

Resolution

- Let P be a set of *clauses* $\{C_1, C_2, \dots, C_n\}$, where
 - each clause C_i is of the form $(L_1 \vee L_2 \vee \dots \vee L_k)$, and where
 - each L_j is a *literal*: i.e. a possibly negated atom
- For simplicity, consider a *propositional* P : where each atom is a proposition.
- A model for P makes every one of C_i 's in P true.
- Let G be a literal (called "Goal")
- Consider the question: $P \stackrel{?}{\models} G$
- We can use a proof procedure, based on *resolution* to answer this question.

Resolution (Propositional case)

$$\frac{(A \vee C_1) \quad (\neg A \vee C_2)}{(C_1 \vee C_2)}$$

- This is a form of *Proof Rule*.
(Modus Ponens is a special case of Resolution)
- $\{(A \vee C_1), (\neg A \vee C_2)\} \models (C_1 \vee C_2)$
- How to construct *proofs* using resolution?

Use the following two rules:

$$\frac{\{C\} \cup P \vdash C}{\{C\} \cup P \vdash C} \quad (\in P) \quad \frac{P \vdash (A \vee C_1) \quad P \vdash (\neg A \vee C_2)}{P \vdash (C_1 \vee C_2)} \quad \text{Res}$$

- A proof system based on resolution is sound, but not complete.
- E.g., $p \models (p \vee q)$ but there is no way to derive it using Resolution.

Resolution Proof (An Alternative View)

- The clauses of P are all in a “soup”.
- Resolution rule picks two clauses from the “soup”, of the form $A \vee C_1$ and $\neg A \vee C_2$
- and adds $C_1 \vee C_2$ to the “soup”.
- The newly added clause can now interact with other clauses and produce yet more clauses.
- Ultimately, the soup consists of all clauses C such that $P \vdash C$.

Resolution Proof (An Example)

$$P = \{(p \vee q), (\neg p \vee r), (\neg q \vee r)\}$$

Here is a proof for $P \models r$:

Clause Number	Clause	How Derived
1	$p \vee q$	$\in P$
2	$\neg p \vee r$	$\in P$
3	$\neg q \vee r$	$\in P$
4	$q \vee r$	Res. 1 & 2
5	r	Res. 3 & 4

Refutation Proofs

- While resolution alone is insufficient to prove all logical consequences, *resolution is sufficient to show inconsistency* i.e. show when P has no model.
- This leads to *Refutation* proofs for showing logical consequence.
- Say we want to determine $P \stackrel{?}{\models} r$, where r is a proposition. This is equivalent to checking if $P \cup \{\neg r\}$ has an empty model.
- This we can check by constructing a resolution proof for $P \cup \{\neg r\} \vdash \square$ where \square denotes the unsatisfiable empty clause.

Refutation Proofs: An Example

Let $P = \{(p \vee q), (\neg p \vee r), (\neg q \vee r), (p \vee s)\}$, and $G = (r \vee s)$

Clause Number	Clause	How Derived
1	$p \vee q$	$\in P \cup \neg G$
2	$\neg p \vee r$	$\in P \cup \neg G$
3	$\neg q \vee r$	$\in P \cup \neg G$
4	$\neg r$	$\in P \cup \neg G$
5	$\neg s$	$\in P \cup \neg G$
6	$q \vee r$	Res. 1 & 2
7	r	Res. 3 & 6
8	\square	Res. 4 & 7

Refutation in Predicate Logic

```

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

```

- For what values of Q is $\text{anc}(\text{tom}, Q)$ a logical consequence of the above program?
- Assume $\forall Q. \neg \text{anc}(\text{tom}, Q)$
- We know that $\forall X, Y. \text{anc}(X, Y) \subset \text{parent}(X, Y)$ (from the program)
- Which is equivalent to $\forall X, Y. \neg \text{anc}(X, Y) \supset \neg \text{parent}(X, Y)$
- It then follows that $\forall Y. \neg \text{anc}(\text{tom}, Y) \supset \neg \text{parent}(\text{tom}, Y)$
- From M.P., we can conclude that $\forall Q. \neg \text{parent}(\text{tom}, Q)$
- But the program says $\text{parent}(\text{tom}, \text{bob})$.
- Hence we know that $\forall Q. \neg \text{anc}(\text{tom}, Q)$ is *not* true; the evidence is $Q = \text{bob}$.

Refutation: An Example

```

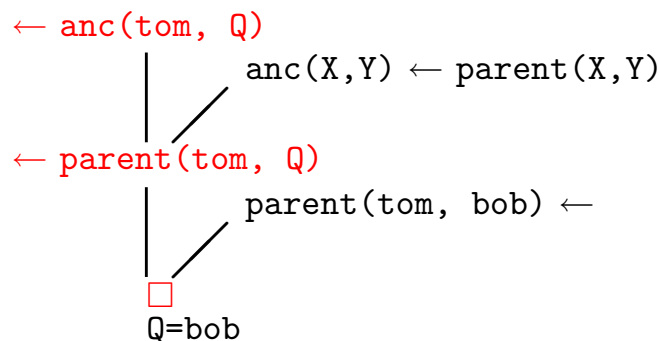
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).

```

```

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

```



Substitutions

A substitution is a mapping between variables and values (terms).

- Denoted by $\{X_1/t_1, X_2/t_2, \dots, X_n/t_n\}$ such that
 - $X_i \neq t_i$, and
 - X_i and X_j are distinct variables when $i \neq j$.
- Empty substitution is denoted by ϵ .
- A substitution is said to be a **renaming** if it is of the form $\{X_1/Y_1, \dots, X_n/Y_n\}$ and Y_1, \dots, Y_n is a permutation of X_1, \dots, 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 $X/t \notin \theta$ for any term t .
- Application of a substitution $\{X_1/t_1, \dots, X_n/t_n\}$ 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 of substitutions $\theta = \{X_1/s_1, \dots, X_m/s_m\}$ and $\sigma = \{Y_1/t_1, \dots, Y_n/t_n\}$:
 - First form the set $\{X_1/s_1\sigma, \dots, X_m/s_m\sigma, Y_1/t_1, \dots, Y_n/t_n\}$
 - Remove from the set $X_i/s_i\sigma$ if $s_i\sigma = X_i$
 - 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 $\sigma = \{Y/a\}$
 - $\theta\sigma = \{X/f(a), Y/a\}$
 - $\theta\sigma = \{X/f(Y), Y/a\}$
- Composition is not *commutative* but is *associative*: $\theta(\sigma\gamma) = (\theta\sigma)\gamma$
- Also, $E(\theta\sigma) = (E\theta)\sigma$

Idempotence

- A substitution θ 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 $\theta = \{X_1/t_1, \dots, X_n/t_n\}$,
 - $Dom(\theta) = \{X_1, X_2, \dots, X_n\}$
 - $Range(\theta) =$ set of all variables in t_1, \dots, t_n
- A substitution θ is idempotent iff $Dom(\theta) \cap Range(\theta) = \emptyset$

Unifiers

- A substitution θ is a **unifier** of two terms s and t if $s\theta$ is identical to $t\theta$.
- θ is a unifier of a set of equations $\{s_1 \doteq t_1, \dots, s_n \doteq t_n\}$, if for all i , $s_i\theta = t_i\theta$.
- A substitution θ is more general than σ (written as $\theta \succeq \sigma$) if there is a substitution ω such that $\sigma = \theta\omega$
- 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) $\theta \succeq \sigma$
- 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
 - θ_2 is a most general unifier

Equations and Unifiers

- A set of equations \mathcal{E} is in **solved form** if it is of the form $\{X_1 \doteq t_1, \dots, X_n \doteq t_n\}$ iff no X_i appears in any t_j .
- Given a set of equations $\mathcal{E} = \{X_1 \doteq t_1, \dots, X_n \doteq t_n\}$ the substitution $\{X_1/t_1, \dots, X_n/t_n\}$ is an idempotent mgu of \mathcal{E} .
- Two sets of equations \mathcal{E}_1 and \mathcal{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 $\{s \doteq t\}$.
If there is no equivalent solved form, there is no mgu.

A Simple Unification Algorithm (via Examples)

- Example 1: Find the mgu of $f(X, g(Y))$ and $f(g(Z), Z)$

$$\begin{aligned} \{f(X, g(Y)) \doteq f(g(Z), Z)\} &\Rightarrow \{X \doteq g(Z), g(Y) \doteq Z\} \\ &\Rightarrow \{X \doteq g(Z), Z \doteq g(Y)\} \\ &\Rightarrow \{X \doteq g(g(Y)), Z \doteq g(Y)\} \end{aligned}$$

- Example 2: Find the mgu of $f(X, g(X), b)$ and $f(a, g(Z), Z)$

$$\begin{aligned} \{f(X, g(X), b) \doteq f(a, g(Z), Z)\} &\Rightarrow \{X \doteq a, g(X) \doteq g(Z), b \doteq Z\} \\ &\Rightarrow \{X \doteq a, g(a) \doteq g(Z), b \doteq Z\} \\ &\Rightarrow \{X \doteq a, a \doteq Z, b \doteq Z\} \\ &\Rightarrow \{X \doteq a, Z \doteq a, b \doteq Z\} \\ &\Rightarrow \{X \doteq a, Z \doteq a, b \doteq a\} \\ &\Rightarrow \mathbf{fail} \end{aligned}$$

A Simple Unification Algorithm

Given a set of equations \mathcal{E} :

repeat

select $s \doteq t \in \mathcal{E}$;

case $s \doteq t$ **of**

1. $f(s_1, \dots, s_n) \doteq f(t_1, \dots, t_n)$:

replace the equation by $s_i \doteq t_i$ for all i

2. $f(s_1, \dots, s_n) \doteq g(t_1, \dots, t_m)$, $f \neq g$ or $n \neq m$:

halt with **failure**

3. $X \doteq X$: remove the equation

4. $t \doteq X$: where t is not a variable

replace equation by $X \doteq t$

5. $X \doteq t$: where $X \neq t$ and X occurs more than once in \mathcal{E} :

if X is a proper subterm of t

then halt with **failure** (5a)

else replace all other X in \mathcal{E} by t (5b)

until no action is possible for any equation in \mathcal{E}

return \mathcal{E}

A Simple Unification Algorithm (More Examples)

- Example 1: Find the mgu of $f(X, g(Y))$ and $f(g(Z), Z)$
 - $\{f(X, g(Y)) \doteq f(g(Z), Z)\} \Rightarrow \{X \doteq g(Z), g(Y) \doteq Z\}$ case 1
 - $\Rightarrow \{X \doteq g(Z), Z \doteq g(Y)\}$ case 4
 - $\Rightarrow \{X \doteq g(g(Y)), Z \doteq g(Y)\}$ case 5b
- Example 3: Find the mgu of $f(X, g(X))$ and $f(Z, Z)$
 - $\{f(X, g(X)) \doteq f(Z, Z)\} \Rightarrow \{X \doteq Z, g(X) \doteq Z\}$ case 1
 - $\Rightarrow \{X \doteq Z, g(Z) \doteq Z\}$ case 5b
 - $\Rightarrow \{X \doteq Z, Z \doteq g(Z)\}$ case 4
 - \Rightarrow **fail** case 5a

Complexity of the unification algorithm

Consider

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

- By applying **case 1** of the algorithm, we get

$$\{X_1 = f(X_0, X_0), X_2 = f(X_1, X_1), \dots, X_n = f(X_{n-1}, X_{n-1})\}$$

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

Most General Unifiers

- Note that mgu stands for a most general unifier.
- There may be more than one mgu. E.g. $f(X) \doteq f(Y)$ has two mgus:
 - $\{X/Y\}$
 - $\{Y/X\}$
- If θ is an mgu of s and t , and ω is a *renaming*, then $\theta\omega$ is an mgu of s and t .
- If θ and σ are mgus of s and t , then there is a renaming ω such that $\theta = \sigma\omega$.

SLD Resolution

SLD Resolution

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

where:

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

SLD resolution (contd.)

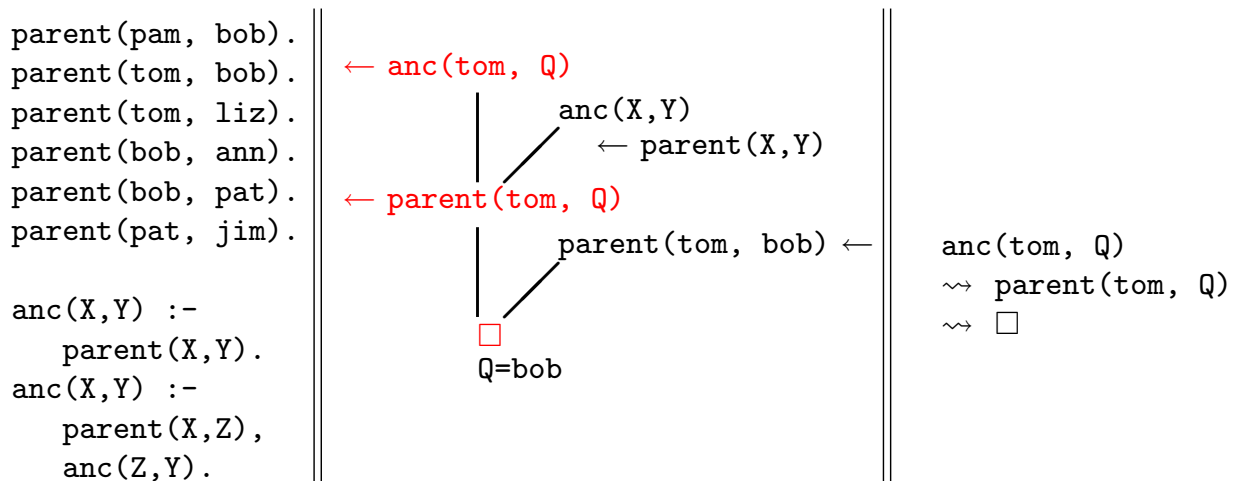
- When the resolution rule is applied, from a goal G and a clause C , we get a new goal G' .
We then say that G' is derived directly from G and C :

$$G \xrightarrow{C} G'$$

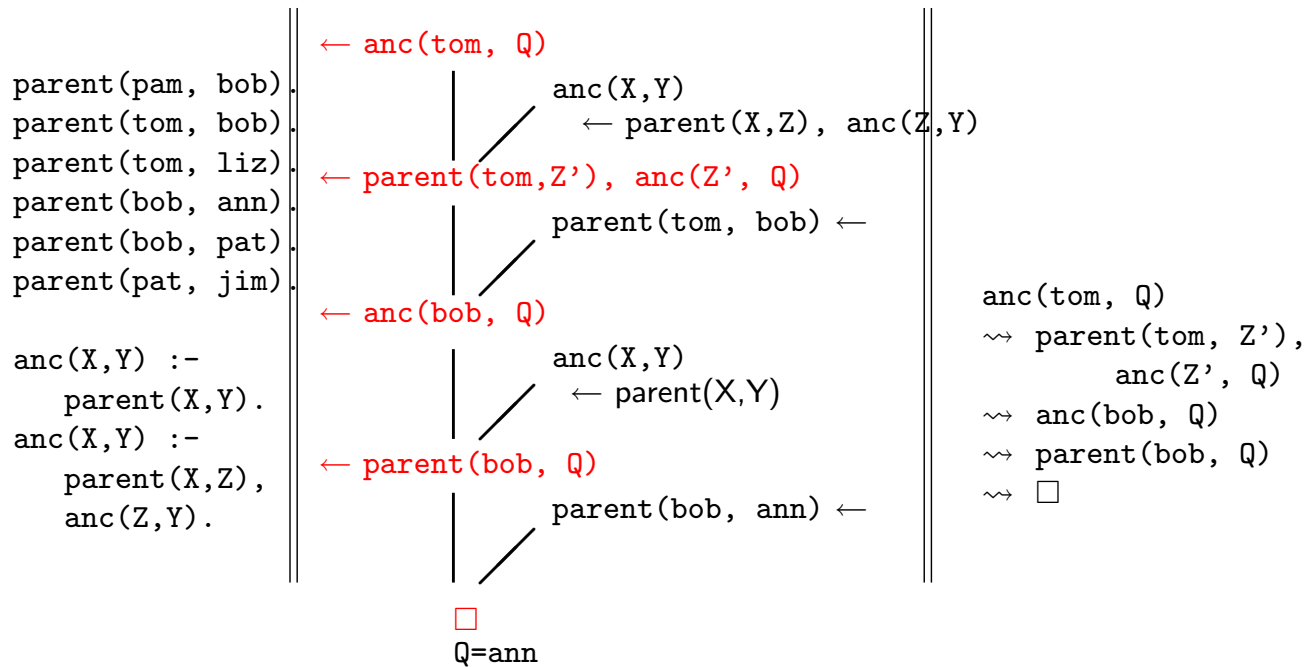
- An **SLD Derivation** is a sequence

$$G_0 \xrightarrow{C_0} G_1 \dots G_i \xrightarrow{C_i} G_{i+1} \dots$$

Refutation & SLD Derivation



SLD Derivation (contd.)



Computed Answer Substitution

- Let $\theta_0, \theta_1, \dots, \theta_{n-1}$ 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 *computed substitution* of the derivation.

- Example:

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

- 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 (contd.)

- 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 (contd.): The computed answer substitution for the above SLD refutation is
 $\{X'/\text{tom}, Y'/\text{ann}, Z'/\text{bob}, X''/\text{bob}, Y''/\text{ann}, Q/\text{ann}\}$ restricted to Q :

$$\{Q/\text{ann}\}$$

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:
`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 derivation of `grandfather(a,Q)` is:

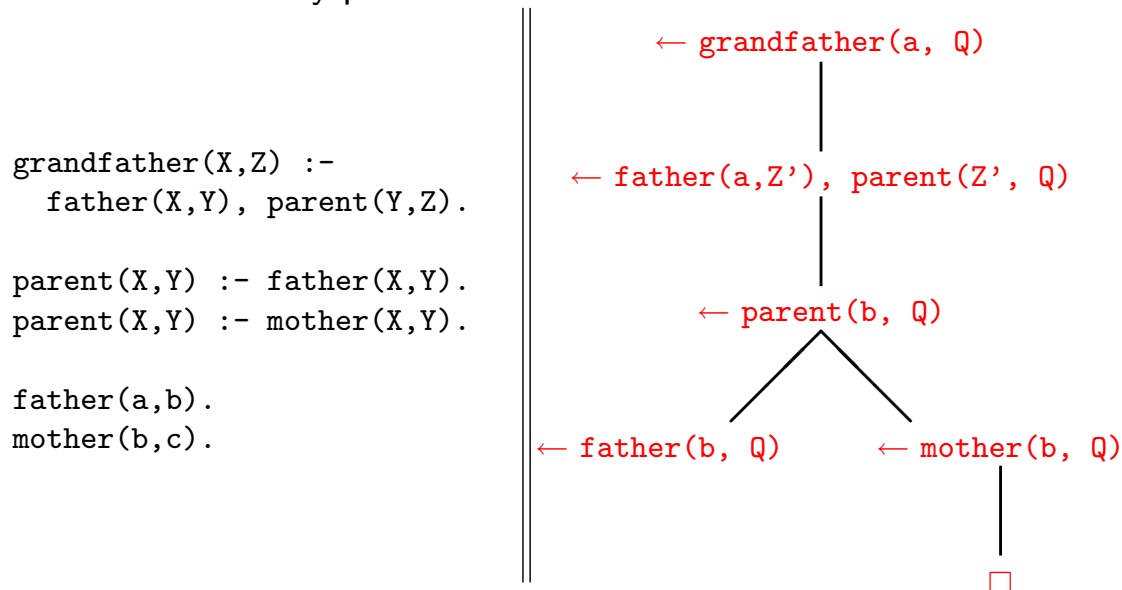
$$\rightsquigarrow \text{father}(a, Y'), \text{parent}(Y', Q)$$

$$\rightsquigarrow \text{parent}(b, Q)$$

$$\rightsquigarrow \text{father}(b, Q)$$

SLD Tree

A tree where every path is an SLD derivation



Soundness of SLD resolution

- *Let P be a definite program, \mathfrak{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 $(\forall F) \models (\forall F')$.