What is this?

- Resolution is a special kind of theorem proving used in:
  - Automated theorem proving and reasoning, where the goal is complete automation
  - Databases and AI Answer Extraction
  - Prolog language (a subset of general resolution)
- Resolution forms a complete proof system for refutation.
Resolution Example

Consider a problem from an earlier homework:

Prove the following using natural deduction (with = rules):

\[ \forall x \forall y \forall z ((P(x, z) \land P(y, z)) \rightarrow x = y) \]

\[ \models (P(a, c) \land P(b, d) \land P(c, e) \land P(d, e)) \rightarrow a = b \]

where a, b, c, d, e are constant symbols.
Resolution Example

We enter this problem into Prover9, an automated reasoning system, as:

Assumptions:
all x all y all z ((p(x,z) & p(y,z)) -> x = y).

Goals:
(p(a,c) & p(b,d) & p(c,e) & p(d,e)) -> a = b.
Resolution Example

Here’s how it looks. Then we press *Start*. 
Resolution Proof Generated

% Comments from original proof -------
% Proof 1 at 0.00 (+ 0.00) seconds.
% Length of proof is 12.
% Level of proof is 4.
% Maximum clause weight is 9.
% Given clauses 6.

1 (all x all y all z (p(x,z) & p(y,z) -> x = y)) # label(non_clause). [assumption].
2 p(a,c) & p(b,d) & p(c,e) & p(d,e) -> a = b # label(non_clause) # label(goal). [goal].
3 ~p(x,y) | ~p(z,y) | z = x. [classify(1)].
4 p(a,c). [deny(2)].
5 p(b,d). [deny(2)].
6 p(c,e). [deny(2)].
7 p(d,e). [deny(2)].
8 b != a. [deny(2)].
9 d = c. [hyper(3,a,e,b,7,0)].
10 p(b,c). [back_rewrite(5), rewrite([9(2)])].
11 ~p(b,c). [ur(3,c,4,o,c,8,0)].
12 $$. [resolve(11,c,10,o)].

----------------------------------------------- end of proof -------------------------------
What just happened?

Prover9 was successful at proving the result:

• The assumption and goal were first converted into **clausal form**.
• The clausal form was proved by **resolution**.
• The steps in resolution are displayed.
Clausal Form Translation

\[(\forall x \forall y \forall z \ (p(x,z) \land p(y,z) \rightarrow x = y))\]

becomes (clausal form):
\[-p(x,y) \lor -p(z,y) \lor z = x.\]

Negated
\[p(a,c) \land p(b,d) \land p(c,e) \land p(d,e) \rightarrow a = b\]
becomes (clausal form):
\[p(a,c).\]
\[p(b,d).\]
\[p(c,e).\]
\[p(d,e).\]
\[b \neq a.\]
Resolution-Based Proof Steps

- $d = c$  
  Hyperresolution  
  (combines several resolution steps into one)
- $p(b, c)$.  
  Rewriting (equality rule)
- $\neg p(b, c)$.  
  Resolution
- $\bot$  
  Original clause set refuted.
History

- Resolution was introduced by Prof. J. Alan Robinson in 1965 at Syracuse U.

- There were previous hints at it by Dag Prawitz (1960, for the function-free case) and Herbrand (1930’s).
Recent

- Resolution is used for fully automated theorem-proving and reasoning systems.
  
  
  - [http://www.cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP](http://www.cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP)
How resolution works

- A logic formula is first negated, then converted into “clausal form”. (Significant logic is “wired into” this step.)

- In clausal form, **quantifiers** have been eliminated.

- The clausal form is proved by **refutation**, i.e. showing that its negation is unsatisfiable.
Two Types of Resolution

- Predicate calculus resolution:
  - Our main objective

- Propositional resolution:
  - Needed to understand predicate resolution
  - Big role in algorithms and complexity theory
    (NP completeness, for example)
Propositional Form of Resolution

- A **literal** is a proposition symbol or its negation.

- A **clause** is a disjunction of literals.

- The **negation** of the formula to be proved is first converted to a **clause set**, effectively a **conjunction** of those clauses.

- The original formula is a theorem iff the set of clauses is **not satisfiable**.
Resolution Schematic

- Formula
  - negate
  - clausify
- Negated Formula
- Clauses
  - resolvents added back to clauses

iff original formula is valid
Example Clause Set

- Clause set:
  - $p \lor \neg q$
  - $\neg q \lor \neg r$
  - $q$

- This clause set is **satisfiable**:
  - Valuation $p = 1$, $q = 1$, $r = 0$ will satisfy it.
Example Clause Set

- **Clause set:**
  - \( p \lor \neg q \)
  - \( q \lor r \)
  - \( \neg p \)
  - \( \neg r \)

- **This clause set is **unsatisfiable**:**
  - There is **no valuation** that makes all clauses \( T \) at the same time.

  - Why not? We’d need \( p = r = 0 \) in order to satisfy all clauses, but then there is no way to set \( q \) so that all clauses are satisfied.
Conjunctive Normal Form (CNF)

- A clause set is another way of representing a propositional formula.

- A formula is in conjunctive normal form iff it is a conjunction of disjunctions of literals (an “and” of “ors” of propositions and their negations)
CNF Example

\((p \lor \neg q \lor r) \land (\neg s) \land (q \lor r \lor s)\)

Three clauses, containing 3, 1, and 3 literals, respectively.
Sets

- Throughout this discussion, we treat each **clause** as a mathematical **set**, meaning that duplicates literals are eliminated.

- Likewise, a set of clauses (i.e. CNF) has no clause duplicated.
Example

- Clause set:
  - \( p \lor \neg q \)
  - \( q \lor r \)
  - \( \neg p \)
  - \( \neg r \)

- is representable as a set of sets:

\[
\{\{p, \neg q\}, \{q, r\}, \{\neg p\}, \{\neg r\}\}
\]

where the \( \lor \) connective is implicit \textit{within} clauses, and the \( \land \) connective is implicit \textit{among} them.
Equivalence of Propositional Formulas

- Two propositional formulas (including clauses) are **equivalent** (≡) iff they are satisfied by the same valuations.

- Examples:
  - $p \land \neg q \equiv \neg (q \lor \neg p)$
  - $p \rightarrow (q \rightarrow r) \equiv (p \land q) \rightarrow r$
Equivalence of Clause Sets

- Two clause sets are called equivalent if they are satisfied by the same set of valuations.

- In particular, if two clause sets are equivalent, they are either:
  - both satisfiable, or
  - both unsatisfiable
How General is the Clausal Form?

- **Claim:** Every propositional formula can be represented in clausal form.
- **Examples:**
  - \( p \lor q \) in clausal form is \( \{p \lor q\} \) (one clause)
  - \( p \land q \) in clausal form is \( \{p, q\} \) (two clauses)
  - \( p \rightarrow q \) in clausal form is \( \{\neg p \lor q\} \) (one clause)
  - \( p \leftrightarrow q \) in clausal form is \( \{\neg p \lor q, p \lor \neg q\} \) (two clauses)
Extreme Cases

- $\emptyset$ or $\{\}$, the empty set of clauses, is equivalent to $1$(true)
  
  (as $1$ is the identity for the $\land$ operation, and a set of clauses is a conjunction).

- Note: Any valuation satisfies every clause in $\{\}$, because there are no clauses to satisfy.
The Empty Clause

- The **empty clause** is $\bot$, sometimes denoted by an empty box $\Box$, and is equivalent to False.

- Do not confuse the empty clause with an empty *set* of clauses $\{}$.

- Observation: Any clause set containing the empty clause is unsatisfiable, because no valuation can make $\bot$ true.

Example: $\{\neg p \lor q, p, \bot\}$
Conversion to Clausal Form

- We rely on rules that can be proved in propositional logic:
  - Commutative
    - $A \land B \equiv B \land A$
    - $A \lor B \equiv B \lor A$
  - Distributive
    - $A \land (B \lor C) \equiv (A \land B) \lor (A \land C)$
    - $A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)$
  - DeMorgan
    - $\neg (A \land B) \equiv \neg A \lor \neg B$
    - $\neg (A \lor B) \equiv \neg A \land \neg B$
Conversion to Clausal Form

1. Replace each $\varphi \rightarrow \psi$ with $(\neg \varphi \lor \psi)$.

2. Replace each $\varphi \leftrightarrow \psi$ with $(\neg \varphi \lor \psi) \land (\varphi \lor \neg \psi)$.

3. Push $\neg$ inward, toward proposition symbols:
   - Replace $\neg(\varphi \land \psi)$ with $(\neg \varphi \lor \neg \psi)$.
   - Replace $\neg(\varphi \lor \psi)$ with $(\neg \varphi \land \neg \psi)$.
   - Replace $\neg \neg \varphi$ with $\varphi$.

4. Push $\lor$ inward toward literals:
   - Replace $\chi \lor (\varphi \land \psi)$ with $(\chi \lor \varphi) \land (\chi \lor \psi)$.
   - Replace $(\varphi \land \psi) \lor \chi$ with $(\varphi \lor \chi) \land (\psi \lor \chi)$. 
Example of Conversion to Clauses

- $\neg(p \rightarrow (\neg q \land (r \land \neg s)))$ replace $\rightarrow$
- $\neg(\neg p \lor (\neg q \land (r \land \neg s)))$ push $\neg$ inward
- $\neg\neg p \land \neg (\neg q \land (r \land \neg s))$ delete $\neg\neg$
- $p \land (\neg\neg q \lor \neg (r \land \neg s))$ push $\neg$ inward
- $p \land (q \lor \neg (r \land \neg s))$ push $\neg$ inward
- $p \land (q \lor \neg r \lor \neg\neg s)$ delete $\neg\neg$
- $p \land (q \lor \neg r \lor s)$ conjuncts are clauses
- $\{p, \quad q \lor \neg r \lor s\}$
Example of Conversion to Clauses

- \( \neg(p \land (\neg q \lor (r \land \neg s))) \)  push \( \neg \) inward
- \( \neg p \lor \neg (\neg q \lor (r \land \neg s)) \)  push \( \neg \) inward
- \( \neg p \lor (\neg \neg q \land \neg (r \land \neg s)) \)  delete \( \neg \neg \)
- \( \neg p \lor (q \land \neg (r \land \neg s)) \)  push \( \neg \) inward
- \( \neg p \lor (q \land (\neg r \lor \neg \neg s)) \)  delete \( \neg \neg \)
- \( \neg p \lor (q \land (\neg r \lor s)) \)  distribute \( \lor \) over \( \land \)
- \((\neg p \lor q) \land (\neg p \lor (\neg r \lor s)) \)  flatten \( \lor \)
- \((\neg p \lor q) \land (\neg p \lor r \lor s) \)
- \{ (\neg p \lor q), (\neg p \lor \neg r \lor s) \}  conjuncts are clauses
Try this one

- \((\neg p \land (\neg q \to (r \leftrightarrow s)))\)
Code for CNF conversion

- **Python**
  http://aima-python.googlecode.com/svn/trunk/logic.py

- **Prolog**
  http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/logic_topics/normal_forms/normal_form.html
Clausal Form from a Truth Table

- A truth table is typically thought of as displaying one of possibly many **disjunctive normal forms** (DNF).

- Each row is a **min-term**:

\[ p_1^* \land p_2^* \land \ldots \land p_n^* \] where \( p_i^* \) indicates a **literal**.

- The overall truth-function is a **disjunction** of min-terms:

\[ \lor (p_1^* \land p_2^* \land \ldots \land p_n^*) \] over the rows for which the result is **T**.

- If there are no rows with result **T**, the DNF represents False.

- Minterms are **additive**.
**Example: Truth Table**

<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>q</th>
<th>r</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

\[ A \text{ DNF: } \neg p \land q \land \neg r \lor \neg p \land q \land r \lor p \land \neg q \land \neg r \lor p \land q \land \neg r \lor p \land q \land r \]
Clausal Form from Truth Table

- A DNF has the value **True** exactly when at least one of the disjuncts does, e.g.
  \((-p \land q \land r) \lor (-p \land q \land r) \lor \ldots\)

- Similarly a CNF has the value **False** exactly when at least one of the **maxterms** does, e.g.
  \((p \lor q \lor r) \land (p \lor q \lor -r) \land (-p \lor q \lor -r)\)

- Hence we can “read off” a CNF from the **False** rows of the table by **inverting** the sense of the variables (courtesy of deMorgan’s law).

- The True case where no rows are False.

- Maxterms are **subtractive**.
Example: Clausal Form from Truth Table

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>q</td>
<td>r</td>
<td>value</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

\[ (p \lor q \lor r) \land (p \lor q \lor \neg r) \land (\neg p \lor q \lor \neg r) \]

**A CNF:** \( (p \lor q \lor r) \land (p \lor q \lor \neg r) \land (\neg p \lor q \lor \neg r) \)
Summary

- In a DNF (such as minterm form) a term (conjunction) is trying to make the function “more True”. (The empty case is False).

- In CNF (such as maxterm form) a term (disjunction) is trying to make the function “more False”. (The empty case is True).
Housekeeping: Reduced Clause Sets

- A clause set is **reduced** provided:
  - No literal occurs multiple times in any clause.
    - $p \lor \neg q \lor p$ is disallowed in a reduced set.
  - No clause contains a literal and its negation.
    - $p \lor q \lor \neg p$ is disallowed in a reduced set.

- Any clause set $S$ is equivalent to a reduced set $\text{reduce}(S)$:
  - Replace multiple occurrences of a literal with a **single occurrence** of the literal.
  - **Drop** any clauses containing a literal and its negation.
    (Such clauses are equivalent to $T$, and do thus do not affect satisfiability of the set of clauses.)
  - Replace multiple occurrences of a clause (as a **set**) with a single occurrence.
reduce example

\[
\text{reduce} \left( \{ p \lor \neg q \lor p, \\
p \lor q \lor \neg p \lor q \} \right) = \\
\{ p \lor \neg q \}
\]
Resolution Method

- **Input**: A reduced set of clauses.

- **Output**: A set of clauses equivalent to the input set, such that the original set is unsatisfiable iff the final set contains the empty clause $\bot$.

- There is no valuation that satisfies $\bot$ (much less $\bot$ together with other clauses).
How Resolution Works

- Do Repeatedly, until no further steps can be taken:
  - From the set of clauses, pick a pair from which a new clause, called the “resolvent”, can be created. (Must resolve the pair to find this out.)

  - Add the resolvent to the set.

  - If $\bot$ is ever added to the set, stop. The original set of clauses is unsatisfiable.

- Conversely, if the original set of clauses is unsatisfiable, it is possible to eventually derive $\bot$. 

What is the Resolvent?

• Suppose $p$ is a proposition symbol.

• If the set contains clauses of both forms
  • $p \lor \varphi$
  • $\neg p \lor \psi$

  where $\varphi$ and $\psi$ are clauses (either could be empty), then the resolvent is the reduced version of

  $$\varphi \lor \psi.$$ 

• $p$ and $\neg p$ are said to be “clashing” literals.
Resolution as a Deduction Rule

\[ p \lor \varphi \quad \rightarrow p \lor \psi \]

\[ \varphi \lor \psi \quad \text{(in reduced form)} \]

where \( p \) is any proposition symbol and \( \varphi \) and \( \psi \) are clauses (possibly empty).
Example of Resolvents

- Consider the clauses
  - $p \lor \neg q \lor \neg s$
  - $q \lor r \lor \neg s$

- A resolvent (based on literals $q$, $\neg q$) is:
  - $p \lor r \lor \neg s$
Example of Resolvents

- Consider the clauses
  - $p \lor r$
  - $\neg r$
- The resolvent is:
  - $p$
Example of Resolvents

- Consider the clauses
  - p
  - \( \neg p \)

- Since p and \( \neg p \) occur in different clauses, the resolvent is:
  - \( \bot \)
Example of Resolvents

- Consider the clauses
  - $p \lor \neg q \lor r$
  - $q \lor \neg r \lor \neg s$
- One resolvent (based on literals $q, \neg q$) is:
  - $p \lor r \lor \neg r \lor \neg s$
- Another (based on literals $r, \neg r$) is:
  - $p \lor q \lor \neg q \lor \neg s$
- Both of these would be **dropped** in reducing, however, since each contains a literal and its negation.
Resolution Algorithm

• Start with a set $S$ of reduced clauses.

• while $S$ does not contain $\bot$ and the following step adds something new to $S$:
  
  • Add to $S$ the resolvent $R$ of any two clauses such that $R$ is not already in $S$ and the resolvent does not contain complementary literals.

• The original $S$ is unsatisfiable iff $\bot$ is in $S$. 
Unit Clauses

• A clause with exactly one literal is called a unit clause.

• The ultimate step in resolving to ⊥ will be to resolve two unit clauses.

• Resolving a unit clause with a clause having $n > 0$ literals results in a clause with fewer than $n$ literals.
Unit Preference Strategy

• Preferring unit clauses is a good heuristic.
Example 1 (Highlighting unit clauses)

- $S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg r\}$
  resolve $p \lor \neg q$ with $\neg p$, adding $\neg q$ to $S$.

- $S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg r, \ \neg q\}$
  resolve $q \lor r$ with $\neg q$, adding $r$ to $S$.

- $S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg r, \ \neg q, \ r\}$
  resolve $\neg r$ with $r$ adding $\bot$ to $S$.

- Stop $\bot \in S$.

- The original $S$ is unsatisfiable, as $\bot \in S$. 
Example 2

- $S = \{p \lor \neg q, \ q \lor r, \ \neg p\}$
  Resolve $p \lor \neg q$ with $\neg p$, adding $\neg q$ to $S$.

- $S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg q\}$
  Resolve $q \lor r$ with $\neg q$, adding $r$ to $S$.

- $S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg q, \ r\}$
  Resolve $p \lor \neg q$ with $q \lor r$, adding $p \lor r$ to $S$.

- $S = \{p \lor \neg q, \ q \lor r, \ \neg p, \ \neg q, \ r, \ p \lor r\}$
  Stop. No new resolvents are possible. The original set is satisfiable, as $\bot \not\in S$. 
**Soundness:** Any valuation satisfying both $p \lor \varphi$ and $\neg p \lor \psi$ satisfies $\varphi \lor \psi$.

- Suppose $\nu$ satisfies both $p \lor \varphi$ and $\neg p \lor \psi$.
- Either $\nu(p) = T$ or $\nu(p) = F$.
- If $\nu(p) = T$, then $\nu(\neg p) = F$. In order to satisfy $\neg p \lor \psi$ then, $\nu(\psi) = T$. Thus $\nu(\varphi \lor \psi) = T$.
- If $\nu(p) = F$, in order to satisfy $p \lor \varphi$, $\nu(\varphi) = T$. Thus $\nu(\varphi \lor \psi) = T$.
- Thus adding the resolvent preserves the valuations that satisfy the set of clauses.
Completeness

• Completeness is more complicated and we will not prove it here.

• We’d have to show that if a set is unsatisfiable, there is a set of resolution steps that result in the empty clause.

Resolution Algorithm Termination (propositional case)

- Closure is always achievable.

- The set of distinct reduced clause sets for a given set of proposition symbols is finite.

- At worst, every possible clause (regarding reordering of symbols as equivalent) will be generated.

- How many distinct clauses can there be for n proposition symbols?
Resolution in tabular form

1. \( p \lor \neg q \)  
   Premise
2. \( q \lor r \)  
   Premise
3. \( \neg r \)  
   Premise
4. \( \neg p \)  
   Premise
5. \( q \)  
   Resolution 2, 3
6. \( p \)  
   Resolution 1, 5
7. \( \bot \)  
   Resolution 6, 4
Resolution as a Tree

$p \lor \neg q$
$q \lor r$
$\neg r$
$\neg p$

children nodes are resolvents
Try resolving these clause sets:

- $$\neg p \lor \neg q \lor \neg r,$$
  $$\neg q \lor r,$$
  $$q \lor s,$$
  $$\neg s,$$
  $$\neg s,$$
  $$p$$

- $$p \lor \neg q \lor r,$$
  $$q \lor r,$$
  $$\neg r,$$
  $$\neg p$$
Sometimes a DAG is more appropriate than a tree for showing all options.

We avoid identifying the two ⊥ nodes, so as not to confuse the two sets of antecedents.
Useful Resolution Short-cuts

• Uncomplemented Literal Lemma (also called the “Purity Rule”)

If a literal appears in one or more clauses, but its complement appears in no clause, then every clause containing that literal can be deleted from the set without changing satisfiability of the clause set.

• **Rationale**: The literal in question can be assigned T without loss of generality, thus clauses containing it cannot affect satisfiability.
Example of Uncomplemented Literal Lemma

\[-p \lor q \lor r,\]
\[-q \lor r,\]
\[q \lor s,\]
\[-s,\]
\[p\]

- r occurs only uncomplemented.
- The clause set is satisfiable iff the following set is:
  \[q \lor s,\]
  \[-s,\]
  \[-s,\]
  \[p\]

- and this set is satisfiable iff \(-s\) is satisfiable (which it is).
Further Resolution Short-Cuts

• **Unit Clause Lemma:**

  If a *unit* cause (clause with only one literal L) exists within the set, the following operation may be performed without affecting satisfiability:

  • Remove all clauses containing the literal L.
  • Remove the complement of L from all remaining clauses.

• **Rationale:** The literal in question **must** be assigned T in a satisfying interpretation. Hence all clauses containing it will be T and contribute nothing to the set. Likewise, its complement must be assigned F, and thus contribute nothing to the individual clauses.
Example of Unit Clause Lemma

- $\neg p \lor q \lor r$
  
  $q \lor s$
  
  $\neg s$
  
  $p \lor \neg s$

- $\neg s$ is a unit clause. The complement of $\neg s$ is $s$.

- The clause set is unsatisfiable iff the following set is:
  
  $\neg p \lor q \lor r,$
  
  $q,$
  
  (formerly $q \lor s$)

  ($\neg s$ and $p \lor \neg s$ were removed.)

  Rationale: If the set is satisfiable, $\neg s$ must be assigned T.
DPLL

The previous two edit rules are the basis of another algorithm for satisfiability: DPLL for "Davis-Putnam-Logemann-Loveland"

Further Useful Optimizations

Subsumption Lemma:

• One clause \textit{subsumes} another if the former’s literals are a \textit{subset} of the latter’s.

• If one clause of a set subsumes another, the \textit{subsumed} clause (the larger one) can be \textit{dropped} from the set.

• \textbf{Rationale}: If C subsumes D, then any valuation satisfying C must also satisfy D (because the literals are disjoined). Thus the satisfiability of the set of clauses is unaffected if D is removed.
Example of Subsumption Lemma

- \( \lnot p \lor q \lor \lnot r, \)
  \( \lnot p \lor \lnot r, \)
  \( p \lor r \lor q \)

- The second clause subsumes the first.

- The clause set is satisfiable iff the following set is:
  \( \lnot p \lor \lnot r, \)
  \( p \lor r \lor q \)
Common Special Case of Clause Set

- Often we want to prove a sequent such as:
  - \( \phi_{11} \land \phi_{12} \land \ldots \land \phi_{1m_1} \rightarrow \psi_1 \),
  - \( \phi_{21} \land \phi_{22} \land \ldots \land \phi_{2m_2} \rightarrow \psi_2 \),
  - \ldots
  - \( \phi_{n1} \land \phi_{n2} \land \ldots \land \phi_{nm_n} \rightarrow \psi_n \)
  \[
  \vdash \chi_1 \land \chi_2 \land \ldots \land \chi_p
  \]

  where each symbol represents a literal.

- This can be done by showing that the following clause set is unsatisfiable:

  \[
  \{ \neg \phi_{11} \lor \neg \phi_{12} \lor \ldots \lor \neg \phi_{1m_1} \lor \psi_1, \\
  \neg \phi_{21} \lor \neg \phi_{22} \lor \ldots \lor \neg \phi_{2m_2} \lor \psi_2, \\
  \ldots \\
  \neg \phi_{n1} \lor \neg \phi_{n2} \lor \ldots \lor \neg \phi_{nm_n} \lor \psi_n, \\
  \neg \chi_1 \lor \neg \chi_2 \lor \ldots \lor \neg \chi_p \}
  \]
Strategic Optimizations

- **Unit-Preference**: Prefer resolving with unit clauses. These reduce the size of resulting clauses.

- **Set-of-Support**: Divide the clauses into two sets:
  - A *known-satisfiable* subset
  - Other (called the “set of support” SOS)

- Always resolve with an SOS clause or a clause derived from an SOS clause.
Set-of-Support

- Showing that the following clause set is **unsatisfiable**:

\[
\{ \neg \varphi_{11} \lor \neg \varphi_{12} \lor \ldots \lor \neg \varphi_{1m_1} \lor \psi_1, \\
\neg \varphi_{21} \lor \neg \varphi_{22} \lor \ldots \lor \neg \varphi_{1m_2} \lor \psi_2, \\
\ldots \\
\neg \varphi_{n1} \lor \neg \varphi_{n2} \lor \ldots \lor \neg \varphi_{nm_n} \lor \psi_n, \\
\neg \chi_1 \lor \neg \chi_2 \lor \ldots \lor \neg \chi_p \}
\]

Satisfiable "axioms"  
Initial set of support
Horn Clauses

• A Horn clause is one in which there is **at most one non-negated** literal:
  - \( \neg \varphi_1 \vee \neg \varphi_2 \vee \ldots \vee \neg \varphi_m \vee \psi \) (one non-negated)
  or
  - \( \neg \varphi_1 \vee \neg \varphi_2 \vee \ldots \vee \neg \varphi_m \) (no non-negated)

• Horn clause are the basis of the **Prolog** language, where:
  \( \neg \varphi_1 \vee \neg \varphi_2 \vee \ldots \vee \neg \varphi_m \vee \psi \)
  is written
  \( \psi : - \varphi_1, \varphi_2, \ldots \varphi_m. \)
  interpreted as
  \( \varphi_1 \land \varphi_2 \land \ldots \land \varphi_m \rightarrow \psi \)
  If \( m = 0 \), then we just write \( \psi \).
Prolog uses a special case of resolution to do its work (“SLD” = Selective Linear Definite resolution)

\[
\{ p \lor \neg r \lor \neg s, \\
  r \lor \neg q, \\
  s \lor \neg q, \\
  q, \\
  \neg p, \\
\}\]

becomes in Prolog syntax:

- \( p :\neg r, s. \)
- \( r :\neg q. \)
- \( s :\neg q. \)
- \( q. \)
- \(?- p. \)

Dialog with Prolog:

```prolog
consult(user).
p :\neg r, s.
r :\neg q.
s :\neg q.
q.
^D
l ?- p.
yes
```
Resolution Theorem Provers

- Prolog cannot handle general negation

- Resolution theorem provers can

- Examples: Prover9, Vampire, ...
Prover9

- Extends the former program “Otter”
- Developed at Argonne National Laboratory
- Free download for all platforms
  - http://www.cs.unm.edu/~mccune/prover9/
- Also includes “Mace” for finding counterexamples
Prover9 GUI: \(-\) is "not" \\
\(\mid\) is "or"
Prover9 Proof ($F$ is empty clause)
Resolution for Predicate Logic

- *Predicate* Clausal Form:
  - A literal is an atomic formula or its negation (instead of a proposition symbol or its negation).

- The variables of each clause are each implicitly $\forall$-quantified.

- The variables of each clause are thus independent from the other clauses; even if they are the same, they should be thought of as being different (e.g. implicitly rename by indexing with a clause number).
Example: Predicate Clausal Form

- Clause set \{p(X), q(X, Y), \neg q(X, X) \lor p(X)\} stands for the conjunction

- \forall X p(X)
  \land \forall X \forall Y q(X, Y)
  \land \forall X \forall Y (\neg q(X, X) \lor p(X))

which is the same as

- \forall X_1 p(X_1)
  \land \forall X_2 \forall Y_2 q(X_2, Y_2)
  \land \forall X_3 \forall Y_3 (\neg q(X_3, X_3) \lor p(X_3))

i.e. the clause set

- \{p(X_1), q(X_2, Y_2), \neg q(X_3, X_3) \lor p(X_3)\}
How General is This?

- Completely general, as far as showing unsatisfiability is concerned.
Examples of Predicate Clausal Form

- \neg \text{human}(X) \lor \text{mortal}(X)
- \text{human}(\text{socrates})
- \neg \text{mortal}(\text{socrates})

- These clauses can be used to prove the syllogism:
  - All humans are mortal.
  - Socrates is a human.
  - Therefore Socrates is mortal.
Resolution for Predicate Clauses

• To resolve *predicate* clauses, it is no longer sufficient to look for just a literal and its negation in two distinct clauses
  \[ \neg q(X, X) \lor p(X) \]
  \[ \neg p(Z) \lor r(Z, Y) \]
• For one thing, the identity of the **variables** is independent in each.
• For another, the arguments are generally **terms**, not just simple variables:
  \[ \neg q(X, X) \lor p(f(X)) \]
  \[ \neg p(X) \lor r(g(X), c) \]
Example of What Resolution Must Do

• Suppose we have derived three formulas (where c is a constant symbol):
  • \( p(c) \)
  • \( \forall X (p(X) \rightarrow q(f(X)) \)
  • \( \forall X (q(X) \rightarrow r(X, g(X)) \)

• We would expect to be able to infer
  • \( q(f(c)) \)
  • \( r(f(c), g(f(c))) \)

• Resolution must be able to handle such things.
Equivalent Clausal Form

- The clausal form of
  - $p(c)$
  - $\forall X (p(X) \rightarrow q(f(X)))$
  - $\forall X (q(X) \rightarrow r(X, g(X))$ is
    - $\{p(c), \neg p(X) \lor q(f(X)), \neg q(X) \lor r(X, g(X))\}$

- Resolution has to “make a connection” between $p(c)$ and $p(X)$, and between $q(f(X))$ and $q(X)$. 
Unification

• The “connection” alluded to on the previous slide is known as **unification**.

• Two **atoms** are **unifiable** if there is a uniform **set of substitutions** of terms for their variables that makes them **identical**.

• If such a substitution set exists, it is **applied to all** literals in the formulas prior to resolution.
Unification Examples

- Consider atoms $p(c)$, $p(X)$ ($c$ is a constant, $X$ a variable).

- These terms are **unifiable**, since the substitution $[c/X]$ (substitute $c$ for $X$) makes them identical.
Unification Examples

- Consider $q(c, d), q(X, X)$ (c and d are constants, X a variable).

- These terms are **not unifiable**.

- Distinct *constant symbols do not unify*. There is no substitution that will make them identical.

- (Note: This is not the same as saying constant symbols cannot be equated. They can, with a separate equation such as $c = d$. *Equality is handled separately.*)
Renaming Apart

• Consider \( p(X, f(a)) \) vs. \( p(g(Y), f(X)) \)

• These might appear not to unify, since we would have a conflict \([g(Y)/X]\) vs. \([a/X]\).

• However, if we **rename** the variables in the second clause we get:
  \( p(X, f(a)) \) vs. \( p(g(Z), f(W)) \).

• These unify, using \([g(Z)/X, a/W]\).

• **Note**: Renaming apart is done only at the **start** of considering unification of two clauses, and all variables in the clause are renamed **uniformly**.
Notation for Variable Substitutions

- In general, a substitution consists of a set of bindings of variables to terms, e.g.
  \[ \beta = [Z/X, f(Z, c)/Y, c/W] \]

- If \( \tau \) is a term, then \( \tau\beta \) denotes the result of making the substitutions \( \beta \) in for variables in \( \tau \), e.g.
  
  if \( \tau = p(X, g(Y, W)) \), then \( \tau\beta = p(Z, g(f(Z, c), c)) \)
Composing Variable Substitutions

- If $\beta$ and $\gamma$ are substitutions and $\tau$ is a term, then $(\tau\beta)\gamma$ denotes the result of first applying $\beta$ to $\tau$, then $\gamma$ to the result, e.g.
  \[
  \tau = p(X, g(Y, W)) \quad \text{literal} \\
  \beta = \{Z/X, f(Z, c)/Y, c/W\} \quad \text{sub} \\
  \gamma = \{V/Z\} \quad \text{sub} \\
  (\tau\beta)\gamma = p(V, g(f(V, c), c))
  \]

- The **composition** $\beta\gamma$ of substitutions $\beta$ and $\gamma$ is the substitution such that for all terms $\tau$
  \[\tau(\beta\gamma) = (\tau\beta)\gamma\]
  e.g. $[V/X, f(V, c)/Y, c/W]$ above
Unifiers

- A set of substitutions that unifies two literals is called a **unifier**.
More Unification Examples

<table>
<thead>
<tr>
<th>Term 1</th>
<th>Term 2</th>
<th>Unifier, if any?</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(X, X)</td>
<td>p(f(Y), f(Z))</td>
<td></td>
</tr>
<tr>
<td>p(X, X)</td>
<td>p(f(Y), g(Y))</td>
<td></td>
</tr>
<tr>
<td>p(X, Y)</td>
<td>p(Z, f(Z))</td>
<td></td>
</tr>
<tr>
<td>p(X, f(X))</td>
<td>p(g(Y), W)</td>
<td></td>
</tr>
<tr>
<td>p(X, f(X))</td>
<td>p(f(Y), Y)</td>
<td></td>
</tr>
</tbody>
</table>
## Even More Unification Examples

<table>
<thead>
<tr>
<th>Term 1</th>
<th>Term 2</th>
<th>Unifier, if any?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p(X, Y)</code></td>
<td><code>p(f(Z), g(Z))</code></td>
<td></td>
</tr>
<tr>
<td><code>p(X, f(X))</code></td>
<td><code>p(f(Z), U)</code></td>
<td></td>
</tr>
<tr>
<td><code>p(f(X), g(X))</code></td>
<td><code>p(f(U), U)</code></td>
<td></td>
</tr>
<tr>
<td><code>p(f(X), f(X))</code></td>
<td><code>p(c, c)</code></td>
<td></td>
</tr>
<tr>
<td><code>p(f(X), g(X))</code></td>
<td><code>p(Y, g(Y))</code></td>
<td></td>
</tr>
</tbody>
</table>
Most General Unifiers (mgu)

- If two literals unify at all, they have a “most general unifier”, one which adds no unneeded constraints.

- Example: p(X) vs. p(f(Y)) could be unified with the substitution [f(c)/X, c/Y].

- However, this would not be the most general, since we could leave Y as a variable: [f(Y)/X] and each of the original literals would unify under this substitution.
Generality of Substitutions

- Substitution $\beta$ is **as general as** substitution $\nu$ if there is a $\gamma$ such that $\nu = \beta \gamma$.

- To say that $\beta$ is a “most general unifier” means that is as general as *any* unifier.
Find the MGU or indicate non-unifiable

<table>
<thead>
<tr>
<th>Term 1</th>
<th>Term 2</th>
<th>MGU?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p(X, Y))</td>
<td>(p(Z, Z))</td>
<td></td>
</tr>
<tr>
<td>(p(X, c))</td>
<td>(p(Y, Y))</td>
<td></td>
</tr>
<tr>
<td>(p(f(X), Y))</td>
<td>(p(W, f(Z)))</td>
<td></td>
</tr>
<tr>
<td>(p(f(X), Y))</td>
<td>(p(Z, Y))</td>
<td></td>
</tr>
<tr>
<td>(p(f(Z), g(X)))</td>
<td>(p(Y, g(Y)))</td>
<td></td>
</tr>
</tbody>
</table>
MGU Algorithm: Basic Idea

- Given a pair of terms, decompose pairwise by descending through arguments.
- During descent, a substitution is accumulated, until the process is complete or a unification conflict is found.
- If a pair of arguments is identical, continue.
- If one argument is a variable, substitute the other argument for it and record in the substitution (composing with accumulated substitution).
- If arguments are two different constants, fail.
- If arguments have two different function symbols, fail.
- Otherwise add the arguments of each function to the stack.
MGU Algorithm (Martelli & Montanari)

- **Input**: Two terms, or two atoms, $\tau_1$, $\tau_2$, *already renamed apart*.
- **Output**: Either the most general unifier for $\tau_1$, $\tau_2$, or “not unifiable”.

- $S := \{[\tau_1, \tau_2]\}$; // functions as a sort-of stack
  - $\mu$ := the empty substitution;
  - while( $S \neq \emptyset$ )
    - remove a pair [L, R] from S;
    - if( L = R )
      - do nothing;
    - else if( L=f(s_1, s_2, ..., s_n) and R=f(t_1, t_2, ..., t_n) ) // same f, n (2)
      - $S := S \cup \{[s_1, t_1], [s_2, t_2], ... [s_n, t_n]\}$; // pushes
    - else if( L = x where x is a variable not occurring in R) (3)
      - $\mu := \mu[R/x]$;
      - $S := \text{applytoallpairs}([R/x], S)$;
    - else if( R = x where x is a variable not occurring in L) (4)
      - $\mu := \mu[L/x]$;
      - $S := \text{applytoallpairs}([L/x], S)$;
    - else return “not unifiable”;
  - return $\mu$ as the MGU;

see also [http://en.wikipedia.org/wiki/Unification_(computer_science)]
Intuitive Unification

• Remember when two things don’t unify:
  • Distinct constant symbols don’t unify.
  • Terms with outermost function symbols that are distinct don’t unify.
  • A term with an outermost function symbol doesn’t unify with a constant.
  • Two terms with the same outermost function symbol don’t unify if some of their arguments don’t pairwise unify.

• Remember that substitutions are cumulative during unification.
Example

- $p(X, f(X))$ vs. $p(Y, f(Y))$  
  **initial**
- $S := \{[p(X, f(X)), p(Y, f(Y))]\}$
- $\mu := []$

- Remove $[p(X, f(X)), p(Y, f(Y))]$  
  **case 2**
- $S := \{[X, Y], [f(X), f(Y)]\}$

- Remove $[X, Y]$  
  **case 3**
- $\mu := [Y/X]; S := \{[f(Y), f(Y)]\}$

- Remove $[f(Y), f(Y)]$  
  **case 1**
- $S := \{\}$

- Result: unifiable with mgu $[Y/X]$
Diagrammatically

- \( p(X, f(X)) \)
  \[ \uparrow \]
  \( p(Y, f(Y)) \)

- substitution \([Y/X]\)

- \( p(Y, f(Y)) \)
  \[ \uparrow \]
  \[ \uparrow \]
  \( p(Y, f(Y)) \)
Example

- \( p(X, f(X)) \) vs. \( p(f(Y), Y) \)  
  initial
- \( S := \{[p(X, f(X)), p(f(Y), Y)]\} \)
- \( \mu := {} \)

Remove \( [p(X, f(X)), p(f(Y), Y)] \)  
- \( S := \{[X, f(Y)], [f(X), Y]\} \)  
  case 2

Remove \( [X, f(Y)] \)  
- \( \mu := [f(Y)/X]; S := \{[f(f(Y)), Y]\} \)  
  case 3

Remove \( [f(f(Y)), Y] \)  
- Result: not unifiable  
  case 5
Diagrammatically

- $p(X, f(X))$
  
  $\uparrow\downarrow$

  $p(f(Y), Y)$

  substitution $[f(Y)/X]$

- $p(f(Y), f(f(Y)))$
  
  $\uparrow\downarrow$

  $p(f(Y), Y)$

  occur check fails, not unifiable
Example

- $p(X, g(Z), X)$ vs. $p(f(Y), Y, W)$
- $S := \{[p(X, g(Z), X), p(f(Y), Y, W)]\}$
- $\mu := \{\}$

Remove $[p(X, g(Z), X), p(f(Y), Y, W)]$
- $S := \{[X, f(Y)], [g(Z), Y], [X, W]\}$

Remove $[X, f(Y)]$
- $\mu := [f(Y)/X]; S := \{[g(Z), Y], [f(Y), W]\}$

Remove $[g(Z), Y]$
- $\mu := [f(g(Z))/X, g(Z)/Y]; S := \{[f(g(Z)), W]\}$

Remove $[f(g(Z)), W]$
- $\mu := [f(g(Z))/X, g(Z)/Y, f(g(Z))/W]; S := \{\}$

Result: unifiable with mgu $[f(g(Z))/X, g(Z)/Y, f(g(Z))/W]$
Diagrammatically

- \( p(X, g(Z), X) \) vs. \\
  \[ \uparrow \]
  \( p(f(Y), Y, W) \)

  substitution \([f(Y)/X]\)

- \( p(f(Y), g(Z), f(Y)) \) vs. \\
  \[ \downarrow \]
  \( p(f(Y), Y, W) \)

  substitution \([g(Z)/Y, f(g(Z))/X]\)

- \( p(f(g(Z)), g(Z), f(g(Z))) \) vs. \\
  \[ \downarrow \]
  \( p(f(g(Z)), g(Z), W) \)

  substitution \([f(g(Z))/W, g(Z)/Y, f(g(Z))/X]\)

- \( p(f(g(Z)), g(Z), f(g(Z))) \) vs. \\
  \[ \downarrow \]
  \( p(f(g(Z)), g(Z), f(g(Z))) \)

  substitution \([f(g(Z))/W, g(Z)/Y, f(g(Z))/X]\)
Note on Unification in Prolog

- In Prolog, unification is used in goal matching and in the = (unify) operator.

- However, Prolog’s unification is slightly abridged: it bypasses the “occur check”: $X = f(X)$

  \[X = f(X)\]

  will unify in Prolog, but not in ordinary unification. In effect, $X$ gets bound to the infinite term:

  \[f(f(f(\ldots))))\]
Checking Unifiability with Prolog

As long as there are no occur-check violations, can use = to test unifiability

```prolog
$ swipl
Welcome to SWI-Prolog

?- p(X, g(Z), X) = p(f(Y), Y, W).
X = f(g(Z)),
Y = g(Z),
W = f(g(Z))
```
Occur Check Violation in Prolog

Not considered a violation, but what happens is system-dependent.

?- X = f(Y), Y = g(X).
X = f(g(**)),
Y = g(f(**)).

Here ** means some term not shown.
Try These

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>mgu\n(or not unifiable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(X, f(X), d)$</td>
<td>$p(c, f(c), Y)$</td>
<td></td>
</tr>
<tr>
<td>$p(f(g(X)), g(Z))$</td>
<td>$p(f(Y), Y)$</td>
<td></td>
</tr>
<tr>
<td>$p(f(g(X)), Z)$</td>
<td>$p(g(Y), Y)$</td>
<td></td>
</tr>
<tr>
<td>$p(f(g(X)), X)$</td>
<td>$p(f(g(h(Z))), h(Z))$</td>
<td></td>
</tr>
</tbody>
</table>

Checking Unifiability with Prover9

- In contrast to Prolog, Prover9 does use an occur-check.

\[
\begin{align*}
\text{unification succeeds} & \\
-p(f(y), y). \\
p(x, g(z)). \\
\text{unification fails due to occur-check} & \\
-p(f(y), y). \\
p(z, g(z)).
\end{align*}
\]

Proof:

1. \(p(x, g(y)). \ [\text{assumption}].\)
2. \(-p(f(x), x). \ [\text{assumption}].\)
3. \(\$F. \ [\text{resolve}(1, a, 2, a)].\)
Resolving Predicate Calculus Clauses

- Resolvable clauses must contain literals with the same predicate symbol but of opposite sign (one negated, the other not).

- First “rename apart” the clauses (leave no common variables).

- Pick two such literals, one from each clause.

- Determine whether the literals are unifiable, with mgu $\mu$. If they are, apply $\mu$ to all literals in both clauses. If not, the clauses don’t resolve on these particular literals.

- In the modified clauses, remove all instances of the modified literals used in unification, and form the disjunction of the remaining modified literals.
Complete Predicate Resolution Process

• The process is similar the propositional case, except that we have to
  • rename apart the clauses, then
  • unify literals prior to resolution, and
  • apply the mgu to all literals in the two clauses, before obtaining the resolvent.

• There is also a special issue: “factoring”, that needs to be factored in.
Example of Predicate Resolution

- Clauses:
  - $\neg \text{human}(X) \lor \text{mortal}(X)$
  - $\text{human}(\text{socrates})$
  - $\neg \text{mortal}(\text{socrates})$

  $\neg \text{human}(X) \lor \text{mortal}(X) \quad \neg \text{mortal}(\text{socrates})$

  $\text{mgu} \; \mu = [\text{socrates}/X]$

  $\neg \text{human}(\text{socrates}) \quad \text{human}(\text{socrates})$

  $\text{mgu} \; \mu = []$

  $\bot$
Example Resolving Predicate Clauses

- clause 1: $p(X, g(Z), X) \lor q(X, h(Z))$
- clause 2: $\neg p(f(Y), Y, W) \lor r(f(Y), g(W))$
- These are already renamed apart.

- The atoms $p(X, g(Z), X)$ vs. $p(f(Y), Y, W)$ of each unify with mgu $[f(g(Z))/X, g(Z)/Y, f(g(Z))/W]$

- Apply the mgu to both clauses:
  - clause 1’: $p(f(g(Z)), g(Z), f(g(Z))) \lor q(f(g(Z)), h(Z))$
  - clause 2’: $\neg p(f(g(Z)), g(Z), f(g(Z))) \lor r(f(g(Z)), g(f(g(Z))))$

- Remove the instances of the unified atoms and form the disjunction.
- Resolvent: $q(f(g(Z)), h(Z)) \lor r(f(g(Z)), g(f(g(Z))))$
Example of Predicate Resolution

- **Clauses:**
  - \( \neg p(X) \lor q(f(X), X) \)
  - \( p(g(b)) \)
  - \( \neg q(Y, Z) \)

\[ \neg p(X) \lor q(f(X), X) \quad \neg q(Y, Z) \]

\( \text{mgu: } [f(Z)/Y, Z/X] \)

\( \neg p(Z) \)

\( \text{mgu: } [g(b)/Z] \)

\( p(g(b)) \)

\( \bot \)
Unit Preference Strategy

As with propositional resolution, resolving with unit clauses first is a good heuristic.
Check This Set for Unsatisfiability
(use Unit Preference)

1. \( \neg e(x) \lor q(x) \lor s(x, f(x)) \)
2. \( \neg e(x) \lor q(x) \lor r(f(x)) \)
3. \( p(a) \)
4. \( e(a) \)
5. \( \neg s(a, x) \lor p(x) \)
6. \( \neg p(x) \lor \neg q(x) \)
7. \( \neg p(x) \lor \neg r(x) \)
Clausal Form for Sequents

- Often, we’ll want to prove a sequent of the form
  - $\forall x \forall y (\ldots)$
  - $\forall x \forall y (\ldots)$
  - $\vdash \LDots$
- For premises of the form $\forall x \forall y (\ldots)$ where $\ldots$ has no quantifiers, we can just drop the quantifiers.
- We need to negate the conclusion ____.
Mushroom Example

1. Every fungus is a mushroom or a toadstool.
2. Every boletus is a fungus.
3. All toadstools are poisonous.
4. No boletus is a mushroom.
5. Specimen b is a boletus.
6. Therefore: Specimen b is poisonous.
Mushroom Example

1. $\forall x \ (\text{fungus}(x) \rightarrow (\text{mushroom}(x) \lor \text{toadstool}(x)))$

2. $\forall x \ (\text{boletus}(x) \rightarrow \text{fungus}(x))$

3. $\forall x \ (\text{toadstool}(x) \rightarrow \text{poisonous}(x))$

4. $\forall x \ (\text{boletus}(x) \rightarrow \neg \text{mushroom}(x))$

5. boletus(b)

6. Therefore: poisonous(b)
Prove Unsatisfiability of Mushroom Clauses

1. \( \neg \text{fungus}(X) \lor \text{mushroom}(X) \lor \text{toadstool}(X) \)

2. \( \neg \text{boletus}(X) \lor \text{fungus}(X) \)

3. \( \neg \text{toadstool}(X) \lor \text{poisonous}(X) \)

4. \( \neg \text{boletus}(X) \lor \neg \text{mushroom}(X) \)

5. \( \text{boletus}(b) \)

6. \( \neg \text{poisonous}(b) \) (negated conclusion)
Mushroom Clauses in Prover9

- fungus(x) | mushroom(x) | toadstool(x).

- boletus(x) | fungus(x).

- toadstool(x) | poisonous(x).

- boletus(x) | -mushroom(x).

  boletus(b).

Goal:

poisonous(b).
Prover9 Output for Mushrooms

%%%%%%%%%%%%%%%%%%%% PROOF %%%%%%%%%%%%%%%%%%%%

% ------- Comments from original proof -------
% Proof 1 at 0.00 (+ 0.00) seconds.
% Length of proof is 13.
% Level of proof is 5.
% Maximum clause weight is 0.
% Given clauses 0.

1 poisonous(b) # label(non_clause) # label(goal). [goal].
2 -boletus(x) | fungus(x). [assumption].
3 -fungus(x) | mushroom(x) | toadstool(x). [assumption].
4 -boletus(x) | mushroom(x) | toadstool(x). [resolve(2,b,3,a)].
5 -toadstool(x) | poisonous(x). [assumption].
6 boletus(b). [assumption].
7 -boletus(x) | -mushroom(x). [assumption].
8 -boletus(x) | mushroom(x) | poisonous(x). [resolve(4,c,5,a)].
9 mushroom(b) | poisonous(b). [resolve(8,a,6,a)].
10 -poisonous(b). [deny(1)].
11 mushroom(b). [resolve(9,b,10,a)].
12 -mushroom(b). [resolve(6,a,7,a)].
13 $F. [resolve(11,a,12,a)].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% end of proof %%%%%%%%%%%%%%%%%%%%%
Sequents with $\exists$

- Often, we’ll want to prove a sequent of the form
  - $\forall x \forall y (...)$,  
  - $\forall x \forall y (...)  
    \mid \vdash \forall x \forall y (...)  

- For premises of the form $\forall x \forall y (...)$ where $\ldots$ has no quantifiers, we can just drop the quantifiers.

- We need to **negate** the conclusion, so that will become $\neg \forall x \forall y (...)$ which is equivalent to

$$\exists x \exists y \neg (...) .$$

We *cannot* simply drop the $\exists$ quantifiers in this case!!
Example Sequent with $\exists$

- Consider the sequent

$$\forall y \ p(y) \vdash \forall y \ p(x)$$

- The premise translates to a clause

$$p(y)$$

- The **conclusion** is negated to become $\exists x \ \neg p(x)$.

- How do we handle this?
Skolem Constants/Functions to the Rescue!

• To get rid of the quantifier in

\[ \exists x \neg p(x) \]

we use a trick:

Create a **new constant**, say b (called a **Skolem constant**) and replace x with that:

\[ \neg p(b) \]

• Some thought will show that:

There is an interpretation that satisfies \( \neg p(b) \) **iff** there is one that satisfies the original formula \( \exists x \neg p(x) \).

• We get to pick the value for b in finding a satisfying interpretation, just as we get to pick the value for x in \( \exists x \).
Resolution with Skolem Constant

- Consider the sequent
  \[ \forall y \ p(y) \mid \neg \forall x \ p(x) \]
- The premise translates to a clause
  \[ p(y) \]
- The negated conclusion \( \neg \forall x \ p(x) \) translates to \( \exists x \neg p(x) \).
- This gives a clause, where \( b \) is a Skolem function:
  \[ \neg p(b) \]
- We are good to go!
- Resolution produces \( \bot \) in 1 step.
Another Example

• Consider the sequent

\[ \exists x \forall y \ p(x, y) \vdash \forall y \exists x \ p(x, y) \]

• Premise clause, with Skolem constant b:

\[ p(b, y) \]

• Negated conclusion: \( \forall y \exists x \ p(x, y) \) giving clause with Skolem constant c:

\[ \neg p(x, c) \]

Resolution produces \( \bot \) in 1 step.
Yet Another Example

• Consider the sequent

\[(\forall x \ a(x)) \rightarrow \exists x \ b(x) \mid \neg \exists x \ (a(x) \rightarrow b(x))\]

• Premise clause:

\[\neg a(c) \lor b(d)\]

• Conclusion clauses (\(\neg \exists x\) becomes \(\forall x \neg\)):

\[a(x)\]
\[\neg b(x)\]

Resolution produces \(\bot\) in 2 steps.
Prover9 Proof of Previous Example

c1 and c2 are Skolem constants

1 (all x a(x)) -> (exists x b(x)) # label(non_clause).  [assumption].
2 (exists x (a(x) -> b(x))) # label(non_clause) # label(goal).  [goal].
3 a(x).  [deny(2)].
4 -a(c1) | b(c2).  [clausify(1)].
5 b(c2).  [resolve(3,a,4,a)].
6 -b(x).  [deny(2)].
7 $F$.  [resolve(5,a,6,a)].
Skolem Functions: the General Case

• \( \forall x \, \forall y \ldots \exists z \ldots \) (\( \exists \) preceded by \( \forall \))

• \( z \) is replaced with \( f(x, y, \ldots) \)

• \( f \) is a new function symbol, the arguments of which are the \( \forall \) quantified variables on the left.

• The rationale here is that “the \( z \)” that exists depends on \( x, y, \ldots \).

• Again, there is an interpretation satisfying the original formula iff there is an interpretation satisfying the revised formula.
Example: Skolem with Arguments

Prove: “The composition of two ‘onto’ functions is ‘onto’.”
Example: Skolem with Arguments

Prove: “The composition of two ‘onto’ functions is ‘onto’.”

• (Because we don’t have = yet) **represent the two functions as binary predicates.**
  F(x, y) means y is the image of x.

• “H is the composition of F and G”:
  \[
  \forall x \forall y \forall z ((F(x, y) \land G(y, z)) \rightarrow H(x, z))
  \land \forall x \forall z (H(x, z) \rightarrow \exists y (F(x, y) \land G(y, z)))
  \]

• “F is onto”: \( \forall y \exists x F(x, y) \)
• “G is onto”: \( \forall z \exists y G(y, z) \)
• “H is onto”: \( \forall z \exists x H(x, z) \)
Translation to Clausal Form

- $\forall y \exists x F(x, y)$ becomes $F(f(y), y)$  
  [f is a Skolem function]

- $\forall z \exists y G(y, z)$ becomes $G(g(z), z)$  
  [g is a Skolem function]

- $\forall x \forall y \forall z ((F(x, y) \land G(y, z)) \rightarrow H(x, z))$ becomes
  
  $\neg F(x, y) \lor \neg G(y, z) \lor H(x, z)$

- $\forall x \forall z (H(x, z) \rightarrow \exists y (F(x, y) \land G(y, z)))$ becomes
  
  $\neg H(x, z) \lor F(x, h(x, z))$  
  [h is a Skolem function]

  $\neg H(x, z) \lor G(h(x, z), z)$

- $\neg \forall z \exists x H(x, z)$ becomes $\exists z \forall x \neg H(x, z)$,  
  which, as a clause, is:

  $\neg H(x, a)$  
  [a is a Skolem constant]
Resolution Proof

1. $F(f(x), y)$
2. $G(g(z), z)$
3. $\neg F(x, y) \lor \neg G(y, z) \lor H(x, z)$
4. $\neg H(x, z) \lor F(x, h(x, z))$
5. $\neg H(x, z) \lor G(h(x, z), z)$
6. $\neg H(x, a)$

7. $\neg F(x, y) \lor \neg G(y, a)$ from 3, 6
8. $\neg G(y, a)$ from 1, 7
9. $\bot$ from 2, 8

(4 and 5 were not needed in the proof.)
Prover9 Version

Proof:
1. $-F(x, y) \mid -G(y, z) \mid H(x, z)$. [assumption].
2. $F(f(x), y)$. [assumption].
3. $-G(x, y) \mid H(f(z), y)$. [resolve(1,a,2,a)].
4. $G(g(x), x)$. [assumption].
5. $H(f(x), y)$. [resolve(4,a,5,a)].
6. $-H(x, a)$. [assumption].
7. $F$. [resolve(8,a,9,a)].

Clauses:
- $F(f(x), y)$
- $G(g(z), z)$
- $-F(x, y) \mid -G(y, z) \mid H(x, z)$.
- $-H(x, z) \mid F(x, h(x, z))$.  
- $-H(x, z) \mid G(h(x, z), z)$.
- $-H(x, a)$. 

How to get a clause form in general?

• First, using rules to be described, convert the formula into “prenex form” (all quantifiers are outside on the left), e.g.

\[
\forall x \exists y \ (F(x) \to (G(x, y) \to H(y)))
\]

prefix matrix

• The parts of this form are called the “prefix” and the “matrix”.

• Skolemize \( \exists \) quantified variables.

• Drop \( \forall \) quantifiers.

• Convert the resulting matrix to CNF.
Conversion to Prenex Form

- Replace all connectives other than $\wedge \vee \neg$ with their $\wedge \vee \neg$ counterparts.

- **Push** negations inward.

- **Pull** quantifiers to the outside using rules to be defined.
Example of Prenex Conversion

- $\forall x \forall y((\exists z (p(x, z) \land p(y, z))) \rightarrow \exists u q(x, y, u))$ replace $\rightarrow$
- $\forall x \forall y(\neg (\exists z (p(x, z) \land p(y, z))) \lor \exists u q(x, y, u))$ push $\neg$ in
- $\forall x \forall y((\forall z (\neg (p(x, z) \land p(y, z))) \lor \exists u q(x, y, u))$ push $\neg$ in
- $\forall x \forall y (\forall z (\neg p(x, z) \lor \neg p(y, z))) \lor \exists u q(x, y, u))$ pull $\exists u$ out
- $\forall x \forall y \exists u (\forall z (\neg p(x, z) \lor \neg p(y, z))) \lor q(x, y, u))$ pull $\forall z$ out
- $\forall x \forall y \exists u \forall z (\neg p(x, z) \lor \neg p(y, z) \lor q(x, y, u))$ prefix
- matrix (already CNF in this case)
Completion of Conversion to CNF

- Prenex Form:

\[ \forall x \forall y \exists u \forall z (\neg p(x, z) \lor \neg p(y, z) \lor q(x, y, u)) \]

- Skolemize \( u \) as \( f(x, y) \) then drop \( \forall x \forall y \forall z \):

\[ \neg p(x, z) \lor \neg p(y, z) \lor q(x, y, f(x, y)) \]
Prenex Form Conversion

- It is tacitly assumed that we have made the non-empty universe assumption.
Basic Prenex Quantifier Rules
(for pulling quantifiers to the outside)

- Here $\Rightarrow$ means “replace with”
  1. $(\forall x \ F) \land G \Rightarrow \forall x (F \land G)$, provided $x$ is not free in $G$
  2. $(\forall x \ F) \lor G \Rightarrow \forall x (F \lor G)$, provided $x$ is not free in $G$
  3. $(\exists x \ F) \land G \Rightarrow \exists x (F \land G)$, provided $x$ is not free in $G$
  4. $(\exists x \ F) \lor G \Rightarrow \exists x (F \lor G)$, provided $x$ is not free in $G$

- plus the symmetric counterparts of these rules with $G$ part quantified instead of the $F$ part.

- **Renaming some variables** may be need to enable the rule to be applied.
Example of Basic Prenex Quantifier Rules

\[(\exists x \, F[x]) \land \forall x \, G[x] \Rightarrow (by \text{ renaming second } x)\]

\[(\exists x \, F[x]) \land \forall y \, G[y] \Rightarrow (by \text{ rule 3, as } x \text{ is not free in } G)\]

\[\exists x \, (F[x] \land \forall y \, G[y]) \Rightarrow (by \text{ rule 1 symmetric counterpart})\]

\[\exists x \, \forall y \, (F[x] \land G[y])\]
Justifying Prenex Quantifier Rules Using Natural Deduction: $\forall \land$ Rule a.

Proviso is introduced by prefixing ‘WHERE x NOTIN G IS’ in Jape.
To establish **equivalence**, rather than just implication, we need the **converse** of each rule: $\forall \land$ Rule b.

Need the **non-empty universe** assumption in this direction (not implicit in Jape).

Otherwise, there is no way to get $G$ by itself.
Justification of Rules
Using Natural Deduction: $\forall \lor$ Rule a.
Justification of Rules
Using Natural Deduction: $\forall \lor$ Rule b.

\[
\begin{align*}
1: & \quad \forall x. (F(x) \lor G) \text{ premise} \\
2: & \quad G \lor \neg G \quad \text{Theorem } \lor \neg \lor E \\
3: & \quad G \quad \text{assumption} \\
4: & \quad \forall x. F(x) \lor G \quad \lor \text{ intro 3} \\
5: & \quad \neg G \quad \text{assumption} \\
6: & \quad \text{actual i} \quad \text{assumption} \\
7: & \quad F(i) \lor G \quad \forall \text{ elim 1,6} \\
8: & \quad F(i) \quad \text{assumption} \\
9: & \quad G \quad \text{assumption} \\
10: & \quad \bot \quad \neg \text{ elim 9,5} \\
11: & \quad F(i) \quad \text{contra (constructive) 10} \\
12: & \quad F(i) \quad \lor \text{ elim 7,8-8,9-11} \\
13: & \quad \forall x. F(x) \quad \forall \text{ intro 6-12} \\
14: & \quad \forall x. F(x) \lor G \quad \lor \text{ intro 13} \\
15: & \quad \forall x. F(x) \lor G \quad \lor \text{ elim 2,3-4,5-14} \\
\end{align*}
\]
Justification of Rules
Using Natural Deduction: $\exists \forall \ a$. 

Non-empty universe assumption, needed in 7-11
Justification of Rules
Using Natural Deduction: $\exists \forall b$. 
Justification of Rules
Using Natural Deduction: $\exists \land a$. 
Justification of Rules
Using Natural Deduction: $\exists \land b$. 

\begin{align*}
1: & \exists x. (F(x) \land G) & \text{premise} \\
2: & \text{actual i1, } F(i1) \land G & \text{assumptions} \\
3: & F(i1) & \land \text{elim 2.2} \\
4: & \exists x. F(x) & \exists \text{ intro 3.2.1} \\
5: & \exists x. F(x) & \exists \text{ elim 1,2–4} \\
6: & \text{actual i, } F(i) \land G & \text{assumptions} \\
7: & G & \land \text{elim 6.2} \\
8: & G & \exists \text{ elim 1,6–7} \\
9: & \exists x. F(x) \land G & \land \text{ intro 5,8} \\
\end{align*}
Example of Conversion by Prover9

Input for the “onto” example:
all y exists x F(x, y).
all z exists y G(y, z).
all x all y all z (F(x, y) & G(y, z) -> H(x, z)).
all x all z (H(x, z) -> exists y (F(x, y) & G(y, z))).
goal: all z exists x H(x, z).

Proof:
1 (all x exists y F(y,x)) # label(non_clause). [assumption].
2 (all x exists y G(y,x)) # label(non_clause). [assumption].
3 (all x all y all z (F(x,y) & G(y,z) -> H(x,z))) # label(non_clause). [assumption].
5 (all x exists y H(y,x)) # label(non_clause) [goal].
6 -F(x,y) | -G(y,z) | H(x,z). [clausify(3)].
7 F(f1(x),x). [clausify(1)].
9 -G(x,y) | H(f1(x),y). [resolve(6,a,7,a)].
10 G(f2(x),x). [clausify(2)].
13 -H(x,c1). [deny(5)].
14 H(f1(f2(x)),x). [resolve(9,a,10,a)].
15 $F. [resolve(14,a,13,a)].
Example: Group Theory Clauses

- $f$ is the group operation, $i$ is the inverse operation, $e$ is the equality predicate

- $\forall x \forall y \forall x \ e(f(x, f(y, z)), f(f(x, y), z))$ becomes $e(f(x, f(y, z)), f(f(x, y), z))$

- $\forall x \ e(f(x, u), x)$ becomes $e(f(x, u), x)$

- $\forall x \ e(f(x, i(x)), u)$ becomes $e(f(x, i(x)), u)$
Example: Equality Theory Clauses

- We need to axiomatize equality predicate e, e.g.

- $\forall x \ e(x, x)$
  becomes
  $e(x, x)$

- $\forall x \ \forall y \ \forall u \ \forall v \ ((e(x, y) \land e(v, w)) \rightarrow e(f(x, v), f(y, w)))$
  becomes
  $\neg e(x, y) \lor \neg e(v, w) \lor e(f(x, v), f(y, w))$

- $\forall x \ \forall u \ (e(x, u) \rightarrow e(i(x), i(u)))$
  becomes
  $\neg e(x, u) \lor e(i(x), i(u))$

- $\forall x \ \forall y \ (e(x, y) \rightarrow e(y, x))$
  becomes
  $\neg e(x, y) \lor e(y, x)$

Exercise: Convert the transitive property of e.
Example of Group Theory Clauses with Negated Conclusion

1. $e(f(x, f(y, z)), f(f(x, y), z))$
2. $e(f(x, u), x)$
3. $e(f(x, i(x)), u)$
4. $e(x, x)$
5. $\neg e(x, y) \lor \neg e(v, w) \lor e(f(x, v), f(y, w))$
6. $\neg e(x, y) \lor e(y, x)$
7. $\neg e(x, y) \lor \neg e(y, z) \lor e(x, z)$
8. $\neg e(i(i(b)), b)$

This is to show that $\forall x \ e(i(i(x)), x)$:

“In a group, the inverse of the inverse of an element is the element itself.”
Equality: Paramodulation

- Prover9 has a built-in equality, so axiomatizing equality as a predicate is not generally necessary.

- The “paramodulation” rule, which essentially captures the $=$ rules of Natural Deduction.

\[
\frac{\alpha \lor (s = t) \quad \beta \lor \gamma[r]}{(\alpha \lor \beta \lor \gamma[t])\theta}
\]

$\gamma[r]$ is a literal containing term $r$

$\theta = \text{unify}(s,r)$
Prover9 Group theory input using builtin equality

\[ f(x, f(y, z)) = f(f(x, y), z). \]
\[ f(x, c) = x. \]
\[ f(x, i(x)) = c. \]
\[ i(i(b)) \neq b. \]

(Show unsatisfiable)
Prover9 proof using builtin equality

1  f(x,f(y,z)) = f(f(x,y),z).  [assumption].
2  f(f(x,y),z) = f(x,f(y,z)).  [copy(1),flip(a)].
3  f(x,c) = x.  [assumption].
4  f(x,i(x)) = c.  [assumption].
5  i(i(b)) != b.  [assumption].
6  f(x,f(c,y)) = f(x,y).  [para(3(a,1),2(a,1,1)),flip(a)].
7  f(x,f(i(x),y)) = f(c,y).  [para(4(a,1),2(a,1,1)),flip(a)].
12 f(c,i(i(x))) = x.  [para(4(a,1),7(a,1,2)),rewrite([3(2)]),flip(a)].
14 f(x,i(i(y))) = f(x,y).  [para(12(a,1),2(a,2,2)),rewrite([3(2)])].
15 f(c,x) = x.  [para(12(a,1),6(a,2)),rewrite([14(5),6(4)])].
18 f(x,f(i(x),y)) = y.  [back_rewrite(7),rewrite([15(5)])].
21 i(i(x)) = x.  [para(4(a,1),18(a,1,2)),rewrite([3(2)]),flip(a)].
22 $F$.  [resolve(21,a,5,a)].
Example: Non-Clausal Input in Prover9: Automatic Translation to Clausal Form

all x exists y r(x, y).

all x all y all z ((r(x,y)&r(y,z)) -> r(x,z)).

all x all y (r(x,y) -> r(y,x)).

-(all x r(x,x)).
Example: Non-Clausal Input in Prover9: Automatic Translation to Clausal Form

becomes (c1, f1 are Skolem constant and function):

1 (all x exists y r(x,y)) # label(non_clause).  [assumption].
2 (all x all y all z (r(x,y) & r(y,z) -> r(x,z))) # label(non_clause).  [assumption].
3 (all x all y (r(x,y) -> r(y,x))) # label(non_clause).  [assumption].
4 -(all x r(x,x)) # label(non_clause).  [assumption].

5 r(x,f1(x)).  [clausify(1)].
6 -r(x,y) | -r(y,z) | r(x,z).  [clausify(2)].
7 -r(x,y) | r(y,x).  [clausify(3)].
8 -r(c1,c1).  [clausify(4)].

10 r(f1(x),x).  [ur(7,a,5,a)].
12 -r(f1(c1),c1).  [ur(6,a,5,a,c,8,a)].
13 $F.  [resolve(12,a,10,a)].
Answer Extraction

- Resolution is not just for proving theorems anymore.

- Resolution can be used for extracting answers from a database, knowledge base, or reasoning system.
From Yes-No Answer to Terms

• Consider the clause set:
  • \neg \text{human}(x) \lor \text{mortal}(x)
  • \text{human}(\text{socrates})
  • \neg \text{mortal}(\text{socrates})

• Obviously this set is unsatisfiable, and a proof can be obtained by resolution.

• What if we drop the third clause. The first two clauses are satisfiable, and can be considered a “knowledge base”.

• We can ask a question of this knowledge base, such as:

  Name someone who is mortal.
Answer Literals

• An answer literal is a special literal that captures the answer to a question.

• We convert the negation of a specific conclusion into a clause involving an answer literal:

\[ \neg \text{mortal}(x) \lor \text{answer}(x) \].

• Resolution stops when a clause with only the answer is present.
Resolution with Answer Literals

- **Clauses:**
  1. $\neg \text{human}(x) \lor \text{mortal}(x)$
  2. $\text{human}(\text{socrates})$
  3. $\neg \text{mortal}(x) \lor \text{answer}(x)$

- **Resolution steps:**
  4. $\text{mortal}(\text{socrates})$ from 1, 2
  5. $\text{answer}(\text{socrates})$ from 3, 4
Answering Questions in Prover9

- To find terms such that $p(x)$, incant:

  $$-p(x) \# \text{answer}(x).$$
Who is Mortal, in Prover9

Clausal Form

- human(x) | mortal(x).

human(socrates).

-mortal(x) # answer(x).
Clauses

- \(-\text{human}(x) \lor \text{mortal}(x)\).
- \(\text{human}(\text{socrates})\).
- \(-\text{mortal}(x) \# \text{answer}(x)\).

\begin{align*}
\text{Proof} \\
1 & \text{human}(\text{socrates}). \ \text{[assumption]}.
2 & -\text{human}(x) \lor \text{mortal}(x). \ \text{[assumption]}.
3 & \text{mortal}(\text{socrates}). \ \text{[resolve(1,a,2,a)]}.
4 & -\text{mortal}(x) \# \text{answer}(x). \ \text{[assumption]}.
5 & $F \# \text{answer}(\text{socrates}). \ \text{[resolve(3,a,4,a)]}.
\end{align*}
Who is Caroline’s Grandfather?

**Clauses**

- father(x, y) | parent(x, y).
- father(x, y) | -parent(y, z) | grandfather(x, z).

father(joe, john).
father(john, caroline).

-grandfather(x, caroline) # answer(x).

**Proof**

1 father(joe, john).  [assumption].
2 -father(x,y) | parent(x,y).  [assumption].
3 -father(x,y) | -parent(y,z) | grandfather(x,z).  [assumption].
4 father(john, caroline).  [assumption].
5 -parent(john, x) | grandfather(joe, x).  [resolve(1,a,3,a)].
6 -grandfather(x, caroline) # answer(x).  [assumption].
8 -parent(john, caroline) # answer(joe).  [resolve(5,b,6,a)].
10 parent(john, caroline).  [resolve(4,a,2,a)].
12 $F # answer(joe).  [resolve(8,a,10,a)].
Logic Puzzles solvable by Resolution

% Professors Dodds, Lewis, and Stone each frequent different establishments (one of Alice's, Harry's, or Joe's) for liquid refreshment.

% Each prof prefers a different beer (one of Anchor, Bud, and Miller)
% Each establishment serves a unique beer.

% Professor Stone prefers Bud.
% Professor Lewis doesn't prefer Miller.
% The prof who prefers Miller frequents Alice's bar.
% The prof who prefers Anchor does not frequent Joe's.

% Which bar does each prof frequent and what beer does each prefer?
% Clauses corresponding to the clues:
prefer(Stone, Bud). % Clue 1
-prefer(Lewis, Miller). % Clue 2
-prefer(x, Miller) | frequent(x, Alice). % Clue 3
-prefer(x, Anchor) | -frequent(x, Joe). % Clue 4

% Individuals
prof(Dodds). prof(Stone). prof(Lewis).
beer(Anchor). beer(Bud). beer(Miller).
bar(Alice). bar(Harry). bar(Joe).

% Although constants do not unify, they could conceivably be equal.
Dodds != Stone. Dodds != Lewis. Stone != Lewis.

% Any bar that a professor frequents serves the beers that he or she prefers.
-frequent(x, y) | serves(y, z) | prefer(x, z).

% Every bar is frequented by some prof.
-bar(y) | frequent(Dodds, y) | frequent(Stone, y) | frequent(Lewis, y).

% Every beer is preferred by some prof.
-beer(y) | prefer(Dodds, y) | prefer(Stone, y) | prefer(Lewis, y).

% Each bar serves a unique beer.
-serves(x, y) | -serves(x, z) | y = z.

% Each prof prefers a unique beer.
-prefer(x, y) | -prefer(x, z) | y = z.

% Each prof frequents a unique bar.
-frequent(x, y) | -frequent(x, z) | y = z.

% Which bars are frequented, and which beers preferred, by which professors?
-frequent(Dodds, x) | -frequent(Stone, y) | -frequent(Lewis, z)
| -prefer(Dodds, u) | -prefer(Stone, v) | -prefer(Lewis, w)
#answer([Dodds, x, u, Stone, y, v, Lewis, z, w]).
% Proof 1 at 0.01 (+ 0.01) seconds: [Dodds,Alice,Miller,Stone,Joe,Bud,Lewis,Harry,Anchor].
% Length of proof is 46.
% Level of proof is 11.
% Maximum clause weight is 18.
% Given clauses 85.
1 -beer(x) | prefer(Dodds,x) | prefer(Stone,x) | prefer(Lewis,x).  [assumption].
2 beer(Anchor).  [assumption].
4 beer(Miller).  [assumption].
5 -bar(x) | frequent(Dodds,x) | frequent(Stone,x) | frequent(Lewis,x).  [assumption].
7 bar(Harry).  [assumption].
8 bar(Joe).  [assumption].
9 prefer(Stone,Bud).  [assumption].
10 -prefer(Lewis,Miller).  [assumption].
11 -prefer(x,Miller) | frequent(x,Alice).  [assumption].
12 -prefer(x,Anchor) | frequent(x,Joe).  [assumption].
54 Anchor != Bud.  [assumption].
55 Alice != Miller.  [assumption].
61 Bud != Miller.  [assumption].
62 Miller != Bud.  [copy(61),flip(a)].
73 Alice != Harry.  [assumption].
74 Harry != Alice.  [copy(73),flip(a)].
75 Alice != Joe.  [assumption].
76 Joe != Alice.  [copy(75),flip(a)].
77 Harry != Joe.  [assumption].
78 Joe != Harry.  [copy(77),flip(a)].
81 -prefer(x,y) | -prefer(x,z) | y = z.  [assumption].
82 -prefer(x,y) | -prefer(x,z) | y = z.  [assumption].
83 -frequent(Dodds,x) | -frequent(Stone,y) | -frequent(Lewis,z) | -prefer(Dodds,u) | -prefer(Stone,w) | -prefer(Lewis,v5) # answer([[Dodds,x,u,Stone,y,w,Lewis,z,v5]].  [assumption]
84 prefer(Dodds,Anchor) | prefer(Stone,Anchor) | prefer(Lewis,Anchor).  [prefer(Anchor,Anchor)].
85 prefer(Dodds,Miller) | prefer(Stone,Miller) | prefer(Lewis,Miller).  [resolve(1,0,4,a)].
86 prefer(Dodds,Miller) | prefer(Stone,Miller).  [copy(85),unit_del(c,10)].
88 frequent(Dodds,Harry) | frequent(Stone,Harry) | frequent(Lewis,Harry).  [resolve(5,a,7,o)].
89 frequent(Dodds,Joe) | frequent(Stone,Joe) | frequent(Lewis,Joe).  [resolve(5,a,8,o)].
93 -prefer(Stone,Miller).  [ur(83,b,9,a,c,62,a)].
94 -prefer(Stone,Anchor).  [ur(83,b,9,a,c,54,a)].
99 prefer(Dodds,Miller).  [back_unit_del(80),unit_del(b,93)].
100 prefer(Dodds,Anchor).  [back_unit_del(80),unit_del(b,93)].
110 frequent(Dodds,Joe) | frequent(Stone,Joe) | -prefer(Lewis,Alice).  [resolve(89,c,12,b)].
112 frequent(Dodds,Anchor).  [resolve(99,a,11,o)].
115 -prefer(Dodds,Anchor).  [ur(81,b,9,a,c,55,a)].
121 prefer(Lewis,Anchor).  [back_unit_del(118),unit_del(c,121)].
122 frequent(Dodds,Joe) | frequent(Stone,Joe).  [back_unit_del(118),unit_del(c,121)].
123 -frequent(Stone,x) | -frequent(Lewis,y) | -frequent(Dodds,z) | -prefer(Dodds,u) | -prefer(Stone,w) | -prefer(Lewis,w) # answer([[Dodds,Alice,z,Stone,x,u,Lewis,y,w]].  [resolve(112,a,83,o)].
126 -frequent(Dodds,Joe).  [ur(82,b,112,a,c,76,a)].
127 -frequent(Dodds,Harry).  [ur(82,b,112,a,c,74,a)].
134 frequent(Stone,Joe).  [back_unit_del(132),unit_del(a,126)].
138 frequent(Stone,Harry) | frequent(Lewis,Harry).  [back_unit_del(88),unit_del(a,127)].
150 -frequent(Stone,Harry).  [ur(82,b,134,a,c,78,o(flip))].
158 frequent(Lewis,Harry).  [back_unit_del(138),unit_del(a,150)].
197 -frequent(Stone,x) # answer([[Dodds,Alice,Miller,Stone,x,Bud,Lewis,Harry,Anchor]].  [ur(123,b,758,a,c,99,a,d,9,a,e,121,o)].
198 $P # answer([[Dodds,Alice,Miller,Stone,x,Bud,Lewis,Harry,Anchor]].  [resolve(197,a,134,a)].
What if Solution not Unique?

- I’m not quite sure how Prover9 returns multiple solutions, if it indeed can.
- Its predecessor, Otter, could handle it by showing the alternatives as a disjunction of answer literals.
State and Motion Puzzles and Games

- Moves in a motion puzzle or game can often be encoded as logic.

- Resolution can be used to find a solving or winning sequence of moves.
Example: Linear Peg Solitaire
Linear Peg Solitaire Objective

- Pegs of two colors are shown in their home positions at the top.
- The objective is to completely reverse the pegs, so that each peg’s original home is occupied by a peg of the opposite color.
- Allowable actions:
  - Move: A peg can be moved toward the opposite side by moving into an adjacent empty hole.
  - Jump: A peg can jump toward the opposite side over a peg of the opposite color, provided that there is a hole to receive the jumping peg.
  - Versions of the puzzle exists for 2n pegs (n of each color) and 2n+1 holes.
  - Ideally, each version can be solved.
Formulation (This will be important when we talk about **computability** later on.)

- Represent the **state** of the game with two terms.
- Say the pegs are **w** for white, **r** for red.
- Represents the pegs **away from the hole** in either direction as a composition of function symbols.
- The initial state shown is:
  \[ s(w(w(w(w(c))))), r(r(r(r(c)))) \]
- The second state shown is:
  \[ s(r(w(r(w(w(c)))))), w(r(r(c))) \]
- **c** is a dummy constant symbol
Formulating Moves

- **Simple moves (non-jump):**
  - move(s(w(X), Y), s(X, w(Y)))
  - move(s(X, r(Y)), s(r(X), Y))

- **Jump moves:**
  - move(s(r(w(X))), Y), s(X, r(w(Y)))
  - move(s(X), w(r(Y))), s(w(r(X)), Y)
Formulating Reachability

- Initial state:
  \[
  \text{reachable}(s(w(w(w(w(c))))), r(r(r(r(c))))))
  \]

- State change:
  \[
  \neg \text{reachable}(X) \lor \neg \text{move}(X, Y) \lor \text{reachable}(Y)
  \]

- Final state:
  \[
  \neg \text{reachable}(s(r(r(r(r(c))))), w(w(w(w(c))))))
  \]
move(s(w(x), y), s(x, w(y))).
move(s(x, r(y)), s(r(x), y)).

move(s(r(w(x)), y), s(x, r(w(y)))).
move(s(x, w(r(y))), s(w(r(x)), y)).

reachable(s(w(w(w(w(c)))), r(r(r(r(c)))))).

-reachable(x) | -move(x, y) | reachable(y).

-reachable(s(r(r(r(r(c)))), w(w(w(w(c)))))).
Proof for 2 pegs of each color

1 -reachable(x) | -move(x,y) | reachable(y). [assumption].
2 move(s(r(x),y),s(x,r(y))). [assumption].
3 move(s(x,b(y)),s(b(x),y)). [assumption].
5 move(s(b(r(x)),y),s(x,b(r(y))))). [assumption].
6 move(s(x,r(b(y))),s(r(b(x)),y)). [assumption].
8 reachable(s(r(r(c)),b(b(c)))). [assumption].
9 -reachable(s(b(b(c)),r(r(c)))). [assumption].
10 -reachable(s(r(x),y)) | reachable(s(x,r(y)))). [resolve(1,b,2,a)].
11 -reachable(s(x,b(y)) | reachable(s(b(x),y)). [resolve(1,b,3,a)].
13 -reachable(s(b(r(x)),y)) | reachable(s(x,b(r(y))))). [resolve(1,b,5,a)].
14 -reachable(s(x,r(b(y)))) | reachable(s(r(b(x)),y)). [resolve(1,b,6,a)].
16 -reachable(s(r(b(b(c))),r(c))). [resolve(10,b,9,a)].
17 reachable(s(r(c),r(b(b(c)))). [ur(10,a,8,a)].
22 -reachable(s(b(c),r(b(r(c))))). [resolve(16,a,14,b)].
25 -reachable(s(c,b(r(b(c))))). [resolve(22,a,11,b)].
28 reachable(s(r(b(r(c))),b(c))). [ur(14,a,17,a)].
31 reachable(s(b(r(b(r(c)))),b(c)). [ur(11,a,28,a)].
33 -reachable(s(b(r(c)),b(r(c))))). [resolve(25,a,13,b)].
37 -reachable(s(b(r(b(r(c)))),c)). [resolve(33,a,13,b)].
38 $F$. [resolve(37,a,31,a)].
Proof for 3 Pegs of Each Color
(output of Otter rather than Prover9, more traceable)

7 [ ]  reachable(s(w(w(w(c)))),r(r(r(c)))))
8 [hyper,7,1,4]  reachable(s(r(w(w(w(c)))),r(r(c))))
12 [hyper,5,1,8]  reachable(s(w(w(c))),r(w(r(r(c)))))
16 [hyper,12,1,3]  reachable(s(w(c)),w(r(w(r(r(c)))))
20 [hyper,16,1,6]  reachable(s(w(r(w(c)))),w(r(r(c))))
27 [hyper,20,1,6]  reachable(s(w(r(w(r(w(c))))),r(c))
34 [hyper,27,1,4]  reachable(s(r(w(r(w(r(w(c))))),c))
39 [hyper,34,1,5]  reachable(s(r(w(r(w(c))))),r(w(c)))
44 [hyper,39,1,5]  reachable(s(r(w(c))),r(w(r(w(c))))
51 [hyper,44,1,5]  reachable(s(c,r(w(r(w(r(w(c)))))))
57 [hyper,51,1,4]  reachable(s(r(c)),w(r(w(r(w(c))))))
63 [hyper,57,1,6]  reachable(s(w(r(r(c)))),w(r(w(c))))
69 [hyper,63,1,6]  reachable(s(w(r(w(r(r(c))))),w(c))
72 [hyper,69,1,3]  reachable(s(r(w(r(r(c)))),w(w(c))))
75 [hyper,72,1,5]  reachable(s(r(r(c)),r(w(w(w(c))))))
77 [hyper,75,1,4]  reachable(s(r(r(r(c))),w(w(w(c))))
78 [binary,77.1,2.1]  $F.$
Pegs vs. # of Moves in Solution

<table>
<thead>
<tr>
<th>Pegs of Each Color</th>
<th># of Moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td>n</td>
<td>$n^2 + 2n$</td>
</tr>
</tbody>
</table>
Determining the Move Sequence

- The previous proofs only showed that the puzzle could be solved for those variations.

- The actual move sequence would have to be dug out from the proof steps.

- We can modify the rules so that the move sequence is obtained as a byproduct.
Determining the Move Sequence

- Use function composition to represent accumulated move sequence.
- Revised rules (4-pegs, where specific):
  - move(s(w(x), y), s(x, w(y)), z, wm(z)).
  - move(s(x, r(y)), s(r(x), y), z, rm(z)).
  - move(s(r(w(x)), y), s(x, r(w(y))), z, wj(z)).
  - move(s(x, w(r(y))), s(w(r(x)), y), z, rj(z)).
  - reachable(s(w(w(w(w(c))))), r(r(r(r(c))))), d).
  - -reachable(x, z) | -move(x, y, z, zz) | reachable(y, zz).
  - -reachable(s(r(r(r(r(c))))), w(w(w(w(w(c))))), z) #answer(z).
Move Sequence Read Inside-Out

- For 4 pegs of each color:
  $answer(rm(wj(wm(rj(rj(rm(wj(wj(wj(wm(rj(rj(rj(wm(wj(wj(rm(rj(rj(wm(wj(rm(d))))))))))))))))))))).

- The sequence is:
  rm wj wm rj rj rm wj wj wj wj wj rm rj rj
  rj rj wj wj wj wj rm rj rj rm wj wj rm

- (For this puzzle, the move sequence is coincidentally a palindrome.)
Herbrand’s Theorem: Why Resolution Works

• Jacques Herbrand showed the following (1930):
  • A set of clauses is unsatisfiable iff there is a refutation using a certain kind of **symbolic interpretation** (known as a **Herbrand Interpretation**):
    • For each constant symbol, the interpretation is literally that symbol. (If no constant symbols, add one.)
    • Functions are defined as follows:
      \[ I[f(t_1, \ldots, t_n)] = \text{the string } f(I[t_1], \ldots, I[t_n]) \]

http://mathworld.wolfram.com/HerbrandUniverse.html
http://en.wikipedia.org/wiki/Herbrand_interpretation
Ground Terms and Clauses

• A **ground term** is a term in which there are no variables.

• A **ground clause** is a clause in which there are no variables.
Herbrand’s Theorem

If there is a refutation at all, there is one using only ground clauses from a Herbrand interpretation.
Other Fine Points of Resolution

- Treat clauses as sets (reduce when necessary).

- Factoring may be necessary.
Remember to treat clauses as sets.

- $q(b, X) \lor p(X) \lor q(b, a)$
- $\neg q(Y, a) \lor p(Y)$
- These are already renamed apart.

- unify $q(b, X)$ with $\neg q(Y, a)$
- mgu is $[a/X, b/Y]$

- Modified clauses:
  - $q(b, a) \lor p(a) \lor q(b, a)$
  - $\neg q(b, a) \lor p(b)$

- There are **two** instances of $q(b, a)$ in the first clause; both are removed in resolving.
- Resolvent: $p(a) \lor p(b)$
Binary Resolution and Factoring

- What we have seen so far is “binary” resolution — unifying two literals to achieve a resolvent.

- In general, binary resolution is not enough.

- We might need to “factor” two or more literals in the same clause to make progress.
Factoring

- Two or more literals of the same sign in one clause can be unified (before renaming apart) so that the resulting literals can be collapsed into one.

- The resulting clause is called a factor of the original.

- The factor (with all variables quantified) is logically implied by the more-general original (with all variables quantified).
Factoring Example

- Consider the clause:
  \[ P(x) \lor P(f(y)) \lor \neg Q(x) \]

- The first two literals can be unified using the substitution \([f(y)/x]\).

- The resulting factor is:
  \[ P(f(y)) \lor \neg Q(f(y)) \]

- \[(\forall x \forall y (P(x) \lor P(f(y)) \lor \neg Q(x))) \rightarrow \forall y (P(f(y)) \lor \neg Q(f(y)))\]
  is valid.
Use of Factoring

- Suppose our clause set includes:

  \[ P(x) \lor P(f(y)) \lor \neg Q(x) \]
  \[ \neg P(f(a)) \]

- With binary resolution, we’d get the resolvent:
  \[ P(x) \lor \neg Q(x) \].

- If we **first factor**, to get \( P(f(y)) \lor \neg Q(f(y)) \) as on the previous slide, we can get a resolvent
  \[ \neg Q(f(a)) \]
  which is more helpful (as a unit clause).
Full Resolution of Two Clauses

- Binary resolution of the clauses.
- Binary resolution of one clause with a factor of the other.
- Binary resolution of factors of both clauses.
Case Where Factoring is Necessary

\[ P(x) \lor P(y) \]
\[ \neg P(a) \lor \neg P(b) \]

- Without factoring, generate:
  \[ P(y) \lor \neg P(b) \]
  \[ P(x) \lor \neg P(a) \]

- and other similar clauses, but never the empty clause.
Case Where Factoring is Necessary

\[ P(x) \lor P(y) \]
\[ \neg P(a) \lor \neg P(b) \]

- With factoring, get factor \( P(x) \) from first clause, then resolve twice to get:

\[ \neg P(b) \text{ then } \perp \]
Subsumption

- A clause $C$ **subsumes** a clause $D$ if there is a substitution $\theta$ such that $C\theta \subseteq D$, where we interpret the clauses as **sets** of their literals.

- If a clause $D$ in a set of clauses is subsumed by another clause $C$ **within the set**, then we can delete $D$ from the set without affecting the case of whether the empty clause $\bot$ is derivable.

The subsuming clause is more general.
Subsumption Examples

• $P(X)$ subsumes $P(X) \lor Q(Y)$ by the empty substitution $[]$.

• $\neg P(X) \lor Q(f(X))$ subsumes $\neg P(Z) \lor \neg P(h(Y)) \lor Q(f(h(Y)))$ by the substitution $[h(Y)/X, h(Y)/Z]$. 