Resolution Theorem Proving
Robert Keller
March 2010

What is this?
- Resolution is a special kind of theorem proving used in:
  - Automated theorem proving and reasoning
  - Answer extraction and databases
  - Prolog language
- Resolution in itself is a complete proof rule for refutation.

How it works
- A special stripped-down representation is used: “clausal form”.
- Quantifiers have been eliminated.
- A formula is proved by refutation, i.e. showing that its negation is unsatisfiable (as with the tableau method).

Two Types of Resolution
- Predicate calculus resolution:
  - Our main objective
- Propositional resolution:
  - Needed to understand predicate resolution
  - Used in algorithms and complexity theory (NP completeness, for example)

Propositional Version 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.

Example
- Clause set:
  - p ∨ ¬q
  - ¬q ∨ ¬r
  - q
- This clause set is satisfiable:
  - Valuation p = T, q = T, r = F will satisfy it.
Example

- 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 formulas \( T \) at the same time.
  - Why not?
    We'd need \( p = r = F \), but then there is no way to set \( q \).

Purpose

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

- It represents a formula in conjunctive normal form:
  - a conjunction of disjunctions of literals
  - the set clauses

Equivalence of Propositional Formulas

- Two propositional formulas are equivalent (\( \equiv \)) 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)

- What about \( T \)?
  - \( T \) in clausal form is the empty set of clauses: \( \{ \} \) (no clauses)

The Empty Clause

- The empty clause is \( \bot \), sometimes denoted by an empty box \( \square \).

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

  Example: \( \{ \neg p \lor q, p, \bot \} \)
Check CNF from Truth Table

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>r</th>
<th>value</th>
<th>p\lor q\lor r</th>
<th>p\land q\land r</th>
<th>\neg p\lor \neg q\lor \neg r</th>
<th>conj.</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

\(p\lor q\lor r = p\land q\land r\lor p\land q\land r\lor p\land q\land r\lor \neg p\lor \neg q\lor \neg r\lor \neg p\lor \neg q\lor \neg r\lor \neg p\lor \neg q\lor \neg r\)
reduce example
\[
\text{reduce}((p \lor q \lor \neg p, \\
p \lor q \lor \neg p \lor q)) = \\
(p \lor \neg q)
\]

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 null clause \(\bot\).
- There is no interpretation that satisfies \(\bot\), much less \(\bot\) together with other clauses.

How Resolution Works
- Do Repeatedly:
  - From the set of clauses, pick a pair from which a new clause, called the "resolvent", can be created.
- Add the resolvent to the set.
- If \(\bot\) is ever added to the set, 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 both
  - \(p \lor \varphi\)
  - \(\neg p \lor \psi\)
- where \(\varphi\) and \(\psi\) are formulas (either could be empty), then the resolvent is
  \[\varphi \lor \psi,\]
- \(p\) and \(\neg p\) are said to be "clashing" literals.

Resolution as a Deduction Rule
\[p \lor \varphi \quad \neg p \lor \psi \quad \varphi \lor \psi\]
where \(p\) is any proposition symbol and \(\varphi\) and \(\psi\) are clauses.

Example of Resolvents
- Consider the clauses
  - \(p \lor -q\)
  - \(q \lor r \lor -s\)
- A resolvent (based on literal \(q\)) is:
  - \(p \lor r \lor -s\)
Example of Resolvents

- Consider the clauses
  - \( p \lor r \)
  - \( q \land r \land s \)
- Since \( r \) and \( \neg r \) occur in different clauses, a resolvent is:
  - \( p \lor \neg q \land r \lor \neg q \lor \neg s \)
- Another (based on literal \( r \)) is:
  - \( p \lor q \land \neg q \lor \neg s \)
- Both of these will be dropped in reducing, however.

Example 1

- \( S = \{ p \lor \neg q, q \lor r, -p, -r \} \)
- \( T = \text{reduce}(\text{resolveall}(S)) = \{ p \lor r, q, -q \} \)
- \( \bot \notin S \land T - S \neq \emptyset \)
- \( S = \{ p \lor \neg q, q \lor r, -p, -r, p \lor r, -q, q \} \)
- \( T = \text{reduce}(\text{resolveall}(S)) = \{ p \lor r, -q, q, \ldots, \bot \} \)
- Stop \( \bot \in S \).
- \( S \) is unsatisfiable, since \( \bot \in S \).

Example of Resolvents

- Consider the clauses
  - \( p \)
  - \( -p \)
- Since \( p \) and \( -p \) occur in different clauses, the resolvent is:
  - \( \bot \)

Resolution Algorithm (Crude form)

- Input: \( S \), the clause set to be tested.
  - \( S := \text{reduce}(S) \);
  - \( T := \text{reduce}(\text{resolveall}(S)) \);
  - while\( \bot \notin S \land T - S \neq \emptyset \)
    - \( S := S \cup T \);
    - \( T := \text{reduce}(\text{resolveall}(S)) \);
  - \( \)\n  - if \( \bot \in S \), then unsatisfiable, else satisfiable.
  - where \( \text{resolveall}(S) = \{ \varphi \lor \psi \mid (p \lor \varphi) \in S \land (\neg p \lor \psi) \in S \} \)

Example 2

- \( S = \{ p \lor \neg q, q \lor r, -p \} \)
- \( T = \text{reduce}(\text{resolveall}(S)) = \{ p \lor r, -q \} \)
- \( \bot \notin S \land T - S \neq \emptyset \)
- \( S = \{ p \lor \neg q, q \lor r, -p, p \lor r, -q, q \} \)
- \( T = \text{reduce}(\text{resolveall}(S)) = \{ p \lor r, q, -q, r \} \)
- Stop \( T - S = \emptyset \).
- \( S \) is satisfiable, since \( \bot \notin S \).
Adding the resolvent does not alter satisfiability

- A reduced clause set \( \Gamma \cup \{ p \lor q, \neg p \lor q \} \)
  is equivalent to \( \Gamma \cup \{ p \lor q, \neg p \lor q, \neg q \lor q \} \)

- Reason: Suppose \( \nu \) is a valuation that satisfies both \( p \lor q \) and \( \neg p \lor q \).
  But since \( \nu \) satisfies \( \neg p \lor q \), \( \nu(q) = T \).
  The case for \( \nu(p) = F \) is symmetric: \( \nu(q) = T \).
  So \( \nu(q \lor q) = T \).

Adding the resolvent does not alter satisfiability (converse)

- A reduced clause set \( \Gamma \cup \{ p \lor q, \neg p \lor q \} \)
  is equivalent to \( \Gamma \cup \{ p \lor q, \neg p \lor q, \neg q \lor q \} \)

- Converse: If \( \nu \) satisfies \( q \lor q \) (which contains neither \( p \) nor \( \neg p \)), then it must satisfy one or the other of \( q \) or \( q \).

  - If \( \nu \) satisfies \( q \lor q \), then it also satisfies \( p \lor q \).
    Extend \( \nu \) to \( \nu' \) such that \( \nu'(p) = F \). Then \( \nu'(-p \lor q) = T \).
    - If \( \nu \) satisfies \( q \lor q \), so extend \( \nu \) to \( \nu' \) such that \( \nu'(p) = T \). Then \( \nu'(p \lor q) = T \).

Resolution Algorithm Loop Invariant

- \( S := \text{reduce}(S_0) \)
  \( T := \text{reduce}(\text{resolveall}(S)) \)
  while( \( \bot \not\in S \land T - S \neq \emptyset \) )
  
  \( \{ \begin{align*} 
  S &:= S \cup T; \\
  T &:= \text{reduce}(\text{resolveall}(S)); 
  \end{align*} \)
  
  If \( \bot \in S \), then unsatisfiable, else satisfiable.

- Invariant:
  \( S \) is unsatisfiable \iff \( S_0 \) is unsatisfiable

Resolution Algorithm Termination

- 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 as a Tree

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
Try these clause sets:

- \( p \lor \neg q \lor r, \)
  - \( q \lor \neg r, \)
  - \( \neg p \)
- \( \neg p \lor \neg q \lor \neg r, \)
  - \( \neg q \lor r, \)
  - \( q \lor s, \)
  - \( \neg s, \)
  - \( p \)

Sometimes a DAG is more appropriate than a tree for showing all options

We avoid identifying the two \( \bot \) nodes, so as not to confuse the two sets of antecedents.

Resolution is a Complete Rule

- The single rule of resolution is **refutation-complete**: If a set of clauses is unsatisfiable, this can be determined using only the resolution rule.
- However, considerable logic went into getting everything into **clausal form** in the first place, so it is perhaps **unfair** to compare the single rule to the set of natural deduction rules, which cover all logical steps.

Resolution Algorithm Refinement 2

- The idea is to avoid revisiting pairs that were resolved in earlier steps.
- Input: \( S \), the clause set to be tested.
  - \( S := \text{reduce}(S); \)
  - \( T := \text{reduce}((\text{resolveall}(S, S)) - S); \)
  - \( \text{while}(\bot \not\in S \land T \neq \emptyset) \)
    - \( S := S \cup T; \)
    - \( T := \text{reduce}((\text{resolveall}(S, T)) - S); \)
  - \( \text{resolveall}(S, T) \)
    - \( = \{ \phi \lor \psi \mid (p \lor \psi) \in S \land (\neg p \lor \psi) \in T \} \)
    - \( \cup \{ \phi \lor \psi \mid (p \lor \psi) \in T \land (\neg p \lor \psi) \in S \} \)

Further Useful Optimizations

- **Uncomplemented Literal Lemma** (Lemma 4.9 of Ben-Ari):
  - 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.
  - **Rationale**: The literal in question can be assigned \( T \) in a satisfying interpretation, without being constrained by the other literals.

Example of Uncomplemented Literal Lemma

- \( \neg p \lor q \lor r, \)
  - \( \neg q \lor r, \)
  - \( q \lor s, \)
  - \( \neg s, \)
  - \( p \)
- \( r \) occurs only uncomplemented.
- The clause set is unsatisfiable iff the following set is:
  - \( q \lor s, \)
  - \( \neg s, \)
  - \( p \)
Further Useful Optimizations

Unit Clause Lemma (Lemma 4.11 of Ben-Ari):

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 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 contribute nothing to the individual clauses.

Example of Unit Clause Lemma

\(-p \lor q \lor r, \neg q \lor r, q \lor s, \neg s, p\)

• \(-s\) is a unit clause. The complement of \(-s\) is s.
• The clause set is unsatisfiable iff the following set is:

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

Further Useful Optimizations

Subsumption Lemma (Lemma 4.15 of Ben-Ari):

• One clause subsumes another if the former’s literals are a subset of the latter’s.
• If one clause of a set subsumes another, the subsumed clause can be dropped from the set.

Rationale: If C subsumes D, then any interpretation satisfying C must also satisfy D (because the literals are disjointed). Thus the satisfiability of the set of clauses is unaffected if D is removed.

Example of Subsumption Lemma

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

• The second clause subsumes the first.
• The clause set is unsatisfiable iff the following set is:

\(-p \lor r, p \lor \neg r\)

Common Special Case of Clause Set

• Often we want to prove a sequent such as:

\(\psi_1 \land \psi_2 \land \ldots \land \psi_m \rightarrow \phi_1, \phi_2 \land \ldots \land \phi_n \rightarrow \chi_1 \land \chi_2 \land \ldots \land \chi_p\)

where each symbol represents a literal.

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

\(\neg \psi_1 \lor \neg \psi_2 \lor \ldots \lor \neg \psi_m \lor \neg \phi_1, \neg \phi_2 \lor \ldots \lor \neg \phi_n \lor \neg \chi_1 \lor \neg \chi_2 \lor \ldots \lor \neg \chi_p\)

Selective Resolution

• Rather than resolving all pairs of clauses, try to pick pairs that will produce \(\bot\) in the fewest number of steps.

• Consider using a "non-deterministic" expression of the algorithm (pick clauses to resolve).
Strategic Optimizations

- **Unit-Preference**: If possible, chose to resolve with unit clauses. These reduce the size of resulting clauses.

- **Set-of-Support**: Divide the clauses into two sets:
  - A known-satisfiable subset.
  - Other
- Always resolve with an "other" or a clause derived from one.
- These clause are called the "set of support" (SOS).

Set-of-Support

- Showing that the following clause set is unsatisfiable:

  $\{\neg \phi_1 \lor \neg \phi_2 \lor \ldots \lor \neg \phi_{m_1} \lor \psi_1, \neg \phi_2 \lor \ldots \lor \neg \phi_{m_2} \lor \psi_2, \ldots \}$

  - Satisfiable "axioms"

  $\{\neg \phi_1 \lor \neg \phi_2 \lor \ldots \lor \neg \phi_{m_n} \lor \psi_n, \ldots \}$

  - Set of support

Horn Clauses

- A Horn clause is one in which there is at most one non-negated literal:

  - $\neg \phi_1 \lor \neg \phi_2 \lor \ldots \lor \neg \phi_m \lor \psi$ (one non-negated)
  - or $\neg \phi_1 \lor \neg \phi_2 \lor \ldots \lor \neg \phi_m$ (no non-negated)

- Horn clause are the basis of the Prolog language, where:

  - $\neg \phi_1 \lor \neg \phi_2 \lor \ldots \lor \neg \phi_m \lor \psi$ is written
  - $\psi :\! - \phi_1, \phi_2, \ldots, \phi_m$. If $m = 0$, then we just write
  - $\psi$.

Prolog uses a special form of resolution to do its work ("SLD" resolution)

- Dialog with Prolog:

  ```
  consult(user).
  p :\! - r, s.
  r :\! - q.
  s :\! - q.
  q.
  ^D
  | ?- p.
  yes
  ```

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 implicitly $\forall$-quantified.

  - The variables of each clause are thus independent from the other clauses; even if they are they 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

  $\forall X p(X)$

  $\forall X \forall Y (q(X, Y) \lor q(X, X) \lor p(X))$

  which is the same as

  $\forall X \forall Y (\neg q(X, X) \lor p(X))$

  i.e. the clause set

  - $\{p(X_1), q(X_0, Y_1), \neg q(X_0, X_2) \lor p(X_1)\}$

  - $\{p(X_1), q(X_0, Y_2), \neg q(X_0, X_2) \lor p(X_1)\}$
How General is This?

- We will see later that it is very general, as far as showing unsatisfiability is concerned.

Examples of Predicate Clausal Form

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

- These clauses can be used to prove the syllogism:
  - All men are mortal.
  - Socrates is a man.
  - 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, e.g. \( p(X) \) in
  \[
  \neg q(X, X) \lor p(X) \\
  \neg p(X) \lor r(X, Y)
  \]
  - 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 substitution of terms for their variables that makes them identical.

- If such a substitution 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).
- These terms are **unifiable**, since the substitution \([c/X]\) (\( c \) for \( X \)) makes them identical.
- Consider \( q(c, d), q(X, X) \) (\( c \) and \( d \) are constants).
- These terms are **not unifiable**, since distinct constant symbols do not unify. There is no substitution that will make them identical.

Literals from Different Clauses

- Remember that variables are not identified across clauses.
- If we are considering whether two literals in different clauses unify, we first must **rename the variables** so that there is no overlap.
- This is called "renaming apart" the clauses.

Literals from Different Clauses

- 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.

More Unification Examples

<table>
<thead>
<tr>
<th>Term 1</th>
<th>Term 2</th>
<th>Unifiable?</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>Unifiable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p(X, Y) )</td>
<td>( p(f(Z), g(Z)) )</td>
<td></td>
</tr>
<tr>
<td>( p(X, f(X)) )</td>
<td>( p(f(Z), U) )</td>
<td></td>
</tr>
<tr>
<td>( p(f(X), g(X)) )</td>
<td>( p(f(U), U) )</td>
<td></td>
</tr>
<tr>
<td>( p(f(X), f(X)) )</td>
<td>( p(c, c) )</td>
<td></td>
</tr>
<tr>
<td>( p(f(X), g(X)) )</td>
<td>( p(Y, g(Y)) )</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: \([f(Z)/X]\) and each of the original literals would unify with this.
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.
  \[ \tau = p(X, g(Y, W)) \]
  \[ \tau \beta = p(Z, g(f(Z, c), c)) \]

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.
  \[ \tau = p(X, g(Y, W)) \]
  \[ \beta = [Z/X, f(Z, c)/Y, c/W] \]
  \[ \gamma = [V/Z] \]
  \[ (\tau \beta)\gamma = p(V, g(f(V, c), c)) \]

• 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

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 it 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>

Note on Unification in Prolog

• In Prolog, unification is used in goal matching and in the ":=" operator.

• However, Prolog's unification is slightly abridged: it bypasses the "occur check":
  \[ X = f(X) \]
will unify in Prolog, but not in ordinary unification. In effect, \( X \) gets bound to the infinite term:
  \( f(f(f(\ldots)) \)

MGU Algorithm (Martelli & Montanari)

- Input: Two terms, or two atoms, \( t_1, t_2 \), already renamed apart.
- Output: Either the most general unifier for \( t_1, t_2 \), or "not unifiable".

\[ S := \{ t_1 = t_2 \} \]
// functions as a sort-of stack
\[ \mu := \text{the empty substitution}; \]

while \( S \neq \emptyset \)
  remove a pair \( \{ L, R \} \) from \( S \);
  // pop
  case
    if \( L = R \)
      do nothing;
    else if \( L = f(s_1, s_2, \ldots, s_n) \) and \( R = f(t_1, t_2, \ldots, t_n) \)
      \( S := S \cup \{ s_1 = t_1, s_2 = t_2, \ldots, s_n = t_n \} \);
    // pushes
    else if \( L = x \) where \( x \) is a variable not occurring in \( R \)
      \( \mu := \mu \{ R/x \} \);
    // composing
    else if \( R = x \) where \( x \) is a variable not occurring in \( L \)
      \( \mu := \mu \{ L/x \} \);
    else return "not unifiable";
  // case

return \( \mu \) as the MGU;
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
  - \( S := \{ [X, Y], [f(X), f(Y)] \} \) case 3
  - \( \mu := [Y/X]; S := \{ [f(Y), f(Y)] \} \) case 1
  - Remove \( [f(Y), f(Y)] \)
  - \( S := \{} \) case 4
  - Result: unifiable with \( \mu = [Y/X] \)

Diagrammatically

- \( p(X, f(X)) \) substitution \( [Y/X] \)
- \( p(Y, f(Y)) \)

Example

- \( p(X, f(X)) \) vs. \( p(f(Y), Y) \)
  - Initial
  - \( S := \{ [p(X, f(X)), p(f(Y), Y)] \} \)
  - \( \mu := \{ \} \)
  - Remove \( [p(X, f(X)), p(f(Y), Y)] \) case 2
  - \( S := \{ [X, f(Y)], [f(X), Y] \} \) case 3
  - \( \mu := [f(Y)/X]; S := \{ [f(Y), Y] \} \) case 5
  - Remove \( [f(Y), Y] \)
  - \( S := \{} \) case 6
  - Result: not unifiable

Diagrammatically

- \( p(X, f(X)) \) substitution \( [f(Y)/X] \)
- \( p(Y, f(Y)) \) occur check fails, not unifiable
- \( p((Y), f((Y))) \)

Example

- \( p(X, g(Z), X) \) vs. \( p(f(Y), Y, W) \)
  - Initial
  - \( S := \{ [p(X, g(Z), X), p(f(Y), Y, W)] \} \)
  - \( \mu := \{ \} \)
  - Remove \( [p(X, g(Z), X), p(f(Y), Y, W)] \) case 2
  - \( S := \{ [X, f(Y)], [g(Z), Y], [X, W] \} \) case 3
  - \( \mu := [f(Y)/X]; S := \{ [g(Z), Y], [f(Y), W] \} \) case 4
  - Remove \( [g(Z), Y] \)
  - \( \mu := [f(g(Z))/X, g(Z)/Y]; S := \{ [f(g(Z)), W] \} \) case 5
  - Remove \( [f(g(Z)), W] \)
  - \( \mu := [f(g(Z))/X, g(Z)/Y, f(g(Z))/W]; S := \{ \} \) case 4
  - Result: unifiable with \( \mu = [f(g(Z))/X, g(Z)/Y, f(g(Z))/W] \)
Diagrammatically

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

Checking Uniﬁability with Prolog

- As long as there are no occur-check violations, can use \( = \) to test.

```
bash-3.2$ /opt/local/bin/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))
```

Try These

<table>
<thead>
<tr>
<th>( \tau_1 )</th>
<th>( \tau_2 )</th>
<th>mgu (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(f(Y), Y) )</td>
<td></td>
</tr>
<tr>
<td>( p(f(g(X)), X) )</td>
<td>( p(f(f(h(Z))), h(Z)) )</td>
<td></td>
</tr>
</tbody>
</table>

Resolving Predicate Calculus Clauses

- Resolvable clauses must contain literals with the same predicate symbol but of opposite sign (one negated, the other not).
- Pick two such literals, one from each clause.
- Rename the clauses apart.
- 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 literals.

Example of Predicate Resolution

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

Complete Predicate Resolution Process

- The process is the same as for the propositional case, except that we have to rename variables, then unify literals prior to resolution and apply the mgu to all literals in the two clauses, before obtaining the resolvent.
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 first literals 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)
- Try This Set
  1. \neg e(X) \lor q(X) \lor s(X, f(X))
  2. \neg e(X) \lor q(X) \lor r(f(f(X)))
  3. p(a)
  4. e(a)
  5. \neg s(a, Y) \lor p(Y)
  6. \neg p(X) \lor \neg q(X)
  7. \neg p(X) \lor \neg r(X)

Try This Set
- \neg p(X) \lor \neg q(X)
- \neg s(a, Y) \lor p(Y)
- \neg p(X) \lor \neg q(X)
- Binary Resolution and Factoring
  - What we have seen so far is “binary” resolution — unifying two literals to achieve a resolvent.
  - In general, binary resolution alone might not be enough.
  - We might need to “factor” two or more literals in the same clause to make progress.

Example where there is more than one instance of a literal to remove
- q(b, X) \lor p(X) \lor q(b, a)
- 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)

Factoring
- Within a single clause, two or more literals of the same sign can be unified 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:
  \[ P(f(y)) \lor \neg Q(f(y)) \]
- \((\forall x \forall y (P(x) \lor P(f(y)) \lor \neg Q(x))) \implies (\forall y (P(f(y)) \lor \neg Q(f(y))))\) 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 better.

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 more similar clauses, but never the empty clause.

Clausal Form for Predicate Logic

- Often, we’ll want to prove a sequent of the form
  - \(\forall x \forall y (...)\)
  - \(\forall x \forall y (...) \vdash ...\)
  - For premises of the form \(\forall x \forall y (...)\) where \(...) 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 Clauses

1. \neg fungus(X) \lor mushroom(X) \lor toadstool(X)
2. \neg boletus(X) \lor fungus(X)
3. \neg toadstool(X) \lor poisonous(X)
4. \neg boletus(X) \lor \neg mushroom(X)
5. boletus(b)
6. \neg poisonous(b) (negated conclusion)

Mushroom Clauses in Otter

set(auto),
list(usable).
-fungus(x) | mushroom(x) | toadstool(x).
-boletus(x) | fungus(x).
-toadstool(x) | poisonous(x).
-boletus(x) | \neg mushroom(x).
boletus(b).
-poisonous(b).
end_of_list.

Using Otter or Prover9

- Otter was a sophisticated resolution theorem prover developed at Argonne National Laboratory.
- It will accept input in either clausal form, or non-clausal form.
- It has been succeeded by prover9:
  http://www.cs.unm.edu/~mccune/prover9/
- There is an on-line version Otter-\lambda which has enhancements of Otter for the \lambda calculus:
  http://www.michaelbeeson.com/research/otter-lambda/

Otter’s Output for Mushrooms

------------- PROOF -------------
1 [ ] \neg fungus(x) | mushroom(x) | toadstool(x).
2 [ ] \neg boletus(x) | fungus(x).
3 [ ] \neg toadstool(x) | poisonous(x).
4 [ ] \neg boletus(x) | \neg mushroom(x).
5 [ ] \neg poisonous(b).
6 [ ] boletus(b).
7 [hyper,6,2] fungus(b).
8 [hyper,7,1] mushroom(b) | toadstool(b).
9 [hyper,8,3,unit_del,1] mushroom(b).
10 [hyper,9,4,6] SF... [SF indicates the null clause.
------------- end of proof -------------
Checking Unifiability with Otter

- In contrast to Prolog, Otter uses an occur-check.

\[ \text{set(auto).} \]
\[ \text{set(prolog_style_variables).} \]
\[ \text{list(usable).} \]
\[ p(X, g(X)) \mid \text{ans}(X). \]
\[ \text{-p(f(Y), Y).} \]
\[ \text{end_of_list.} \]

Clausal Form for Predicate Logic

- Often, we’ll want to prove a sequent of the form

\[ \forall x \forall y \ldots, \quad \forall x \forall y \ldots \]

- 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, so that will become

\[ \neg \forall x \forall y \ldots \] which is equivalent to \( \exists x \exists y \ldots \)

**We cannot simply drop the quantifiers in this case!!**

Clausal Form for Predicate Logic

- Consider the sequent

\[ \forall y \, p(y) \mid \neg \forall y \, p(x) \]

- 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 \( \exists x \, \neg p(x) \).

Colloquially, we get to pick the value for \( b \) in finding a satisfying interpretation, just as we get to pick the value for \( x \) in \( 3x \).

Another Example

- Consider the sequent

\[ \exists x \, \forall y \, p(x, y) \mid \neg \forall y \, \exists x \, p(x, y) \]

- Premise clause:

\[ p(b, y) \]

- Conclusion clause:

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

Resolution produces \( \bot \) in 1 step.
Skolem Functions for the General Case

- \( \forall x \forall y \ldots \exists y \ldots \)
- \( y \) 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 \( v \) 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.
- We love Skolem!

Skolem with Arguments Example

- Prove: "The composition of two onto [surjective] functions is onto."
- Represent the two functions as binary predicates. \( F(x, y) \) means \( y \) is the image of \( x \).
- "\( F \) is onto": \( \forall y \exists x F(x, y) \)
- "\( G \) is onto": \( \forall z \exists y G(y, z) \)
- "\( H \) is the composition of \( F \) and \( G \):
  \[ \forall x \forall y \forall z ((F(x,y) \land G(y,z)) \rightarrow H(x, z)) \]
  \[ \land \forall x \forall z (H(x, z) \rightarrow \exists y (F(x,y) \land G(y,z))) \]
- "\( H \) is onto": \( \forall z \exists x H(x, z) \)

Translation to Clausal Form

- \( \forall y \exists x F(x, y) \) becomes \( F(f(x), y) \)  [\( f \) 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

(3 and 4 were not needed in the proof.)

How to get a clause form in general?

- First convert the formula into "prenex form" (all quantifiers are outside on the left). [The parts of this form are 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 \( \land, \lor, \neg \) with their \( \land, \lor, \neg \) counterparts.
- Push negations inward
- Pull quantifiers to the outside using the rules on the next page.
Basic Prenex Quantifier Rules (for pulling quantifiers to the outside)

- Here ⇒ 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:
    1. \((\exists x) F[x] \land (\forall x) G[x] \Rightarrow (\exists x) (F[x] \land G[x])\)
    2. \((\exists x) F[x] \lor (\forall x) G[x] \Rightarrow (\exists x) (F[x] \lor G[x])\)

Justification of Rules Using Natural Deduction

Proviso is introduced by unifying \(G\) with \(G[x]\) in \(\forall x G[x]\) in JAPE.

Non-empty universe assumption, needed in 7-11

Example of Prenex Conversion

- \(\forall x \forall y ((\exists z (p(x, z) \land p(y, z))) \land \exists u q(x, y, u))\)
- \(\forall x \forall y ((\exists z (p(x, z) \land p(y, z))) \land \exists u q(x, y, u))\)
- \(\forall x \forall y ((\forall z (p(x, z) \land p(y, z))) \land \exists u q(x, y, u))\)
- \(\forall x \forall y ((\forall z (\neg p(x, z) \lor \neg p(y, z))) \land \exists u q(x, y, u))\)
- \(\forall x \forall y ((\forall z (\neg p(x, z) \lor \neg p(y, z))) \land \exists u q(x, y, u))\)
- \(\forall x \forall y (\exists u (\neg p(x, z) \lor \neg p(y, z))) \land q(x, y, u))\)
- \(\forall x \forall y (\exists u (\neg p(x, z) \lor \neg p(y, z))) \land q(x, y, u))\)
- \(\forall x \forall y (\exists u (\neg p(x, z) \lor \neg p(y, z))) \land q(x, y, u))\)
- \(\forall x \forall y (\exists u (\neg p(x, z) \lor \neg p(y, z))) \land q(x, y, u))\)
- \(\forall x \forall y (\exists u (\neg p(x, z) \lor \neg p(y, z))) \land q(x, y, u))\)
- \(\forall x \forall y (\exists u (\neg p(x, z) \lor \neg p(y, z))) \land q(x, y, u))\)
Completion of Conversion

- \( \forall x \forall y \forall z \exists u (\neg p(x, z) \lor \neg p(y, z) \lor q(x, y, u)) \)
- Skolemize as \( f(x, y, z) \) and drop \( \forall x \forall y \forall z \)
- \( \neg p(x, z) \lor \neg p(y, z) \lor q(x, y, f(x, y, z)) \)

Example: Group Theory Clauses

- \( f \) is the group operation, \( e \) is the equality predicate
- \( \forall x \forall y \exists (f(x, f(y, z)), f(f(x, y), z)) \) 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 \( e(x, x) \)
- \( \forall x \forall y \forall v \forall w (e(x, y) \land e(v, w) \to e(f(x, v), f(y, w))) \) becomes \( \neg e(x, y) \lor \neg e(v, w) \lor e(f(x, v), f(y, w)) \)
- \( \forall x \forall y e(x, y) \to e(y, x) \) becomes \( \neg e(x, y) \lor e(y, x) \)

Equality in Otter, Paramodulation

- Otter has a built-in equality, so the approach illustrated in the previous example is not generally necessary.
- Equality can be handled, for example, by the "paramodulation" rule, which essentially captures one of the two ND rules for equality:

\[
\frac{\alpha \lor (x = r)}{\beta \lor \gamma[r]} \quad \gamma[r] \text{ is a literal containing term } r \\
\beta = \text{unify}(\alpha, \beta)
\]