First-Order Logic

Resolution
Resolution for predicate logic

Gilmore’s algorithm is correct and complete, but useless in practice.

We upgrade resolution to make it work for predicate logic.
Recall: resolution in propositional logic

Resolution step:

\[
\{L_1, \ldots, L_n, A\} \quad \frac{}{\{L'_1, \ldots, L'_m, \neg A\}} \quad \{L_1, \ldots, L_n, L'_1, \ldots, L'_m\}
\]

Resolution graph:

\[
\{\neg A, B\} \quad \frac{}{\{A\}} \quad \{\neg B\}
\]

\[
\{B\} \quad \frac{}{\square}
\]

A set of clauses is unsatisfiable iff the empty clause can be derived.
Adapting Gilmore’s Algorithm

Gilmore’s Algorithm:

Let $F$ be a closed formula in Skolem form and let $F_1, F_2, F_3, \ldots$ be an enumeration of $E(F)$.

$n := 0$
repeat $n := n + 1$
until $(F_1 \land F_2 \land \ldots \land F_n)$ is unsatisfiable;

– this can be checked with any calculus for propositional logic

return “unsatisfiable”

“any calculus” $\rightsquigarrow$ use resolution for the unsatisfiability test
Terminology

Literal/clause/CNF is defined as for propositional logic but with the atomic formulas of predicate logic.

A ground term/formula/etc is a term/formula/etc that does not contain any variables.

An instance of a term/formula/etc is the result of applying a substitution to a term/formula/etc.

A ground instance is an instance that does not contain any variables.
Clause Herbrand expansion

Let $F = \forall y_1 \ldots \forall y_n F^*$ be a closed formula in Skolem form with $F^*$ in CNF, and let $C_1, \ldots, C_m$ be the clauses of $F^*$.

The clause Herbrand expansion of $F$ is the set of ground clauses

$$CE(F) = \bigcup_{i=1}^{m} \{ C_i[t_1/y_1] \ldots [t_n/y_n] \mid t_1, \ldots, t_n \in T(F) \}$$

**Lemma**

$CE(F)$ is unsatisfiable iff $E(F)$ is unsatisfiable.

**Proof** Informally speaking, “$CE(F) \equiv E(F)$”.

Ground resolution algorithm

Let $F$ be a closed formula in Skolem form with $F^*$ in CNF. Let $C_1, C_2, C_3, \ldots$ be an enumeration of $CE(F)$.

\[
n := 0; \\
S := \emptyset; \\
\text{repeat} \quad n := n + 1; \\
\quad S := S \cup \{C_n\}; \\
\text{until } S \vdash_{\text{Res}} \square \\
\text{return} \text{ “unsatisfiable”}
\]

**Note:** The search for $\square$ can be performed incrementally every time $S$ is extended.

**Example**

$F^* = \{\{\neg P(x), \neg P(f(a)), Q(y)\}, \{P(y)\}, \{\neg P(g(b, x)), \neg Q(b)\}\}$
The correctness of the ground resolution algorithm can be rephrased as follows:

**Theorem**

A formula $F = \forall y_1 \ldots \forall y_n F^*$ with $F^*$ in CNF is unsatisfiable iff there is a sequence of ground clauses $C_1, \ldots, C_m = \square$ such that for every $i = 1, \ldots, m$

- either $C_i$ is a ground instance of a clause $C \in F^*$, i.e. $C_i = C[t_1/y_1] \ldots [t_n/y_n]$ where $t_1, \ldots, t_n \in T(F)$,
- or $C_i$ is a resolvent of two clauses $C_a, C_b$ with $a < i$ and $b < i$
Where do the ground substitutions come from?

Better:

- allow substitutions with variables
- only instantiate clauses enough to allow one (new kind of) resolution step

Example

Resolve \{P(x), Q(x)\} and \{\neg P(f(y)), R(y)\}
Substitutions as functions

Substitutions are functions from variables to terms: \([t/x]\) maps \(x\) to \(t\) (and all other variables to themselves).

Functions can be composed.

Composition of substitutions is denoted by juxtaposition: \([t_1/x][t_2/y]\) first substitutes \(t_1\) for \(x\) and then substitutes \(t_2\) for \(y\).

Example

\[(P(x,y))[f(y)/x][b/y] = (P(f(y),y))[b/y] = P(f(b),b)\]

Similarly we can compose arbitrary substitutions \(\sigma_1\) and \(\sigma_2\): \(\sigma_1\sigma_2\) is the substitution that applies \(\sigma_1\) first and then \(\sigma_2\).

Substitutions are functions. Therefore

\[\sigma_1 = \sigma_2 \iff \text{for all variables } x, \ x\sigma_1 = x\sigma_2\]
Substitutions as functions

Definition
The domain of a substitution: \( \text{dom}(\sigma) = \{ x \mid x\sigma \neq x \} \)

Example
\( \text{dom}([a/x][b/y]) = \{ x, y \} \)

Substitutions are defined to have finite domain. Therefore every substitution can be written as a simultaneous substitution \([t_1/x_1, \ldots, t_n/x_n]\).
Unifier and most general unifier

Let \( L = \{L_1, \ldots, L_k\} \) be a set of literals.

A substitution \( \sigma \) is a unifier of \( L \) if

\[
L_1\sigma = L_2\sigma = \cdots = L_k\sigma
\]

i.e. if \( |L\sigma| = 1 \), where \( L\sigma = \{L_1\sigma, \ldots, L_k\sigma\} \).

A unifier \( \sigma \) of \( L \) is a most general unifier (mgu) of \( L \) if

for every unifier \( \sigma' \) of \( L \) there is a substitution \( \delta \) such that \( \sigma' = \sigma\delta \).
<table>
<thead>
<tr>
<th>Unifiable?</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(f(x)) ) &amp; ( P(g(y)) )</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>( P(x) ) &amp; ( P(f(y)) )</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>( P(x) ) &amp; ( P(f(x)) )</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>( P(x, f(y)) ) &amp; ( P(f(u), f(z)) )</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>( P(x, f(x)) ) &amp; ( P(f(y), y) )</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>( P(x, g(x), g^2(x)) ) &amp; ( P(f(z), w, g(w)) )</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>( P(x, f(y)) ) &amp; ( P(g(y), f(a)) ) &amp; ( P(g(a), z) )</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Unification algorithm

Input: a set $L \neq \emptyset$ of literals

$\sigma := []$ (the empty substitution)

while $|L\sigma| > 1$ do

Find the first position at which two literals $L_1, L_2 \in L\sigma$ differ

if none of the two characters at that position is a variable

then return “non-unifiable”

else let $x$ be the variable and $t$ the term starting at that position

if $x$ occurs in $t$

then return “non-unifiable”

else $\sigma := \sigma [t/x]$

return $\sigma$

Example

$\{ \neg P(f(z, g(a, y)), h(z)),
\neg P(f(f(u, v), w), h(f(a, b))) \}$
Correctness of the unification algorithm

Lemma

*The unification algorithm terminates.*

**Proof** Every iteration of the *while*-loop (possibly except the last) replaces a variable $x$ by a term $t$ not containing $x$, and so the number of variables occurring in $L\sigma$ decreases by one.

Lemma

*If $L$ is non-unifiable then the algorithm returns “non-unifiable”.*

**Proof** If $L$ is non-unifiable then the algorithm can never exit the loop normally.
Correctness/completeness of the unification algorithm

Lemma
If \( L \) is unifiable then the algorithm returns the mgu of \( L \)
(and so in particular every unifiable set \( L \) has an mgu).

Proof Assume \( L \) is unifiable and let \( n \) be the number of iterations
of the loop on input \( L \).
Let \( \sigma_0 = [] \), for \( 1 \leq i \leq n \) let \( \sigma_i \) be the value of \( \sigma \) after the \( i \)-th
iteration of the loop.
We prove for every \( 0 \leq i \leq n \):
(a) If \( 1 \leq i \), the \( i \)-th iteration does not return “non-unifiable”.
(b) For every unifier \( \sigma' \) of \( L \) there is a substitution \( \delta_i \) such that
\[ \sigma' = \sigma_i \delta_i. \]
By (a) the algorithm exits the loop normally after \( n \) iterations.
By (b) it returns a most general unifier.
Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on $i$:

**Basis** ($i = 0$): For (a) there is nothing to prove. For (b) take $\delta_0 = \sigma'$.

**Step** ($i \Rightarrow i + 1$)

For (a), since $|L \sigma_i| > 1$ and $L \sigma_i$ unifiable, $x$ and $t$ exist and $x$ does not occur in $t$, and so “non-unifiable” is not returned.

For (b): $\delta_i$ must be of the form $[t_1/x_1, \ldots, t_k/x_k, u/x]$, $x_1, \ldots, x_k, x$ distinct. Define $\delta_{i+1} = [t_1/x_1, \ldots, t_k/x_k]$. Note $u = x \delta_i = t \delta_i = t \delta_{i+1}$ ($\sigma_i \delta_i$ is unifier (IH), $x$ not in $t$)

\[
\begin{align*}
\sigma_{i+1} \delta_{i+1} \\
= \sigma_i [t/x] \delta_{i+1} & \quad \text{(algorithm extends $\sigma_i$ with $[t/x]$)} \\
= \sigma_i [t_1/x_1, \ldots, t_k/x_k, t \delta_{i+1}/x] & \quad \text{(Note $u = t \delta_{i+1}$)} \\
= \sigma_i [t_1/x_1, \ldots, t_k/x_k, u/x] & \quad \text{(definition of $\delta_{i+1}$)} \\
= \sigma_i \delta_i & \quad \text{(IH)} \\
= \sigma' & \quad \text{(IH)}
\end{align*}
\]
The standard view of unification

A unification problem is a pair of terms $s = ? t$
(or a set of pairs $\{s_1 = ? t_1, \ldots, s^n = ? t_n\}$)

A unifier is a substitution $\sigma$ such that $s\sigma = t\sigma$
(or $s_1\sigma = t_1\sigma, \ldots, s_n\sigma = t_n\sigma$)
Renaming

Definition
A substitution $\rho$ is a renaming if for every variable $x$, $x\rho$ is a variable and $\rho$ is injective on $\text{dom}(\rho)$.
Resolvents for first-order logic

A clause $R$ is a resolvent of two clauses $C_1$ and $C_2$ if the following holds:

- There is a renaming $\rho$ such that no variable occurs in both $C_1$ and $C_2\rho$ and $\rho$ is injective on the set of variables in $C_2$

- There are literals $L_1, \ldots, L_m$ in $C_1$ ($m \geq 1$) and literals $L'_1, \ldots, L'_n$ in $C_2\rho$ ($n \geq 1$) such that

  $$L = \{ \overline{L_1}, \ldots, \overline{L_m}, L'_1, \ldots, L'_n \}$$

  is unifiable. Let $\sigma$ be an mgu of $L$.

- $R = ((C_1 - \{L_1, \ldots, L_m\}) \cup (C_2 \rho - \{L'_1, \ldots, L'_n\}))\sigma$

Example

$C_1 = \{ P(x), Q(x), P(g(y)) \}$ and $C_2 = \{ \neg P(x), R(f(x), a) \}$
Exercise

How many resolvents are there?

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$C_2$</th>
<th>Resolvents</th>
</tr>
</thead>
<tbody>
<tr>
<td>${P(x), Q(x, y)}$</td>
<td>${\neg P(f(x))}$</td>
<td></td>
</tr>
<tr>
<td>${Q(g(x)), R(f(x))}$</td>
<td>${\neg Q(f(x))}$</td>
<td></td>
</tr>
<tr>
<td>${P(x), P(f(x))}$</td>
<td>${\neg P(y), Q(y, z)}$</td>
<td></td>
</tr>
</tbody>
</table>
Why renaming?

Example

\( \forall x (P(x) \land \neg P(f(x))) \)
Resolution for first-order logic

As for propositional logic, $F \vdash_{Res} C$ means that clause $C$ can be derived from a set of clauses $F$ by a sequence of resolution steps, i.e. that there is a sequence of clauses $C_1, \ldots, C_m = C$ such that for every $C_i$

- either $C_i \in F$
- or $C_i$ is the resolvent of $C_a$ and $C_b$ where $a, b < i$.

Questions:

**Correctness** Does $F \vdash_{Res} \Box$ imply that $F$ is unsatisfiable?

**Completeness** Does unsatisfiability of $F$ imply $F \vdash_{Res} \Box$?
Exercise

Derive $\square$ from the following clauses:

1. $\{\neg P(x), Q(x), R(x, f(x))\}$
2. $\{\neg P(x), Q(x), S(f(x))\}$
3. $\{T(a)\}$
4. $\{P(a)\}$
5. $\{\neg R(a, z), T(z)\}$
6. $\{\neg T(x), \neg Q(x)\}$
7. $\{\neg T(y), \neg S(y)\}$
Definition
The universal closure of a formula $H$ with free variables $x_1, \ldots, x_n$:
\[ \forall H = \forall x_1 \forall x_2 \ldots \forall x_n H \]

Theorem
Let $F$ be a closed formula in Skolem form with matrix $F^*$ in CNF. If $F^* \vdash_{\text{Res}} \Box$ then $F$ is unsatisfiable.
Completeness: The idea

Simulate ground resolution because that is complete

Lift the resolution proof from the ground resolution proof
Lifting Lemma

Let \( C_1, C_2 \) be two clauses and let \( C'_1, C'_2 \) be two ground instances with (propositional) resolvent \( R' \). Then there is a resolvent \( R \) of \( C_1, C_2 \) such that \( R' \) is a ground instance of \( R \).
Lifting Lemma: example

\[ \{ \neg P(f(x)), Q(x) \} \]
\[ \overset{[g(a)/x]}{\downarrow} \]
\[ \{ \neg P(f(g(a))), Q(g(a)) \} \quad \{ P(f(g(y))) \} \]
\[ \overset{[a/y]}{\downarrow} \]
\[ \{ Q(g(y)) \} \quad \{ P(f(g(a))) \} \]
\[ \overset{[a/y]}{\downarrow} \]
\[ \{ Q(g(a)) \} \]
\[ \{ Q(g(a)) \} \]
Completeness of Resolution for First-Order Logic

Theorem
Let $F$ be a closed formula in Skolem form with matrix $F^*$ in CNF. If $F$ is unsatisfiable then $F^* \vdash_{\text{Res}} \Box$.

Proof
If $F$ is unsatisfiable, there is a ground resolution proof $C'_1, \ldots, C'_n = \Box$. We transform this step by step into a resolution proof $C_1, \ldots, C_n = \Box$ such that $C'_i$ is a ground instance of $C_i$.

If $C'_i$ is a ground instance of some clause $C \in F^*$:
Set $C_i = C$

If $C'_i$ is a resolvent of $C'_a, C'_b$ ($a, b < i$):
$C'_a, C'_b$ have been transformed already into $C_a, C_b$ s.t. $C'_a, C'_b$ are ground instances of $C_a, C_b$. By the Lifting Lemma there is a resolvent $R$ of $C_a, C_b$ s.t. $C'_i$ is a ground instance of $R$.
Set $C_i = R$. 
Resolution Theorem for First-Order Logic

**Theorem**

Let $F$ be a closed formula in Skolem form with matrix $F^*$ in CNF. Then $F$ is unsatisfiable iff $F^* \vdash_{\text{Res} \Box}$. 
A resolution algorithm

Input: A closed formula \( F \) in Skolem form with matrix \( S \) in CNF, i.e. \( S \) is a finite set of clauses

while \( \square \notin S \) and
  there are clauses \( C_a, C_b \in S \) and resolvent \( R \) of \( C_a \) and \( C_b \)
  such that \( R \notin S \) (modulo renaming)
do \( S := S \cup \{ R \} \)

The selection of resolvents must be *fair*:
  *every resolvent is added eventually*

Three possible behaviours:

- The algorithm terminates and \( \square \in S \)
  \( \Rightarrow F \) is unsatisfiable
- The algorithm terminates and \( \square \notin S \)
  \( \Rightarrow F \) is satisfiable
- The algorithm does not terminate
  \((\Rightarrow F \) is satisfiable\( ))\)
Refinements of resolution

Problems of resolution:

- Branching degree of the search space too large
- Too many dead ends
- Combinatorial explosion of the search space

Solution:
Strategies and heuristics: forbid certain resolution steps, which narrows the search space.

But: Completeness must be preserved!