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

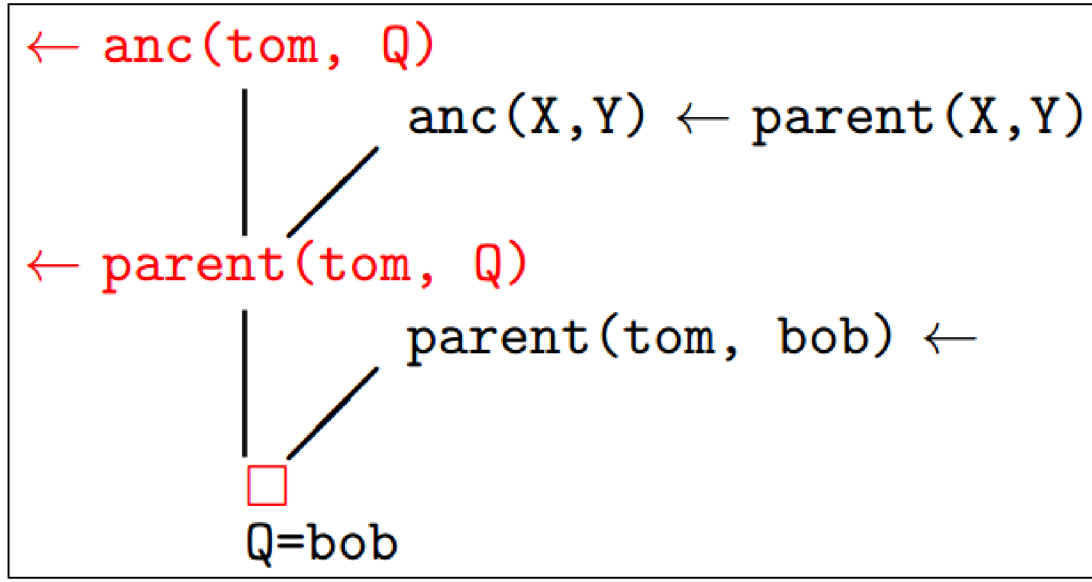
```
parent(pam, bob). parent(tom, bob).
parent(tom, liz). ...
anc(X,Y) :- parent(X,Y).
anc(X,Y) :- parent(X,Z), anc(Z,Y).
```

- **<u>Goal G:</u>** For what values of `Q` is `:- anc(tom,Q)` a logical consequence of the above program?
- **<u>Negate the goal G:</u>** i.e. ¬G ≡ ∀`Q` ¬`anc(tom, Q).`
- Consider the clauses in the program P ∪ ¬G and apply refutation
  - Note that a program clause written as `p(A,B) :- q(A,C), r(B,C)` can be rewritten as: `∀A,B,C (p(A, B) ∨ ¬q(A, C) ∨ ¬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 ← for IMPLICATION

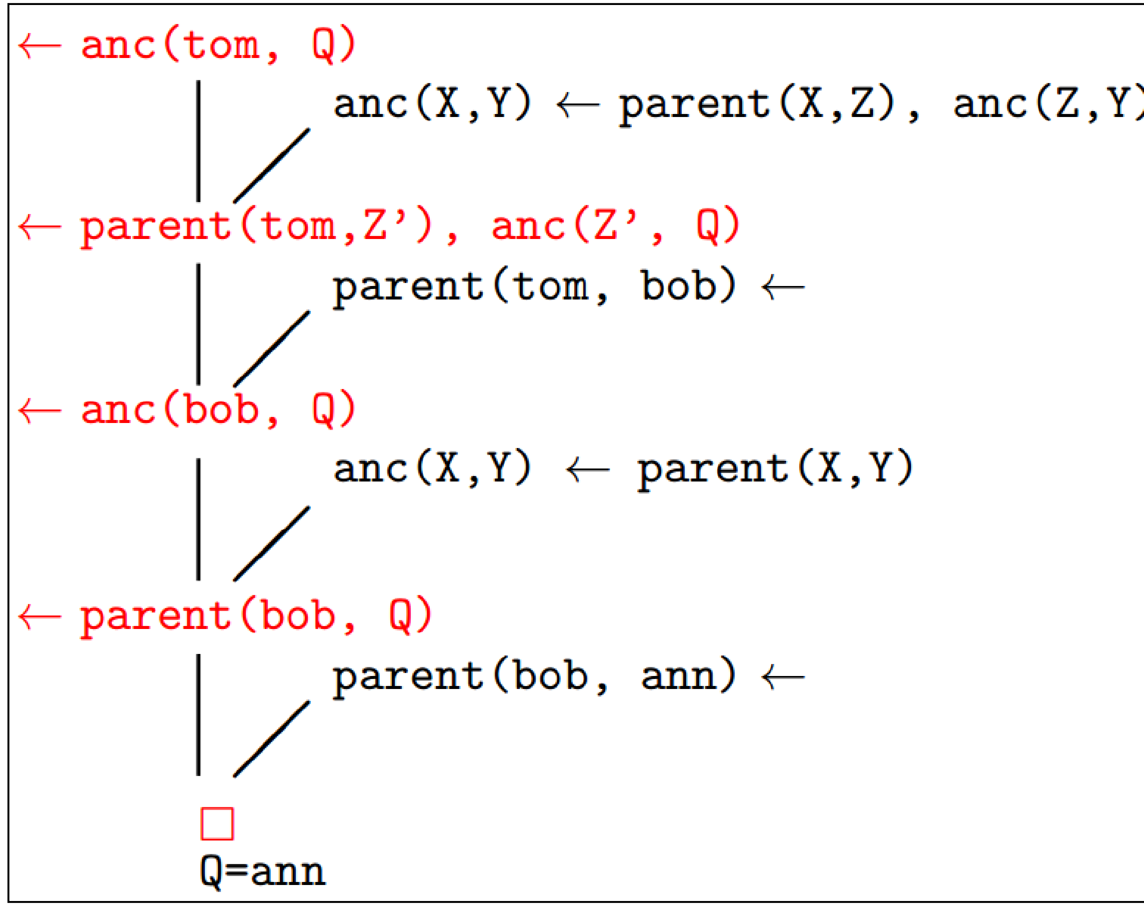
Refutation: An Example

```
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).
```



Refutation: An Example

```
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).
```



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 $\{\mathtt{X_1/t_1, X_2/t_2, ..., X_n/t_n}\}$ such that
    - $\mathtt{X_i} \neq \mathtt{t_i}$, and
    - $\mathtt{X_i}$ and $\mathtt{X_j}$ are distinct variables when $\mathtt{i} \neq \mathtt{j}$.
  - The empty substitution is denoted by $\{\}$ (or ε).
  - A substitution is said to be a ***renaming*** if it is of the form $\{\mathtt{X_1/Y_1, X_2/Y_2, ..., X_n/Y_n}\}$ and $\mathtt{Y_1,Y_2,...,Y_n}$ is a permutation of $\mathtt{X_1,X_2,...,X_n}$.
    - Example: $\{\mathtt{X/Y, Y/X}\}$ is a renaming substitution.

Substitutions and Terms

- **Application of a substitution:**
  - $X\theta = t$ if $X/t \in \theta$.
  - $X\theta = X$ if $X/t \notin \theta$ for any term t.
  - Application of a substitution $\{X_1/t_1, ..., X_n/t_n\}$ 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 Fθ [and Fθ 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 $\theta = \{\mathtt{X_1/s_1, \ldots, X_m/s_m}\}$ and $\sigma = \{\mathtt{Y_1/t_1, \ldots, Y_n/t_n}\}$:
  1. First form the set $\{\mathtt{X_1/s_1}\sigma, \ldots, \mathtt{X_m/s_m}\sigma, \mathtt{Y_1/t_1}, \ldots, \mathtt{Y_n/t_n}\}$
  2. Remove from the set $\mathtt{X_i/s_i}\sigma$ if $\mathtt{s_i}\sigma = \mathtt{X_i}$
  3. Remove from the set $\mathtt{Y_j/t_j}$ if $\mathtt{Y_j}$ is identical to some variable $\mathtt{X_i}$
  - Example: Let `θ = σ = {X/g(Y), Y/Z, Z/a}`. Then

`θσ={X/g(Y), Y/Z, Z/a}{X/g(Y), Y/Z, Z/a}=`

`{X/g(Z), Y/a, Z/a}`

- More examples: Let `θ = {X/f(Y)}` and `σ = {Y/a}`
  - `θσ = {X/f(a), Y/a}`
  - `σθ = {Y/a, X/f(Y)}`
- Composition is not commutative but is associative: `θ(σγ) = (θσ)γ`

Idempotence

- A substitution θ is ***idempotent*** iff θθ = θ.
- 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 θ = {$X_1/t_1, X_2/t_2, \ldots, X_n/t_n$},
  - Dom(θ) = {$X_1, X_2, \ldots, X_n$}
  - Range(θ) = set of all variables in $t_1, t_2, \ldots, t_n$
- A substitution θ is ***idempotent*** iff Dom(θ) ∩ Range(θ) = ∅

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 θ is a ***unifier of*** two terms s and t if sθ is identical to tθ
  - θ 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 θ is ***more general*** than σ (written as θ ≥ σ) if there is a substitution ω such that σ = θω
- A substitution θ is a ***most general unifier*** (*mgu*) of two terms (or a set of equations) if for every unifier σ of the two terms (or equations) θ≥σ
  - Example: Consider two terms `f(g(X),Y,a)` and `f(Z,W,X)`.

    θ_1 = `{X/a, Y/b, Z/g(a), W/b}` is a unifier

    θ_2 = `{X/a, Y/W , Z/g(a)}` is also a unifier

  θ_2 is more general than θ_1 because $\theta_1 = \theta_2\omega$ where ω=`{W/b}`

  θ_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

$\{X_1=t_1, \ldots, X_n=t_n\}$ iff no X_i 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 ∈ E;
  case s = t of
      1. f(s1, ...,sn) = f(t1, ...,tn):
            replace the equation by si = ti for all i
      2. f(s1, ...,sn) = g(t1, ...,tm), f ≠ g or n ≠ 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 ≠ 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
```

A Simple Unification Algorithm

Example: Find the mgu of **`f(X,g(Y))`** and **`f(g(Z),Z)`**

```
{f(X, g(Y)) = f(g(Z), Z)} ⇒
     ⇒ {X = g(Z), g(Y) = Z}            case 1
     ⇒ {X = g(Z), Z = g(Y)}            case 4
     ⇒ {X = g(g(Y)), Z = g(Y)}         case 5b
```

A Simple Unification Algorithm

Example: Find the mgu of **`f(X, g(X))`** and **`f(Z, Z)`**

```
{f(X, g(X)) = f(Z, Z)} ⇒
    ⇒ {X = Z, g(X) = Z}               case 1
    ⇒ {X = Z, g(Z) = Z}               case 5b
    ⇒ {X = Z, Z = g(Z)}               case 4
    ⇒ fail                            case 5a
```

A Simple Unification Algorithm

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)} ⇒
        ⇒ {X = a, g(X) = g(Z), b = Z} case 1
        ⇒ {X = a, g(a) = g(Z), b = Z} case 5b
        ⇒ {X = a, a = Z, b = Z}       case 1
        ⇒ {X = a, Z = a, b = Z}       case 4
        ⇒ {X = a, Z = a, b = a}       case 2
        ⇒ fail
```

Complexity of the unification algorithm

- Consider the set of equations:

$E=\{\ g(X_1,...,X_n)=g(f(X_0,X_0),f(X_1,\ X_1),\ldots,f(X_{n-1},X_{n-1})\ \}$

  - 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 θ is an mgu of **`s`** and **`t`**, and ω is a renaming, then $\theta\omega$ is a mgu of **`s`** and **`t`**
- If θ and σ are mgus of **`s`** and **`t`**, then there is a renaming ω such that $\theta = \sigma\omega$
  - MGU is unique up to renaming!

SLD Resolution

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

$$\frac{\leftarrow A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_m \quad B_0 \leftarrow B_1, \ldots, B_n}{\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m)\theta}$$

where:

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

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***:

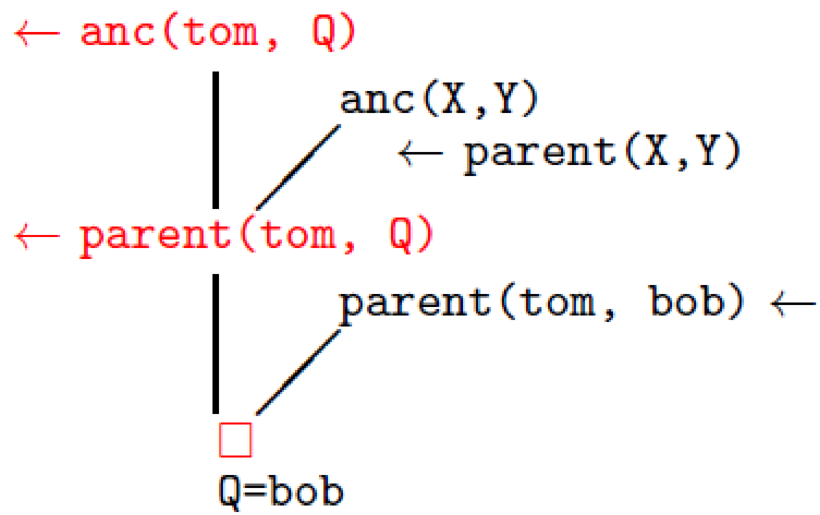
$$G \overset{C}{\leadsto} G'$$

- An ***SLD Derivation*** is a sequence:

$$G_0 \overset{C_0}{\leadsto} G_1 \cdots G_i \overset{C_i}{\leadsto} G_{i+1} \cdots$$

SLD Derivation

```
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)
⇝ □

SLD Derivation

```
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)
                    anc(X,Y)
                      ← parent(X,Z), anc(Z,Y)
← parent(tom,Z'), anc(Z', Q)
                    parent(tom, bob) ←
← anc(bob, Q)
                    anc(X,Y)
                      ← parent(X,Y)
← parent(bob, Q)
                    parent(bob, ann) ←
□
Q=ann
```

```
anc(tom, Q)
⤳ parent(tom, Z')
        anc(Z', Q)
⤳ anc(bob, Q)
⤳ parent(bob, Q)
⤳ □
```

Computed Answer Substitution

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

$$G_0 \overset{C_0}{\rightsquigarrow} G_1 \cdots G_{n-1} \overset{C_{n-1}}{\rightsquigarrow} G_n$$

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

- Example derivation **in tabled form**:

Goal	Clause Used	mgu
`anc(tom, Q)`	`anc(X',Y') :-`	
	`parent(X',Z'), anc(Z',Y')`	$\theta_0 = \{X'/\texttt{tom}, Y'/Q\}$
`parent(tom, Z'),`		
`anc(Z', Q)`	`parent(tom, bob).`	$\theta_1 = \{Z'/\texttt{bob}\}$
`anc(bob, Q)`	`anc(X'', Y'') :-`	
	`parent(X'', Y'').`	$\theta_2 = \{X''/\texttt{bob}, Y''/Q\}$
`parent(bob, Q)`	`parent(bob, ann).`	$\theta_3 = \{Q/\texttt{ann}\}$
$\square$		

- Computed substitution for the above derivation is

$\theta_0\theta_1\theta_2\theta_3$ = {`X'/tom, Y'/ann, Z'/bob, X''/bob, Y''/ann, Q/ann`}

Computed Answer Substitution

- A finite derivation of the form

$$G_0 \overset{C_0}{\rightsquigarrow} G_1 \cdots G_{n-1} \overset{C_{n-1}}{\rightsquigarrow} G_n$$

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'/tom, Y'/ann, Z'/bob, X''/bob, Y''/ann, Q/ann`}

restricted to Q is: {`Q/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:

```
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 **grandfather(a,Q)** is:

```
grandfather(a,Q)
 ⇝ father(a,Y'), parent(Y',Q)
 ⇝ parent(b,Q)
 ⇝ father(b,Q)
```

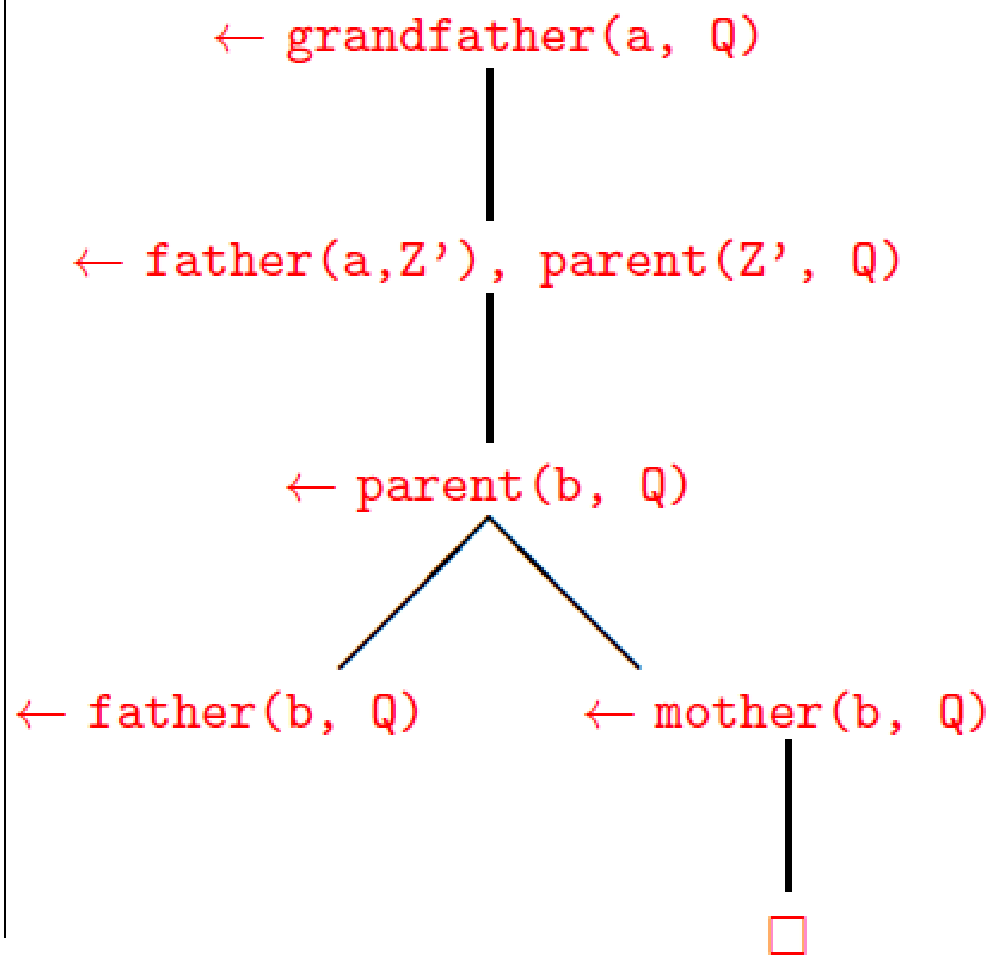
OLD Resolution

- Prolog follows *OLD resolution* = SLD with **<u>left-to-right literal selection</u>**
  - 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
    - **<u>If</u>** and **<u>when</u>** 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).
```



Soundness of SLD resolution

- Let P be a definite program, R be a computation rule, and θ 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 θ.

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