (d \vee e))$$

We now rewrite each equivalence of the form $u \equiv v \vee p$ into a clause set of the form $\{\{u, \neg v\}, \{u, \neg p\}, \{\neg u, v, p\}\}$. In this way, ϕ is equivalent to

$$\begin{aligned}\phi \quad \equiv \quad & \{\{q\}\} \cup \\ & \{\{q, \neg a\}, \{q, \neg r\}, \{\neg q, a, r\}\} \cup \\ & \{\{r, \neg b\}, \{r, \neg s\}, \{\neg r, b, s\}\} \cup \\ & \{\{s, \neg c\}, \{s, \neg t\}, \{\neg s, c, t\}\} \cup \\ & \{\{t, \neg d\}, \{t, \neg e\}, \{\neg t, d, e\}\}.\end{aligned}$$

Solution (2, cont'd)

Hence,

$$\begin{aligned}\phi \quad \equiv \quad & \{\{q\}\}, \{\{q, \neg a\}, \{q, \neg r\}, \{\neg q, a, r\}\}, \{r, \neg b\}, \{r, \neg s\}, \{\neg r, b, s\}, \\ & \{s, \neg c\}, \{s, \neg t\}, \{\neg s, c, t\}, \{t, \neg d\}, \{t, \neg e\}, \{\neg t, d, e\}\}.\end{aligned}$$

The Idea Behind Resolution

- Resolution is an inference process on clause sets that takes a number of clauses to infer a new clause.
- The idea behind resolution is to prove a theorem by proving that its negation is inconsistent with the assumptions on the basis of which it is proven.

The Idea Behind Resolution (cont'd)

For example, suppose that we would like to know whether

$$\neg p,\ (\neg q \wedge r) \supset p,\ \neg q \vdash \neg r.$$

Thus, we would like to prove $\neg r$ on the basis of $\neg p$, $(\neg q \wedge r) \supset p$, and $\neg q$. This is the same as proving the inconsistency of

$$\{\neg p,\ (\neg q \wedge r) \supset p,\ \neg q\} \cup \{r\},$$

which amounts to proving the inconsistency of

$$(\neg p) \wedge ((\neg q \wedge r) \supset p) \wedge (\neg q) \wedge r.$$

The Idea Behind Resolution (cont'd)

Rewriting $(\neg p) \wedge ((\neg q \wedge r) \supset p) \wedge (\neg q) \wedge r$ in CNF yields

$$(\neg p) \wedge (p \vee q \vee \neg r) \wedge (\neg q) \wedge r,$$

which is equivalent with the clause set

$$S = \{\{\neg p\}, \{p, q, \neg r\}, \{\neg q\}, \{r\}\}.$$

The objective is to show that S is inconsistent (or unsatisfiable, which is the same for that matter). This is done by adding clauses to S and deleting clauses from S *such that satisfiability of S remains invariant*. If, somewhere in the process, the empty clause $\square$ is added to S, we know that S must have been unsatisfiable, so that $\neg p, (\neg q \wedge r) \supset p, \neg q \vdash \neg r$ follows.

Cleaning Up and Simplifying Clause Sets

Useful/necessary at the beginning and during the resolution process to reduce the amount of “garbage” clauses.

The One-Literal Rule (OLR)

Can be applied as soon as there are unit clauses. For example:

$$F = p \;\wedge\; (p \vee \phi) \;\wedge\; (\neg p \vee \psi) \;\wedge\; \chi$$

where ϕ, ψ and χ are $\{p, \neg p\}$-free. Obviously, F corresponds to a clause set of the form

$$\{\{p\},\, \{p\} \cup A,\, \{\neg p\} \cup B,\, C\}$$

F is satisfiable if and only if the formula $\psi \wedge \chi$ is satisfiable. Thus, the clause set may be reduced to:

$$\{B,\, C\}$$

The transition does not respect logical equivalence, but is satisfiability-equivalent.

The One-Literal Rule (OLR) (cont'd)

The general case amounts to

$$F = p \wedge (p \vee \phi_1) \wedge \ldots \wedge (p \vee \phi_m) \wedge (\neg p \vee \psi_1) \wedge \ldots \wedge (\neg p \vee \psi_n) \wedge \chi_1 \wedge \ldots$$

where all the ϕ_i's, ψ_j's and χ_k's are $\{p, \neg p\}$-free.

As above, it can be shown that F is satisfiable if and only if $\psi_1 \wedge \ldots \wedge \psi_n \wedge \chi_1 \wedge \ldots \wedge \chi_r$ is satisfiable.

Thus, every clause set of the form

$$\{\{p\}, \{p\} \cup A_1, \ldots, \{p\} \cup A_m, \{\neg p\} \cup B_1, \ldots, \{\neg p\} \cup B_n, C_1, \ldots, C_r\}$$

may be reduced to

$$\{B_1, \ldots, B_n, C_1, \ldots, C_r\}$$

respecting satisfiability-equivalence.

Monotone Variable Fixing

- a.k.a. *Pure Literal Rule (PLR)*
- Idea: Verify if every literal of every clause is complemented by a literal in some other clause.
- If not, the entire clause is useless and can be removed.
- Validity resides on the fact that

$$\phi_1 \wedge \ldots \wedge \phi_m \wedge (\psi_1 \vee p) \wedge \ldots \wedge (\psi_n \vee p)$$

is satisfiable iff $\phi_1 \wedge \ldots \wedge \phi_m$ is satisfiable, provided that $\phi_1, \ldots, \phi_n$ are $\neg p$-free.

Example

$$S = \{\{p, \neg q, t\}, \{\neg q, r, \neg s\}, \{\neg t, \neg u, \}, \{r, \neg s, \neg q\}, \{p, u, \neg v\}\}$$

The literals p, $\neg q$, r, $\neg s$, and $\neg v$ occur "uncomplemented" in S, and can therefore be removed by monotone variable fixing.
If p is set to true and q to false, we obtain:

$$\{\underline{\{p, \neg q, t\}}, \{\neg q, r, \neg s\}, \{\neg t, \neg u, \}, \{r, \neg s, \neg q\}, \underline{\{p, u, \neg v\}}\} \qquad \text{set } p = 1$$

$$\{\underline{\{\neg q, r, \neg s\}}, \{\neg t, \neg u, \}, \underline{\{r, \neg s, \neg q\}}\} \qquad \text{set } q = 0$$

$$\{\underline{\{\neg t, \neg u, \}}\} \qquad \text{set } u = 0$$

$$\emptyset.$$

In the process, $\neg t$ and $\neg u$ become uncomplemented and the last clause is also deleted (by setting u to false.)

Tautology Rule

Based on the equivalence

$$(p \lor \neg p \lor \phi) \land \psi \equiv \psi$$

clauses with complementary literals (tautologies) may be removed from the clause set without compromising logical equivalence.

Subsumption

- If C_i and C_j are two clauses in a clause set $\{C_1, \ldots, C_n\}$ such that $C_i \subset C_j$, we say that C_i *subsumes* C_j, or that C_j is *subsumed by* C_i.
- The dependent clause C_j may be deleted on the basis of the fact that $C_i \vdash C_j$, so that

$$C_1 \wedge \ldots \wedge C_n \equiv C_1 \wedge \ldots \wedge C_{j-1} \wedge C_{j+1} \wedge \ldots \wedge C_n.$$

- With clause sets, subsumption is equivalent to the subset relation.

The Davis-Putnam/Logemann-Loveland Algorithm (DPLL)

- Idea: pick an arbitrary proposition variable and apply the OLR in both directions (*Splitting Rule*).
- Rests on $\phi \equiv (\phi \wedge p) \vee (\phi \wedge \neg p)$
- Procedure (apply repeatedly):
  1. *Selection.* Select a literal p.
  2. *Distinguishing cases.* Split the current clause set $C = \phi$ into a clause set $C_1 = \phi \cup \{p\}$ and the clause set $C_2 = \phi \cup \{\neg p\}$.
  3. *Recursion.* Proceed with C_1. If C_1 is satisfiable, then C is satisfiable with $p = 1$ and halt. Else, proceed with C_2. If C_2 is satisfiable then C is satisfiable too, with $p = 0$. Else C is unsatisfiable.
- Produces a binary tree of clause sets.
- DPLL is sound and complete (proof in lecture notes).

Example

Example:

$$\{\{t\}, \{\neg t, \neg r\}, \{p, \neg r, s, \neg t\}, \{\neg p, q, r\}, \{\neg s, u, \neg u\}, \{p, \neg q, \neg t\}, \{p, t\}\}.$$

Application of different simplifications and DPLL algorithm:

$\{\{t\}, \{\neg t, \neg r\}, \{p, \neg r, s, \neg t\}, \ldots$

$\ldots \{\neg p, q, r\}, \{\neg s, u, \neg u\}, \{p, \neg q, \neg t\}, \{p, t\}\}$ tautology rule with $\neg u$

$\{\{t\}, \{\neg t, \neg r\}, \{p, \neg r, s, \neg t\}, \{\neg p, q, r\}, \{p, \neg q, \neg t\}, \{p, t\}\}$ OLR with t

$\{\{\neg r\}, \{p, \neg r, s\}, \{\neg p, q, r\}, \{p, \neg q\}\}$ OLR with $\neg r$

$\{\{\neg p, q\}, \{p, \neg q\}$ split with p

$\{\{p\}, \{\neg p, q\}, \{p, \neg q\}\}$ OLR with p

$\{\{q\}\}$

(We split $C = \{\{\neg p, q\}, \{p, \neg q\}\}$ into $C_1 = \{\{p\}, \{\neg p, q\}, \{p, \neg q\}\}$ and $C_2 = \{\{\neg p\}, \{\neg p, q\}, \{p, \neg q\}\}$ and proceeded with C_1.)

Exercise

Apply the DPLL to the following clause sets:

1. $\{\{a\}, \{\neg a, b\}, \{\neg b, \neg c, d\}, \{\neg d, e\}, \{e\}\}$
2. $\{\{a, b, c\}, \{a, b, \neg c\}, \{a, \neg b, c\}, \{a, \neg b, \neg c\}\} \cup$
   $\{\{\neg a, b, c\}, \{\neg a, b, \neg c\}, \{\neg a, \neg b, c\}, \{\neg a, \neg b, \neg c\}\}$

Solution (1)

	$\{\{a\}, \{\neg a, b\}, \{\neg b, \neg c, d\}, \{\neg d, e\}, \{e\}\}$
OLR with a	$\{\{b\}, \{\neg b, \neg c, d\}, \{\neg d, e\}, \{e\}\}$
OLR with b	$\{\{\neg c, d\}, \{\neg d, e\}, \{e\}\}$
OLR with e	$\{\{\neg c, d\}\}$
	satisfiable with $\neg c$, d.

Hence, original clause set satisfiable with $\neg c$, d, and e, b, a.

Solution (2)

This clause set is an exhaustive enumeration of all "possible worlds" that can be made with a, b, and c. Clearly, no simple reduction rule can be applied. Therefore, we will have to split. Since this clause set is symmetrical the choice of the splitting variable does not matter.

	$\{\{a,b,c\},\{a,b,\neg c\},\{a,\neg b,c\},\{a,\neg b,\neg c\}\} \cup$ $\{\{\neg a,b,c\},\{\neg a,b,\neg c\},\{\neg a,\neg b,c\},\{\neg a,\neg b,\neg c\}\}$
SPLIT on a, branch a:	$\{\{b,c\},\{b,\neg c\},\{\neg b,c\},\{\neg b,\neg c\}\}$
SPLIT on b, branch b:	$\{\{c\},\{\neg c\}\}$
OLR with c:	$\{\Box\}$

Hence, the original clause set is unsatisfiable in branch $a - b - c$. So this branch does not yield a counter-model. Similarly, other branches (like $a - b - \neg c$, $a - \neg b - c$, and $a - \neg b - \neg c$) will not yield counter-models either. We can conclude that the original clause set is unsatisfiable.

Choice of Branching Variables

- Important step in DPLL: If good branching variables are chosen, the search tree stays relatively small.
- MOM's heuristic: Pick the literal that occurs most often in the smallest clauses.
- Jeroslow-Wang heuristic: Estimate the contribution that each literal is likely to make to satisfying the clause set.
- Most proposals can be divided in the following steps:
  1. *Restrict*. Determine a set B of candidate-branching variables.
  2. *Estimate*. For each $b \in B$, compute $f(b)$ and $f(\neg b)$, where f is some heuristic function that estimates the quality of branching on b.
  3. *Balance*. Compare, or balance the two values $f(b)$ and $f(\neg b)$, by means of some other heuristic function $g : R^2 \to R$.
  4. *Choose*. Take $b \in B$ such that $g(f(b), f(\neg b))$ is maximal. Break ties if necessary.

Resolution

- Small inference steps: verifiable (by a human), but combinatorial explosion of the search space.
- Robinson (1965): resolution as explicitly machine-oriented, more efficient form of inference.

Binary Resolution

- Precisely two clauses are used to infer a new clause.
- Basis: $(a \vee p) \wedge (b \vee \neg p) \equiv (a \vee p) \wedge (b \vee \neg p) \wedge (a \vee b)$
- Extend clause sets with other (hopefully simpler) clauses.
- For example,

$$\{C_1, \ldots, C_m, \{a, p\}, \{b, \neg p\}\}$$

  may be extended with the clause $\{a, b\}$:

$$\{C_1, \ldots, C_m, \{a, p\}, \{b, \neg p\}, \{a, b\}\}.$$

- The new clause is obtained by joining the two parent clauses such that a complementary pair of literals is deleted.

Binary Resolution (cont'd)

- The clause $\{a, b\}$ is called a *resolvent* of the clauses $\{a, p\}$ and $\{b, \neg p\}$ and we write:

$$\{a, p\}, \{b, \neg p\} \rightsquigarrow \{a, b\}$$

- Resolvents are not produced for their own sake. If produced and added repeatedly, we may or may not arrive at the empty clause $\Box$. If we do, we have shown that the original clause set was inconsistent.
- A sequence of elementary resolution steps that ends with the empty clause is also called a *resolution refutation*.

Example

Suppose we would like to prove $\phi = [(p \supset r) \wedge (q \supset r) \wedge (p \vee q)] \supset r$, with refutation by resolution. To this end, we form the negation of ϕ, and convert it to a clause set:

$$\begin{aligned}
\neg\phi &\equiv \neg[[(p \supset r) \wedge (q \supset r) \wedge (p \vee q)] \supset r] \\
&\equiv [(p \supset r) \wedge (q \supset r) \wedge (p \vee q)] \wedge \neg r \\
&\equiv (\neg p \vee r) \wedge (\neg q \vee r) \wedge (p \vee q) \wedge \neg r \\
&\equiv \{\{\neg p, r\}, \{\neg q, r\}, \{p, q\}, \{\neg r\}\}.
\end{aligned}$$

Extend repeatedly by adding resolvents:

$$\begin{aligned}
&\equiv \{\{\neg p, r\}, \{\neg q, r\}, \{p, q\}, \{\neg r\}\} && \{\neg q, r\}, \{p, q\} \rightsquigarrow \{p, r\} \\
&\equiv \{\{\neg p, r\}, \{\neg q, r\}, \{p, q\}, \{\neg r\}, \{p, r\}\} && \{\neg p, r\}, \{p, r\} \rightsquigarrow \{r\} \\
&\equiv \{\{\neg p, r\}, \{\neg q, r\}, \{p, q\}, \{\neg r\}, \{p, r\}, \{r\}\} && \{\neg r\}, \{r\} \rightsquigarrow \Box \\
&\equiv \{\{\neg p, r\}, \{\neg q, r\}, \{p, q\}, \{\neg r\}, \{p, r\}, \{r\}, \Box\}
\end{aligned}$$

We end up with an empty clause $\Box$, which means that $\neg\phi$ is false, so that ϕ is true. Hence the proof of ϕ is completed.

Example (cont'd)

The entire resolution process can be depicted linearly:

1.	$\{\neg p, r\}$	*premise*
2.	$\{\neg q, r\}$	*premise*
3.	$\{p, q\}$	*premise*
4.	$\{\neg r\}$	*premise*
5.	$\{p, r\}$	*(resolvent of 2 and 3)*
6.	$\{r\}$	*(resolvent of 1 and 5)*
7.	$\square$	*(resolvent of 4 and 6),*

or as a directed acyclic graph (DAG):

$$\begin{tikzcd}
\{\neg p, r\} \arrow[ddr] & & \{\neg q, r\} \arrow[dr] & & \{p, q\} \arrow[dl] & & \{\neg r\} \arrow[dddllll] \\
 & & & \{p, r\} \arrow[dll] & & & \\
 & \{r\} \arrow[dr] & & & & & \\
 & & \square & & & &
\end{tikzcd}$$

Exercise

Use (binary) resolution to prove that the following proposition is unsatisfiable. Convert to clause sets first.

$$(r \vee \neg u) \wedge (\neg r \vee s) \wedge (u \vee s) \wedge (\neg s \vee v) \wedge (\neg v \vee \neg s)$$

Solution

One possible solution:

$$\begin{aligned}
\neg\phi \equiv & \{\{r, \neg u\}, \{\neg r, s\}, \{u, s\}, \{\neg s, v\}, \{\neg v, \neg s\}\} & \{r, \neg u\}, \{\neg r, s\} \rightsquigarrow \{\neg u, s\} \\
\equiv & \{\{r, \neg u\}, \{\neg r, s\}, \{u, s\}, \{\neg s, v\}, \{\neg v, \neg s\}, & \{u, s\}, \{\neg u, s\} \rightsquigarrow \{s\} \\
& \{\neg u, s\}\} & \\
\equiv & \{\{r, \neg u\}, \{\neg r, s\}, \{u, s\}, \{\neg s, v\}, \{\neg v, \neg s\}, & \{\neg s, v\}, \{\neg v, \neg s\} \rightsquigarrow \{\neg s\} \\
& \{\neg u, s\}, \{s\}\} & \\
\equiv & \{\{r, \neg u\}, \{\neg r, s\}, \{u, s\}, \{\neg s, v\}, \{\neg v, \neg s\}, & \{s\}, \{\neg s\} \rightsquigarrow \Box \\
& \{\neg u, s\}, \{s\}, \{\neg s\}\} & \\
\equiv & \{\{r, \neg u\}, \{\neg r, s\}, \{u, s\}, \{\neg s, v\}, \{\neg v, \neg s\}, & \\
& \{\neg u, s\}, \{s\}, \{\neg s\}, \Box\} &
\end{aligned}$$

Sound- and Completeness

- Binary resolution is sound and complete for propositional logic.
- Proof in the lecture notes.
- Good: we have a form of resolution that is complete.
- Not so good: it's inefficient.
- We will look at different alternatives in the following.

Linear Resolution

Given a set S of clauses, a *linear resolution* of $C_0, \ldots, C_n$ from S is a resolution where $C_0 \in S$ and, for each $1 \leq i \leq n$, C_i is a resolvent of C_{i-1} and B where B is some previous clause (that may be in S but must differ from C_i).

- Simple structure.
- Sound and complete for propositional logic.
- (Proof sketch in the lecture notes.)
- Resolvents often larger than their parent clauses.

Unit Resolution and Input Resolution

Unit Resolution:

- At least one of the parent clauses must be a unit clause.
- Very much alike OLR (see example in lecture notes).
- Resolvents not larger than parent clauses.

Input Resolution:

- Let S be an input set.
- Each resolvent has a parent in S.

Semantic Resolution

- Powerful specialization of binary resolution.
- Pick a model m and divide the clause set S into two groups: one group S_1 that is made true by m, i.e., $S_1 =_{Def} \{ C \mid m \models C \}$ and the rest $S_2 =_{Def} S \setminus S_1$.
- Then require that parent clauses must come from different groups.
- Semantic resolution is complete.
- Often combined with an ordering on proposition letters or predicate symbols.

Ordered Resolution

- Only the $<$-smallest element of the *m*-falsified clause of every resolution step $C_1, C_2 \rightsquigarrow R$ may be used as a literal to resolve upon.
- In such cases, R is called $<$-*admissible*.
- Ordered semantic resolution is still complete.

Example

For example, if

$$C_1 = \{a, \neg g, k, m, \neg p\} \text{ and } C_2 = \{\neg b, c, \neg k, \neg n\}$$

then

$$R_1 = \{a, \neg b, c, \neg g, m \neg n, \neg p\}$$

would normally be a resolvent of C_1 and C_2.

However, R_1 is not a $<$-admissible resolvent of C_1 and C_2, because a, rather than k, is the $<$-smallest element of C_1.

If $C_3 = \{k, m, \neg p\}$, then C_3 can be resolved with C_2, because k is the $<$-smallest element of C_3.

Thus, $C_3, C_2 \rightsquigarrow R_2$ with $R_2 = \{\neg b, c, \neg k, m, \neg n, \neg p\}$ would be a $<$-admissible resolvent of C_3 and C_2.

Semantic Clash

- Ordered semantic resolution may be further constrained without loosing completeness.
- For example: $S = \{\{\neg p, \neg q, r\}, \{p, r\}, \{q, r\}, \{\neg r\}\}$
- We apply ordered semantic resolution with $p < q < r$ and with m such that m falsifies all positive literals.
- There are two m-falsified (i.e., positive) clauses: $\{p, r\}$ and $\{q, r\}$. The $<$-least elements of both clauses are p and q.
- With ordered semantic resolution both clauses can only resolve with the first clause to produce $\{\neg q, r\}$ and $\{\neg p, r\}$.
- No new positive clauses are added, so the new clauses can only resolve with the two positive clauses already present to produce $\{r\}$ and $\{r\}$.
- From $\{r\}$ (positive) and $\{\neg r\}$ (non-positive) we may then produce the empty clause.

Semantic Clash

Because the two non-positive clauses may never resolve with each other, the step to $\{r\}$ could have been taken at once:

$$\{\overset{\{p,r\}}{\neg p}, \quad \overset{\{q,r\}}{\neg q}, \quad r\} \quad \Rightarrow \quad \{r\}$$

This is not an ordinary resolution step, so it has to have some other name, viz. *semantic clash*, or clash for short.

Semantic Clash

Let m be a model, and let $<$ be an ordering of proposition letters. A clause set

$$\{N, C_1, \ldots, C_n\}, n \geq 1$$

is called a *semantic clash* relative to m and $<$ if

1. $R_1 = N$ and for each $1 \leq i \leq n$, there is a $<$-admissible resolvent $C_i, R_i \rightsquigarrow R_{i+1}$
2. $C_1, \ldots, C_n$ and R_{n+1} are falsified by m

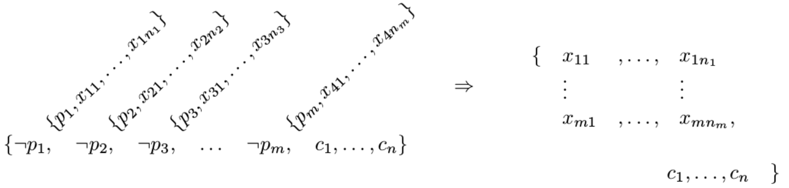
N is called the *nucleus*, $C_1, \ldots, C_n$are called the *satellites*, and R_{n+1} is called a *resolvent*.

Hyperresolution

- Important special case of Semantic Clash.
- m is such that it falsifies all literals.
- Thus, the nucleus is non-positive, while the satellites and the resolvent are positive.
- In a clash, the satellites remove the negative literals from the nucleus to produce a positive resolvent which may then be used as a satellite in future resolution steps.
- A successful application of hyperresolution can be viewed as a sequence of binary resolutions in which each is required to involve exactly one positive clause.
- However, all such binary resolutions must occur simultaneously, thus yielding no intermediate clauses.

Example

Example:



$\{p_1, x_{11}, \dots, x_{1n_1}\}$
$\{p_2, x_{21}, \dots, x_{2n_2}\}$
$\{p_3, x_{31}, \dots, x_{3n_3}\}$
$\{p_m, x_{41}, \dots, x_{4n_m}\}$
$\{\neg p_1, \quad \neg p_2, \quad \neg p_3, \quad \dots \quad \neg p_m, \quad c_1, \dots, c_n\}$
$\Rightarrow$
$\{ \quad x_{11} \quad , \dots , \quad x_{1n_1}$
$x_{m1} \quad , \dots , \quad x_{mn_m},$
$c_1, \dots, c_n \quad \}$

Another way to understand hyperresolution, is to represent clauses as implications. If $c =_{Def} c_1 \wedge \dots \wedge c_n$, and the hyperresolution above is written as

$$\begin{array}{lcccc}
\neg x_{11} \wedge \dots \wedge \neg x_{1n_1} \rightarrow p_1 & & & & \neg x_{11} \wedge \quad \dots \quad \wedge \neg x_{1n_1} \\
\vdots \quad \vdots & & & & \vdots \qquad\qquad \vdots \\
\neg x_{11} \wedge \dots \wedge \neg x_{1n_m} \rightarrow p_m & \text{and} & p_1 \wedge \dots \wedge p_m \rightarrow c & \text{yields} & \neg x_{m1} \wedge \quad \dots \quad \wedge \neg x_{mn_m} \rightarrow c
\end{array}$$

we see that the implications $\{\neg\bar{x} \rightarrow p_j\}_{j=1}^{m}$ together with the implication $p_1, \dots, p_m \rightarrow c$, make a new implication $\neg\bar{x}_1 \wedge \dots \wedge \neg\bar{x}_m \rightarrow c$. Thus, with a little benevolence, hyperresolution can be seen as a disguised form of rule-based reasoning.

Exercise

Show with hyperresolution that the following clause set is unsatisfiable. Remember that with hyperresolution the satellites are positive, so that all resolvents are positive as well.

$$\{\{\neg d, e\}, \{a, \neg d, \neg e, f\}, \{d\}, \{\neg a, \neg d, g\}, \{\neg e, \neg f, g\}, \{\neg g\}\}$$

Solution

One possible refutation:

1.	$\{\neg d, e\}$	*[input]*
2.	$\{a, \neg d, \neg e, f\}$	*[input]*
3.	$\{d\}$	*[input]*
4.	$\{\neg a, \neg d, g\}$	*[input]*
5.	$\{\neg e, \neg f, g\}$	*[input]*
6.	$\{\neg g\}$	*[input]*
7.	$\{e\}$	*[hyper,1,3]*
8.	$\{a, f\}$	*[hyper,2,3,7]*
9.	$\{a, g\}$	*[hyper,8,5,7]*
10.	$\{a\}$	*[hyper,9,6]*
11.	$\{g\}$	*[hyper,10,4,3]*
12.	$\Box$	*[binary,11,6]*

First-Order Resolution

Most of what holds for propositional resolution goes through for the first-order case, but some concepts are deepened due to the fact that the language of first-order logic is more expressive.

One such concept, for example, is the conversion from non-clausal formulas to a clause set.

This is called *normalization*.

Normalization

How to rewrite an arbitrary first-order formula into a clause set:

1. If the task is to prove that ψ follows from $\phi_1, \ldots, \phi_n$, then deny $\phi_1, \ldots, \phi_n \vdash \psi$ for the purpose of resolution refutation in the form of one big formula: $\phi_1 \wedge \ldots \wedge \phi_n \wedge \neg\psi$. All subsequent operations can be performed in this one big formula, or on every conjunct in isolation.
2. Convert all implications, bi-implications, exclusive ors, nands and nors to equivalent sub-formulas in which only the connectives $\neg$, $\wedge$ and $\vee$ occur.
3. Ensure that all quantifiers use variables that are different and do not occur free elsewhere in the formula. This can simply be done by giving each quantifier its own variable Now the formula is said to be *rectified*.

Normalization

4. Make that all quantifiers come first. This can be done with the following *quantifier rewrite rules*:

$$\begin{array}{rcl@{\qquad}rcl}
\neg(\forall x)(\phi) & \rightsquigarrow & (\exists x)(\neg\phi) & \neg(\exists x)(\phi) & \rightsquigarrow & (\forall x)(\neg\phi) \\
(\forall x)(\phi) \wedge \psi & \rightsquigarrow & (\forall x)(\phi \wedge \psi) & (\exists x)(\phi) \wedge \psi & \rightsquigarrow & (\exists x)(\phi \wedge \psi) \\
\phi \wedge (\forall x)(\psi) & \rightsquigarrow & (\forall x)(\phi \wedge \psi) & \phi \wedge (\exists x)(\psi) & \rightsquigarrow & (\exists x)(\phi \wedge \psi) \\
(\forall x)(\phi) \vee \psi & \rightsquigarrow & (\forall x)(\phi \vee \psi) & (\exists x)(\phi) \vee \psi & \rightsquigarrow & (\exists x)(\phi \vee \psi) \\
\phi \vee (\forall x)(\psi) & \rightsquigarrow & (\forall x)(\phi \vee \psi) & \phi \vee (\exists x)(\psi) & \rightsquigarrow & (\exists x)(\phi \vee \psi)
\end{array}$$

   Now the formula is said to be in *prenex normal form*

5. Skolemize variables that are bound by existential quantifiers. In this way, the existential quantifiers are deleted.
6. Drop all quantifiers.
7. Rewrite the formulas in CNF, or (optional) in $\leq$3CNF.
8. Write the CNF thus obtained as a clause set.
9. Introduce fresh variables where necessary so that no variable occurs in more than one clause. Now the clauses are said to be *standardized apart*.

Example

Here is a typical example of a normalized first-order formula:

$$\begin{array}{rll}
\phi = & \{\{\neg pa, \varsigma_1(v_1) \vee \neg rv_1\}, & \{pv_3, g(b, v_3) \vee \neg qf(b), \varsigma_1(v_4), v_5 \vee \neg ra\}, \\
& \{qa, b, c \vee \neg rv_6\}, & \{pf(g(c, \varsigma_0), v_7), f(c) \vee \neg qc, \varsigma_3(v_8, v_9, f(v_{10})), c\}, \\
& \{pv_{11}, g(b, v_{11}) \vee qc, b, a\}, & \{\neg pg(b\varsigma_1(v_{12}), f(a) \vee \neg rf(v_{12})\}, \\
& \{\neg p\varsigma_4(v_{13}, v_{14})\}, & \{\neg pg(\varsigma_0, a), v_{15} \vee qv_{15}, v_{16}, v_{15} \vee \neg r\varsigma_1(v_{15})\}, \\
& \{pa, b \vee \neg qa, b, v_{17} \vee rv_{18}\}, & \{qv_{19}, b, c \vee rc\}\}
\end{array}$$

This formula is perhaps unreadable to us, but resolution-based theorem provers like them this way.

Exercise

Convert the following problems into clause sets. Follow the above steps up to and including standardization of clauses.

1. $\vdash_? (\forall x)(px \wedge \neg px)$
2. $\vdash_? (\exists x)(px \vee \neg px)$

Solution (1)

We follow the prescribed steps:

1. The denial of $\vdash (\forall x)(px \wedge \neg px)$ in formula-form is $\neg(\forall x)(px \wedge \neg px)$.
2. Nothing to do: there are no exotic connectives here unequal to $\neg$, $\wedge$ and $\vee$.
3. Nothing to do: every quantifiers has its own variable.
4. Pulling all quantifiers to the outside yields $(\exists x)\neg(px \wedge \neg px)$.
5. Skolemization yields $\neg(pa \wedge \neg pa)$.
6. Nothing to do here: no universal quantifiers.
7. Rewriting in CNF yields $\neg pa \vee pa$.
8. Writing this as a clause set gives $\{\{\neg pa, pa\}\}$.
9. Nothing to do: no variables.

This clause set is satisfiable. (In this case it is even a tautology.) Hence, no sound resolution method should be able to derive the empty clause from this clause set.

Solution (2)

We follow the prescribed steps:

1. The denial of $\vdash (\exists x)(px \vee \neg px)$ in formula-form is $\neg(\exists x)(px \vee \neg px)$.
4. Pulling all quantifiers to the outside yields $(\forall x)\neg(px \vee \neg px)$.
5. Nothing to do here: no existential quantifiers.
6. Deleting all (universal) quantifiers yields $\neg(px \vee \neg px)$
7. Rewriting in CNF yields $\neg px \wedge px)$.
8. Writing this as a clause set gives $\{\{px\}, \{\neg px\}\}$.
9. Standardizing variables apart yields $\{\{px\}, \{\neg py\}\}$.

This clause says that for every x and y: px and $\neg py$. This is obviously not true if $x = y$. Hence, every decent resolution method should unify the literals of both clauses to apply a resolution step in which the empty clause is formed.

Skolemization

- Step 5 in the normalization process.
- Idea: Make the dependence of existential variables on universal variables explicit by a function that has the universal variables as arguments.

Theorem (Skolemization)

Let ϕ be a first-order formula, and let ς be a function symbol that does not occur in ϕ. Then $(\forall x_1, \ldots, x_n)(\exists y)(\phi)$ is satisfiable if and only if

$$(\forall x_1, \ldots, x_n)(\phi[\varsigma(x_1, \ldots, x_n)/y])$$

is satisfiable.

First-Order Resolution

Definition (First-order resolution)

First-order resolution works analogous to propositional resolution, with the additional requirement that C_1 and C_2 may first have to be standardized apart into $C_1\sigma_1$ and $C_2\sigma_2$, and that a most general unifier may bring about that more than two literals may "clash" in a binary resolution.

$$\begin{tikzcd}
C_1 \arrow[d, "\sigma_1"] & \text{standardize apart with } \sigma_1 \text{ and } \sigma_2 & C_2 \arrow[d, "\sigma_2"'] \\
C_1\sigma_1 \arrow[d, "\tau"] & \text{unify with } \tau & C_2\sigma_2 \arrow[d, "\tau"] \\
C_1\sigma_1\tau = \{\bar{L}_1, \dots, \bar{L}_l, K_1, \dots, K_k\} \arrow[dr] & \text{resolve} & C_2\sigma_2\tau = \{L_1, \dots, L_l, M_1, \dots, M_m\} \arrow[dl] \\
 & R = \{K_1, \dots, K_k, M_1, \dots, M_m\} &
\end{tikzcd}$$

Example

$$\begin{tikzcd}
\{\underline{pf(x)}, \neg qz, \underline{pz}\} \arrow[d] & \text{standardize apart with } \sigma = \{u/x\} & \{\underline{\neg px}, rg(x,a)\} \arrow[d, "\sigma"] \\
\{\underline{pf(x)}, \neg qz, \underline{pz}\} \arrow[d, "\tau"] & \text{unify with } \tau = \{f(x)/z, f(x)/u\} & \{\underline{\neg pu}, rg(u,a)\} \arrow[d, "\tau"] \\
\{\underline{pf(x)}, \neg qf(x)\} \arrow[dr] & \text{resolve} & \{\underline{\neg pf(x)}, rg(f(x),a)\} \arrow[dl] \\
 & \{\neg qf(x), rg(f(x),a)\} &
\end{tikzcd}$$

Why this way? See explanation in the lecture notes.

Resolution Lemma

Lemma (Resolution lemma)

First-order resolution is a sound inference procedure.

Proof: See lecture notes.

Equality

The addition of equality yields expressive power at the price of proof complexity.

With equality rules present, theorem-provers have so many paths to explore that smart methods are necessary to deal with equality.

Two important resolution rules that deal with equality are demodulation and paramodulation.

Demodulation

- Demodulation, or *rewriting*, is based on the idea that every $=$-equivalence class of terms should have a canonical member to which all other elements should be reduced.
- In this way the number of redundent terms (and thus clauses) is significantly reduced.
- Example:

	Predicates to Be Demodulated	Demodulators	Demodulants
(1)	$pf(f(f(a)))$	$f(x) = x$	pa
(2)	$pf(a, f(a, f(a, f(a, x))))$	$f(a, x) = x$	px
(3)	$pf(f(f(f(a, a), a), a), x)$	$f(a, x) = x$	px
(4)	$pf(f(f(f(a, a), a), a), x)$	$f(x, a) = x$	$pf(a, x)$

- A clause that cannot be further reduced with rewrite rules is said to be in *normal form* or *irreducible*.
- Normally used to simplify newly generated clauses.

Exercise

If the clauses in Column 1 are rewritten by means of the demodulators in Column 2, then what clauses are inferred as demodulants?

	Clauses to be demodulated	*Demodulators*
(1)	$D(f(x,x), f(a,a))$	$f(x,a) = a$
(2)	$D(f(f(x,x), f(a,a)))$	$f(x,a) = a$
(3)	$D(f(a,a), f(a,x))$	$f(x,a) = a$

Solution

	Demodulants
(1)	$D(f(x,x),a)$
(2)	$D(a)$
(3)	$D(f(a,a),f(a,x))$

Paramodulation

- Based on Leibniz' law for replacement of equals by equals.
- Technique to infer new clauses from other clauses.
- More powerful than demodulation: resolution-complete for first-order logic with equality
- Also more "dangerous", as it might produce a large number of clauses.
- Can handle cases that are beyond the reach of demodulation (see lecture notes for an example).

Paramodulation

The clause substituted into is called the *into* clause, and the clause containing the equality is called the *from* clause.

One way of paramodulating from C into D:

$$\begin{tikzcd}
C = \{L_1, \ldots, L_m, s = t\} \arrow[d] & \sigma = \mathrm{mgu}(s, s') & D = \{K(s'), K_1, \ldots, K_n\} \arrow[d] \\
C\sigma = \{L_1, \ldots, L_m, s = t\}\sigma \arrow[dr] & & D\sigma = \{K(s'), K_1, \ldots, K_n\}\sigma \arrow[dl] \\
& P = \{L_1, \ldots, L_m, K(t), K_1, \ldots, K_n\}\sigma &
\end{tikzcd}$$

In This Course

- Propositional theorem proving (last Monday), Chapter 2 of the lecture notes
- First-order theorem proving (last Wednesday), Chapter 3 of the lecture notes
- Clause sets and resolution (today), Chapters 4 and 5 of the lecture notes
- Satisfiability checkers, SAT/SMT (Wednesday), Chapter 6 of the lecture notes, additional material

Homework

The following homework exercises are useful to review today's content in preparation for the next lecture:

- Sec. 4.1 Problems 9 and 10 (page 87/88)
- Sec. 4.4 Problems 1–3 (page 96)
- Sec. 5.3 Problems 1 (b)–(c), 2 (page 106)
- Sec. 5.4 Problem 4 (page 108)
- Sec. 5.6 Problem 4 (4)–(6) (page 120)
- Sec. 5.6 Problem 5 (page 120)