Definite Logic Programs: Derivation and Proof Trees

CSE 505 – Computing with Logic
Stony Brook University

http://www.cs.stonybrook.edu/~cse505
Refutation in Predicate Logic

parent(pam, bob). parent(tom, bob).
parent(tom, liz). ...
anc(X,Y) :- parent(X,Y).
anc(X,Y) :- parent(X,Z), anc(Z,Y).

• **Goal G:** For what values of \( Q \) is \( \neg \text{anc(tom,Q)} \) a logical consequence of the above program?

• **Negate the goal G:** i.e. \( \neg G \equiv \forall Q \neg \text{anc(tom, Q)} \).

• Consider the clauses in the program \( P \cup \neg G \) and apply refutation

  • Note that a program clause written as \( p(A,B) :- q(A,C), r(B,C) \)
  can be rewritten as: \( \forall A, B, C \ (p(A, B) \lor \neg q(A, C) \lor \neg r(B, C)) \)
  i.e., l.h.s. literal is positive, while all r.h.s. literals are negative

  • Note also that all variables are universally quantified in a clause!

• Note on syntax: we use \( :- \), \? and \( \leftarrow \) for IMPLICATION
Refutation: An Example

parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).

\[ \text{anc}(X, Y) \leftarrow \text{parent}(X, Y) \]
\[ \text{anc}(X, Y) \leftarrow \text{parent}(X, Z), \text{anc}(Z, Y) \]

\[ \text{anc}(X, Y) \leftarrow \text{parent}(X, Q) \]
\[ \text{parent}(tom, Q) \]
\[ \text{Q=bob} \]
Refutation: An Example

\texttt{parent}(pam, bob).
\texttt{parent}(tom, bob).
\texttt{parent}(tom, liz).
\texttt{parent}(bob, ann).
\texttt{parent}(bob, pat).
\texttt{parent}(pat, jim).

\texttt{anc}(X,Y) :-
    \texttt{parent}(X,Y).

\texttt{anc}(X,Y) :-
    \texttt{parent}(X,Z),
    \texttt{anc}(Z,Y).
Unification

- Operation done to “match” the goal atom with the head of a clause in the program.
- Forms the basis for the matching operation we used for Prolog evaluation:
  - \( f(a, Y) \) and \( f(X, b) \) unify when \( X = a \) and \( Y = b \)
  - \( f(a, X) \) and \( f(X, b) \) do not unify

That is, the query \(?- f(a, X) = f(X, b).\) fails in Prolog
Substitutions

- A substitution is a mapping between variables and values (terms).
- Denoted by $\{x_1/t_1, x_2/t_2, \ldots, x_n/t_n\}$ such that
  - $x_i \neq t_i$, and
  - $x_i$ and $x_j$ are distinct variables when $i \neq j$.
- The empty substitution is denoted by $\{\}$ (or $\varepsilon$).
- A substitution is said to be a renaming if it is of the form $\{x_1/y_1, x_2/y_2, \ldots, x_n/y_n\}$ and $y_1, y_2, \ldots, y_n$ is a permutation of $x_1, x_2, \ldots, x_n$.
- Example: $\{x/y, y/x\}$ is a renaming substitution.
Substitutions and Terms

• Application of a substitution:
  • $x\theta = t$ if $x/t \in \theta$.
  • $x\theta = x$ if $x/t \not\in \theta$ for any term $t$.

• Application of a substitution $\{x_1/t_1, \ldots, x_n/t_n\}$ to a term/formula $F$:
  • is a term/formula obtained by simultaneously replacing every free occurrence of $x_i$ in $F$ by $t_i$.
  • Denoted by $F\theta$ [and $F\theta$ is said to be an instance of $F$]

• Example:

$$p(f(X,Z), f(Y,a))\{X/g(Y), Y/Z, Z/a\} = p(f(g(Y),a), f(Z,a))$$
Composition of Substitutions

- **Composition** \( \theta \sigma \) of substitutions \( \theta = \{x_1/s_1, \ldots, x_m/s_m\} \) and \( \sigma = \{y_1/t_1, \ldots, y_n/t_n\} \):

  1. First form the set \( \{x_1/s_1\sigma, \ldots, x_m/s_m\sigma, \ y_1/t_1, \ldots, y_n/t_n\} \)
  2. Remove from the set \( x_i/s_i\sigma \) if \( s_i\sigma = x_i \)
  3. Remove from the set \( y_j/t_j \) if \( y_j \) is identical to some variable \( x_i \)

  - Example: Let \( \theta = \sigma = \{x/g(y), y/z, z/a\} \). Then \( \theta\sigma = \{x/g(y), y/z, z/a\}\{x/g(y), y/z, z/a\} = \{x/g(z), y/a, z/a\} \)

  - More examples: Let \( \theta = \{x/f(y)\} \) and \( \sigma = \{y/a\} \)
    - \( \theta\sigma = \{x/f(a), y/a\} \)
    - \( \sigma\theta = \{y/a, x/f(y)\} \)

  - Composition is not commutative but is associative: \( \theta(\sigma\gamma) = (\theta\sigma)\gamma \)
Idempotence

• A substitution $\theta$ is idempotent iff $\theta \theta = \theta$.

• Examples:
  • $\{X/g(Y), Y/Z, Z/a\}$ is not idempotent since  
    $\{X/g(Y), Y/Z, Z/a\} \{X/g(Y), Y/Z, Z/a\} = \{X/g(Z), Y/a, Z/a\}$
  • $\{X/g(Z), Y/a, Z/a\}$ is not idempotent either since  
    $\{X/g(Z), Y/a, Z/a\} \{X/g(Z), Y/a, Z/a\} = \{X/g(a), Y/a, Z/a\}$
  • $\{X/g(a), Y/a, Z/a\}$ is idempotent

• For a substitution $\theta = \{x_1/t_1, x_2/t_2, \ldots, x_n/t_n\}$,
  • $\text{Dom}(\theta) = \{x_1, x_2, \ldots, x_n\}$
  • $\text{Range}(\theta) = \text{set of all variables in } t_1, t_2, \ldots, t_n$
• A substitution $\theta$ is idempotent iff $\text{Dom}(\theta) \cap \text{Range}(\theta) = \emptyset$
Unification

- **Unification** is a procedure that takes two atomic formulas as input, and either shows how they can be instantiated to identical atoms or, reports a failure.

- For example:
  
  \[- f(X, g(Y)) = f(a, g(X)) \, . \]

- Any solution of the equations: \( \{ X=a , \ g(Y)=g(X) \} \) must clearly be a solution of equation above

- Similarly, any solution of: \( \{ X = a , \ Y = X \} \) must be a solution of equations \( \{ X = a , \ g(Y) = g(X) \} \)

- Finally any solution of: \( \{ X = a , \ Y = a \} \) is a solution of \( \{ X = a , \ Y = X \} \)
Unifiers

- A substitution $\theta$ is a **unifier of** two terms $s$ and $t$ if $s\theta$ is identical to $t\theta$
- $\theta$ is a unifier of a set of equations $\{s_1=t_1, \ldots, s_n=t_n\}$, if for all $i, s_i\theta = t_i\theta$
- A substitution $\theta$ is *more general* than $\sigma$ (written as $\theta \geq \sigma$) if there is a substitution $\omega$ such that $\sigma = \theta\omega$
- A substitution $\theta$ is a **most general unifier** (**mgu**) of two terms (or a set of equations) if for every unifier $\sigma$ of the two terms (or equations) $\theta \geq \sigma$

**Example:** Consider two terms $f(g(X), Y, a)$ and $f(Z, W, X)$.

$\theta_1 = \{X/a, Y/b, Z/g(a), W/b\}$ is a unifier

$\theta_2 = \{X/a, Y/W, Z/g(a)\}$ is also a unifier

$\theta_2$ is more general than $\theta_1$ because $\theta_1 = \theta_2\omega$ where $\omega = \{W/b\}$

$\theta_2$ is also the most general unifier of the 2 terms
Equations and Unifiers

• A set of equations $E$ is in **solved form** if it is of the form 
  \[ \{x_1 = t_1, \ldots, x_n = t_n \} \text{ iff no } x_i \text{ appears in any } t_j. \]

• Given a set of equations $E = \{x_1 = t_1, \ldots, x_n = t_n\}$, the substitution 
  \[ \{x_1 / t_1, \ldots, x_n / t_n\} \] is an idempotent mgu of $E$.

• Two sets of equations $E_1$ and $E_2$ are said to be **equivalent** iff they have the same set of unifiers.

• To find the mgu of two terms $s$ and $t$, try to find a set of equations in solved form that is equivalent to \( \{s = t\}. \) If there is no equivalent solved form, there is no mgu.
A Simple Unification Algorithm

Given a set of equations \( E \):

repeat

select \( s = t \in E \);

Case \( s = t \) of

1. \( f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \):
   replace the equation by \( s_i = t_i \) for all \( i \)

2. \( f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m) \), \( f \neq g \) or \( n \neq m \):
   halt with failure

3. \( X = X \) : remove the equation

4. \( t = X \) : where \( t \) is not a variable, \( X \) is a variable
   replace equation by \( X = t \)

5. \( X = t \) : where \( X \neq t \) and \( X \) occurs more than once in \( E \):
   if \( X \) is a proper subterm of \( t \)
   then halt with failure  \( (5a) \)
   else replace all other \( X \) in \( E \) by \( t \)  \( (5b) \)

until no action is possible for any equation in \( E \)

return \( E \)
Example: Find the mgu of \( f(X, g(Y)) \) and \( f(g(Z), Z) \)

\[
\begin{align*}
\{ f(X, g(Y)) &= f(g(Z), Z) \} & \Rightarrow \\
\Rightarrow & \{ X = g(Z), g(Y) = Z \} & \text{case 1} \\
\Rightarrow & \{ X = g(Z), Z = g(Y) \} & \text{case 4} \\
\Rightarrow & \{ X = g(g(Y)), Z = g(Y) \} & \text{case 5b}
\end{align*}
\]
Example: Find the mgu of $f(X, g(X))$ and $f(Z, Z)$

$$\{ f(X, g(X)) = f(Z, Z) \} \Rightarrow$$

$$\Rightarrow \{ X = Z, g(X) = Z \} \quad \text{case 1}$$

$$\Rightarrow \{ X = Z, g(Z) = Z \} \quad \text{case 5b}$$

$$\Rightarrow \{ X = Z, Z = g(Z) \} \quad \text{case 4}$$

$$\Rightarrow \text{fail} \quad \text{case 5a}$$
Example: Find the mgu of $f(X, g(X), b)$ and $f(a, g(Z), Z)$

\[
\{ f(X, g(X), b) = f(a, g(Z), Z) \} \Rightarrow
\]

$\Rightarrow \{ X = a, g(X) = g(Z), b = Z \}$ case 1

$\Rightarrow \{ X = a, g(a) = g(Z), b = Z \}$ case 5b

$\Rightarrow \{ X = a, a = Z, b = Z \}$ case 1

$\Rightarrow \{ X = a, Z = a, b = Z \}$ case 4

$\Rightarrow \{ X = a, Z = a, b = a \}$ case 2

$\Rightarrow \text{fail}$
Complexity of the unification algorithm

- Consider the set of equations:
  \[ E = \{ g(X_1, \ldots, X_n) = g(f(X_0, X_0), f(X_1, X_1), \ldots, f(X_{n-1}, X_{n-1}) \} \]

  - By applying case 1 of the algorithm, we get
    \[ \{ X_1 = f(X_0, X_0), X_2 = f(X_1, X_1), X_3 = f(X_2, X_2), \ldots, X_n = f(X_{n-1}, X_{n-1}) \} \]

  - If terms are kept as trees, the final value for \( X_n \) is a tree of size \( O(2^n) \)

  - Recall that for case 5 we need to first check if a variable appears in a term, and this could now take \( O(2^n) \) time

- \( X = t \) is the most common case for unification in Prolog
  - There are linear-time unification algorithms that share structures (terms as DAGs)
  - Therefore, the fastest algorithms are linear in \( t \)
  - **Prolog cuts corners by omitting case 5a (called occur check), thereby doing \( X = t \) in constant time**
Most General Unifiers

- Note that mgu stands for a/one most general unifier
- There may be more than one mgu
- E.g. $f(X) = f(Y)$ has two mgus:
  - $\{X / Y\}$ (by our simple algorithm)
  - $\{Y / X\}$ (by definition of mgu)
- If $\theta$ is an mgu of $s$ and $t$, and $\omega$ is a renaming, then $\theta\omega$ is a mgu of $s$ and $t$
- If $\theta$ and $\sigma$ are mgus of $s$ and $t$, then there is a renaming $\omega$ such that $\theta = \sigma\omega$
- MGU is unique up to renaming!
SLD Resolution

- **Selective Linear** **Definite clause (SLD) Resolution:**

\[
\leftarrow A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_m \quad B_0 \leftarrow B_1, \ldots, B_n
\]

\[
\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m) \theta
\]

where:

1. \(A_j\) are atomic formulas
2. \(B_0 \leftarrow B_1, \ldots, B_n\) is a *(renamed variables)* definite clause in the program
3. \(\theta = \text{mgu}(A_i, B_0)\)
   - \(A_i\) is called the *selected* atom
   - Given a goal \(\leftarrow A_1, \ldots, A_n\) a function called the *selection function* or *computation rule* selects \(A_i\)
SLD Resolution (cont.)

- When the resolution rule is applied, from a goal $G$ and a clause $C$, we get a new goal $G'$.
- We then say that $G'$ is derived directly from $G$ and $C$:

$$G \overset{C}{\Rightarrow} G'$$

- An *SLD Derivation* is a sequence:

$$G_0 \overset{C_0}{\Rightarrow} G_1 \ldots G_i \overset{C_i}{\Rightarrow} G_{i+1} \ldots$$
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).

anc(X,Y) :-
    parent(X,Y).
anc(X,Y) :-
    parent(X,Z),
    anc(Z,Y).

← anc(tom, Q)

← parent(tom, Q)

anc(tom, Q)
⇒ parent(tom, Q)
⇒ □
SLD Derivation

\begin{align*}
\text{parent}(pam, bob). \\
\text{parent}(tom, bob). \\
\text{parent}(tom, liz). \\
\text{parent}(bob, ann). \\
\text{parent}(bob, pat). \\
\text{parent}(pat, jim). \\
\text{anc}(X, Y) :- \\
\quad \text{parent}(X, Y). \\
\text{anc}(X, Y) :- \\
\quad \text{parent}(X, Z), \text{anc}(Z, Y). \\
\end{align*}

\begin{align*}
\text{\textbf{\textLeftarrow anc}(tom, Q)} \\
\quad \text{anc}(X, Y) \\
\quad \quad \textbf{\textLeftarrow parent}(X, Z), \text{anc}(Z, Y). \\
\quad \text{\textbf{\textLeftarrow parent}(tom, Z'), \text{anc}(Z', Q)} \\
\quad \text{\quad parent}(tom, bob) \leftarrow \\
\quad \text{\textbf{\textLeftarrow anc}(bob, Q)} \\
\quad \quad \text{\quad anc}(X, Y) \\
\quad \quad \quad \textbf{\textLeftarrow parent}(X, Y). \\
\quad \quad \text{\textbf{\textLeftarrow parent}(bob, Q)} \\
\quad \quad \quad \quad \text{\quad parent}(bob, ann) \leftarrow \\
\quad \quad \text{\quad \textbf{\textLeftarrow parent}(bob, ann)} \\
\quad \quad \quad \quad \quad \text{\textbf{\textLeftarrow parent}(bob, Q)} \\
\quad \quad \quad \quad \quad \quad \textbf{\textLeftarrow parent}(bob, Q) \\
\quad \quad \quad \quad \quad \quad \quad \textbf{\textLeftarrow \Box} \\
\quad \quad \quad \quad \quad \quad \quad \text{Q=ann} \\
\text{anc}(tom, Q) \\
\quad \textbf{\textRightarrow parent}(tom, Z') \\
\quad \textbf{\textRightarrow \text{anc}(Z', Q)} \\
\quad \textbf{\textRightarrow \text{anc}(bob, Q)} \\
\quad \textbf{\textRightarrow parent}(bob, Q) \\
\quad \textbf{\textRightarrow \Box}
\end{align*}
Computed Answer Substitution

- Let $\theta_0, \theta_1, \ldots, \theta_{n-1}$ be the sequence of mgus used in derivation

\[
G_0 \xrightarrow{C_0} G_1 \cdots G_{n-1} \xrightarrow{C_{n-1}} G_n
\]

Then $\theta = \theta_0 \theta_1 \cdots \theta_{n-1}$ is the \textit{computed substitution} of the derivation

- Example derivation \textit{in tabled form}:

<table>
<thead>
<tr>
<th>Goal</th>
<th>Clause Used</th>
<th>mgu</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{anc}$(tom, Q)</td>
<td>$\text{anc}(X',Y') : -$</td>
<td>$\theta_0 = {X'/\text{tom}, Y'/Q}$</td>
</tr>
<tr>
<td></td>
<td>$\text{parent}(X',Z'), \text{anc}(Z',Y')$</td>
<td></td>
</tr>
<tr>
<td>$\text{parent}$(tom, Z')</td>
<td>$\text{parent}(\text{tom}, \text{bob})$.</td>
<td>$\theta_1 = {Z'/\text{bob}}$</td>
</tr>
<tr>
<td></td>
<td>$\text{anc}(X'',Y'') : -$</td>
<td>$\theta_2 = {X''/\text{bob}, Y''/Q}$</td>
</tr>
<tr>
<td>$\text{anc}$(bob, Q)</td>
<td>$\text{parent}(\text{bob}, \text{ann})$.</td>
<td>$\theta_3 = {Q/\text{ann}}$</td>
</tr>
<tr>
<td>$\text{parent}$(bob, Q)</td>
<td>$\text{parent}(\text{bob}, \text{ann})$.</td>
<td></td>
</tr>
</tbody>
</table>

- Computed substitution for the above derivation is

$\theta_0 \theta_1 \theta_2 \theta_3 = \{X'/\text{tom}, Y'/\text{ann}, Z'/\text{bob}, X''/\text{bob}, Y''/\text{ann}, Q/\text{ann}\}$
Computed Answer Substitution

- A finite derivation of the form
  \[ G_0 \xrightarrow{C_0} G_1 \cdots G_{n-1} \xrightarrow{C_{n-1}} G_n \]
  where \( G_n = \square \) (i.e., an empty goal) is an *SLD refutation* of \( G_0 \).

- The computed substitution of an SLD refutation of \( G \), restricted to variables of \( G \), is a *computed answer substitution* for \( G \).

- Example: the previous SLD-derivation is an SLD refutation.
  - The computed answer substitution is:
    \[ \{ X'/\text{tom}, Y'/\text{ann}, Z'/\text{bob}, X''/\text{bob}, Y''/\text{ann}, Q/\text{ann} \} \]
  restricted to \( Q \) is: \( \{ Q/\text{ann} \} \)
Failed SLD Derivation

- A derivation of a goal clause \( G_0 \) whose last element is not empty, and cannot be resolved with any clause of the program.

- Example: consider the following program:

  ```prolog
  grandfather(X,Z) :- father(X,Y), parent(Y,Z).
  parent(X,Y) :- father(X,Y).
  parent(X,Y) :- mother(X,Y).
  father(a,b).
  mother(b,c).
  ```

- A failed SLD derivation of \( \text{grandfather}(a,Q) \) is:

  ```prolog
  grandfather(a,Q)
  \[\leadsto\] father(a,Y'), parent(Y',Q)
  \[\leadsto\] parent(b,Q)
  \[\leadsto\] father(b,Q)
  ```
OLD Resolution

- Prolog follows OLD resolution = SLD with **left-to-right literal selection**
- Prolog searches for OLD proofs by expanding the resolution tree depth first
- This depth-first expansion is close to how procedural programs are evaluated:
  - Consider a goal $G_1, G_2, \ldots, G_n$ as a "procedure stack" with $G_1$, the selected literal on top
  - Call $G_1$
  - **If** and **when** $G_1$ returns, continue with the rest of the computation: call $G_2$, and upon its return call $G_3$, etc. until nothing is left
  - Note: $G_2$ is "opened up" only when $G_1$ returns, not after executing only some part of $G_1$
**SLD Tree**

- A tree where every path is an SLD derivation (special case is the tree corresponding to all paths for a Prolog query)

```
grandfather(X,Z) :-
    father(X,Y), parent(Y,Z).

parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).

father(a,b).
mother(b,c).
```

Diagram:

```
    grandfather(a, Q)
       /           \
  ← father(a,Z'), parent(Z', Q)
     /     \                        /   \
← parent(b, Q) ← father(b, Q)    ← mother(b, Q)
      /                     \          
← parent(b, Q)           \        
                         /          
                  ← mother(b, Q)
```
Soundness of SLD resolution

- Let $P$ be a definite program, $R$ be a computation rule, and $\theta$ be a computed answer substitution for a goal $G$.

Then $\forall G \theta$ is a logical consequence of $P$.

- Proof is by induction on the number of resolution steps used in the refutation of $G$.

  - Base case uses the following lemma:

    - Let $F$ be a formula and $F'$ be an instance of $F$, i.e., $F' = F \theta$ for some substitution $\theta$.

Then $(\forall F) \models (\forall F')$. 