Resolution for predicate logic

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

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

Resolution step:

\[ \{L_1, \ldots, L_n, A\} \quad \vdash \quad \{L_1', \ldots, L_m', \neg A\} \]

\[ \{L_1, \ldots, L_n, L_1', \ldots, L_m'\} \]

Mini-example:

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

\[ \{B\} \]

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 \( E(F) \).

\[
\begin{align*}
n &:= 0; \\
\text{repeat} \quad n &:= n + 1; \\
\text{until} \quad (F_1 \land F_2 \land \ldots \land F_n) \text{ is unsatisfiable;} \\
& \quad \text{(this can be checked with any calculus for propositional logic)} \\
\text{report} \quad \text{“unsatisfiable” and halt}
\end{align*}
\]

“Any calculus” \( \rightsquigarrow \) use resolution for the unsatisfiability test
**Recall: Definition of** \( Res \)

**Definition:** Let \( F \) be a set of clauses. The set of clauses \( Res(F) \) is defined by

\[
Res(F) = F \cup \{R \mid R \text{ is a resolvent of two clauses } F\}.
\]

We set:

\[
\begin{align*}
Res^0(F) &= F \\
Res^{n+1}(F) &= Res(Res^n(F)) \quad \text{für } n \geq 0
\end{align*}
\]

and define

\[
Res^*(F) = \bigcup_{n \geq 0} Res^n(F).
\]
A ground term is a term without occurrences of variables.

A ground formula is a formula in which only ground terms occur.

A predicate clause is a disjunction of atomic formulas.

A ground clause is a disjunction of ground atomic formulas.

A ground instance of a predicate clause $\mathcal{K}$ is the result of substituting ground terms for the variables of $\mathcal{K}$. 
Let \( F = \forall y_1 \forall y_2 \ldots \forall y_n F^* \) be a closed formula in Skolem form with matrix \( F^* \) in clause form, and let \( K_1, \ldots, K_m \) be the set of predicate clauses of \( F^* \).

The clause Herbrand expansion of \( F \) is the set of ground clauses

\[
CE(F) = \bigcup_{i=1}^{m} \{ K_i[y_1/t_1][y_2/t_2] \ldots [y_n/t_n] \mid t_1, t_2, \ldots, t_n \in D(F) \}
\]

**Lemma:** \( CE(F) \) is unsatisfiable iff \( E(F) \) is unsatisfiable.

**Proof:** Follows immediately from the definition of satisfiability for sets of formulas.
Let $C_1, C_2, C_3, \ldots$ be an enumeration of $CE(F)$.

\[
\begin{align*}
n &:= 0; \\
S &:= \emptyset; \\
\text{repeat} \\
&\quad n := n + 1; \\
&\quad S := S \cup \{C_n\}; \\
&\quad S := \text{Res}^*(S) \\
\text{until } \Box \in S \\
\text{report “unsatisfiable” and halt}
\end{align*}
\]
Ground Resolution Theorem: A formula $F = \forall y_1 \ldots \forall y_n F^*$ with matrix $F^*$ in clause form is unsatisfiable iff there is a set of ground clauses $C_1, \ldots, C_m$ such that:

- $C_m$ is the empty clause, and
- for every $i = 1, \ldots, m$
  - either $C_i$ is a ground instance of a clause $K \in F^*$, i.e., $C_i = K[y_1/t_1] \ldots [y_n/t_n]$ where $t_j \in D(F)$,
  - or $C_i$ is a resolvent of two clauses $C_a, C_b$ with $a < i$ and $b < i$

Proof sketch: If $F$ is unsatisfiable, then $C_1, \ldots, C_m$ can be easily extracted from $S$ by leaving clauses out.
Substitutions

A substitution $\textit{sub}$ is a (partial) mapping of variables to terms. An atomic substitution is a substitution which maps one single variable to a term.

$F_{\textit{sub}}$ denotes the result of applying the substitution $\textit{sub}$ to the formula $F$.

$t_{\textit{sub}}$ denotes the result of applying the substitution $\textit{sub}$ to the term $t$.
The concatenation $sub_1 sub_2$ of two substitutions $sub_1$ and $sub_2$ is the substitution that maps every variable $x$ to $sub_2(sub_1(x))$. (First apply $sub_1$ and then $sub_2$.)
Two substitutions $sub_1, sub_2$ are **equivalent** if $t_{sub_1} = t_{sub_2}$ for every term $t$.

Every substitution is equivalent to a concatenation of atomic substitutions. For instance, the substitution

$$x \mapsto f(h(w)) \quad y \mapsto g(a, h(w)) \quad z \mapsto h(w)$$

is equal to the concatenation

$$[x/f(z)] [y/g(a, z)] [z/h(w)].$$
Swapping substitutions

Rule for swapping substitutions:

\[ [x/t]_{\text{sub}} = \text{sub}[x/t \text{ sub}] \text{ if } x \text{ does not occur in } \text{sub}. \]

Examples:

- \[ [x/f(y)] [y/g(z)] = [y/g(z)][x/f(g(z))] \]
  - \[ \text{sub} \]

- \[ [x/f(y)] [x/g(z)] \neq [x/g(z)][x/f(y)] \]
  - \[ \text{sub} \]

- \[ [x/z] [y/x] \neq [y/x][x/z] \]
  - \[ \text{sub} \]
Let $L = \{L_1, \ldots, L_k\}$ be a set of literals of predicate clauses (terms). A substitution $\text{sub}$ is a unifier of $L$ if

$$L_1\text{sub} = L_2\text{sub} = \ldots = L_k\text{sub}$$

i.e., if $|L_{\text{sub}}| = 1$, where $L_{\text{sub}} = \{L_1\text{sub}, \ldots, L_k\text{sub}\}$.

A unifier $\text{sub}$ of $L$ is a most general unifier of $L$ if for every unifier $\text{sub}'$ of $L$ there is a substitution $s$ such that $\text{sub}' = \text{sub s}$. 
## Exercise

<table>
<thead>
<tr>
<th><strong>Unifiable?</strong></th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(f(x))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(g(y))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(x)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(f(y))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(x, f(y))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(f(u), z)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(x, f(y))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(f(u), f(z))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(x, f(x))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(f(y), y)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(x, g(x), g^2(x))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(f(z), w, g(w))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(x, f(y))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(g(y), f(a))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P(g(a), z)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Unification algorithm

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

$\text{sub} := []$ (the empty substitution)

\textbf{while} $|\text{L}_{\text{sub}}| > 1$ \textbf{do}

Find the first position at which two literals $L_1, L_2 \in \text{L}_{\text{sub}}$ differ

\textbf{if} none of the two characters at that position is a variable \textbf{then}

\textbf{then} report “non-unifiable” and \textbf{halt}

\textbf{else} let $x$ be the variable and $t$ the term starting at that position (possibly another variable)

\textbf{if} $x$ occurs in $t$

\textbf{then} report “non-unifiable” and \textbf{halt}

\textbf{else} $\text{sub} := \text{sub} [x/t]$

\text{report} “unifiable” and \textbf{return} $\text{sub}$
Lemma: The unification algorithm terminates.

Proof: Every execution of the while-loop (but the last) substitutes a variable \( x \) by a term \( t \) not containing \( x \), and so the number of variables occurring in \( L_{sub} \) decreases by one.

Lemma: If \( L \) is non-unifiable then the algorithm reports “non-unifiable”.

Proof: If \( L \) is non-unifiable then the algorithm can never exit the loop.
Correctness of the unification algorithm

**Lemma:** If $L$ is unifiable then the algorithm reports “unifiable” and returns the most general unifier of $L$ (and so in particular every unifiable set $L$ has a most general unifier).

**Proof:** Assume $L$ is unifiable and let $m$ be the number of iterations of the loop on input $L$.

Let $sub_0 = []$, for $1 \leq i \leq m$ let $sub_i$ be the value of $sub$ after the $i$-th iteration of the loop.

We prove for every $0 \leq i \geq m$:

(a) If $1 \leq i \leq m$ the $i$-th iteration does not report “non-unifiable”.

(b) For every (w.l.o.g. ground) unifier $sub'$ of $L$ there is a substitution $s_i$ such that $sub' = sub_i s_i$.

By (a) the algorithm exits the loop normally after $m$ iterations and reports “unifiable”. By (b) it returns a most general unifier.
Correctness of the unification algorithm

Proof by induction on $i$:

**Basis ($i = 0$).** For (a) there is nothing to prove. For (b) take $s_0 = \text{sub}'$.

**Step ($i > 0$).** By induction hypothesis there is $s_{i-1}$ such that $\text{sub}_{i-1}\!s_{i-1}$ is ground unifier.

For (a), since $|L_{\text{sub}_{i-1}}| > 1$ and $L_{\text{sub}_{i-1}}$ unifiable, $x$ and $t$ exist and $x$ does not occur in $t$, and so no “non-unifiable” is reported.

For (b), get $s_i$ by removing from $s_{i-1}$ every atomic substitution of the form $[x/t]$. Further, since $\text{sub}_{i-1}\!s_{i-1}$ ground unifier we can assume w.l.o.g. that $x$ does not occur in $s_i$. We have:
Correctness of the unification algorithm

\[ L \text{sub}_i s_i \]

\[ = L \text{sub}_{i-1} [x/t] s_i \]  \quad \text{(algorithm extends \text{sub}_{i-1} with \([x/t]\))}

\[ = L \text{sub}_{i-1} s_i [x/t s_i] \]  \quad \text{\((x \text{ does not occur in } s_i)\)}

\[ = L \text{sub}_{i-1} s_i [x/t s_{i-1}] \]  \quad \text{\((x \text{ does not occur in } t)\)}

\[ = L \text{sub}_{i-1} s_i [x/x s_{i-1}] \]  \quad \text{\((t s_{i-1} = x s_{i-1} \text{ because \text{sub}_{i-1} s_{i-1} unifier})\)}

\[ = L \text{sub}_{i-1} s_{i-1} \]  \quad \text{\(\text{definition of } s_i)\}

\[ = L \text{sub}' \]  \quad \text{\(\text{induction hypothesis}\)}
A clause $R$ is a resolvent of two predicate clauses $K_1, K_2$ if the following holds:

- There are renamings of variables $s_1, s_2$ (particular cases of substitutions) such that no variable occurs in both $K_1 s_1$ and $K_2 s_2$.
- There are literals $L_1, \ldots, L_m$ in $K_1 s_1$ and literals $L'_1, \ldots, L'_n$ in $K_2 s_2$ such that the set

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

is unifiable. Let $sub$ be the most general unifier of $L$.
- $R = \left( (K_1 s_1 - \{L_1, \ldots, L_m\}) \cup (K_2 s_2 - \{L'_1, \ldots, L'_n\}) \right) sub$. 
Correctness and completeness

Questions:

• If using predicate resolution $\Box$ can be derived from $F$ then $F$ is unsatisfiable (correctness)

• If $F$ is unsatisfiable then predicate resolution can derive the empty clause $\Box$ from $F$ (completeness)
Exercise

Have the following pairs of predicate clauses a resolvent? 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>
Lifting-Lemma

Let $C_1, C_2$ be predicate clauses and let $C'_1, C'_2$ be two ground instances of them that can be resolved into the resolvent $R'$. Then there is predicate resolvent $R$ of $C_1, C_2$ such that $R'$ is a ground instance of $R$.

---

Diagram:

- $K_1 \downarrow K'_1 \rightarrow R' \rightarrow K'_2 \downarrow K_2$
- $\rightarrow$: Resolution
- $\rightarrow$: Substitution
Lifting-Lemma: example

\[
\begin{align*}
\{ \neg P(f(x)), Q(x) \} & \\
\downarrow[x/g(a)] & \\
\{ \neg P(f(g(a))), Q(g(a)) \} & \quad \{ P(f(g(y))) \} \\
\downarrow[y/a] & \\
\{ Q(g(y)) \} & \quad \{ P(f(g(a))) \} \\
\downarrow[y/a] & \\
\{ Q(g(a)) \} & \quad \{ P(f(g(a))) \}
\end{align*}
\]
Resolution Theorem of Predicate Logic:

Let $F$ be a closed formula in Skolem form with matrix $F^*$ in predicate clause form. $F$ is unsatisfiable iff $\square \in Res^*(F^*)$. 
Universal closure

The universal closure of a formula $H$ with free variables $x_1, \ldots, x_n$ is the formula

$$
\forall H = \forall x_1 \forall x_2 \ldots \forall x_n H
$$

Let $F$ be a closed formula in Skolem form with matrix $F^*$. Then

$$
F \equiv \forall F^* \equiv \bigwedge_{K \in F^*} \forall K
$$

Example:

$$
F^* = P(x, y) \land \neg Q(y, x)
$$

$$
F \equiv \forall x \forall y (P(x, y) \land \neg Q(y, x)) \equiv \forall x \forall y P(x, y) \land \forall x \forall y (\neg Q(y, x))
$$
Exercise

Is the set of clauses

\[
\{\{P(f(x))\}, \{\neg P(f(x)), Q(f(x), x)\}, \{\neg Q(f(a), f(f(a)))\}, \\
\{\neg P(x), Q(x, f(x))\}\}
\]

unsatisfiable?
We consider the following set of predicate clauses (Schöning):

\[
F = \{ \{\neg P(x), Q(x), R(x, f(x))\}, \{\neg P(x), Q(x), S(f(x))\}, \{T(a)\},
\{P(a)\}, \{\neg R(a, x), T(x)\}, \{\neg T(x), \neg Q(x)\}, \{\neg T(x), \neg S(x)\}\}
\]

and prove it is unsatisfiable with otter.
Problems of predicate 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!