LECTURE 8
LOGIC PROGRAMMING

Let us look now how PROLOG finds <u>REFUTATION</u> <u>PROOF</u>.

By looking at some examples we outline (formally defined later) the <u>PROLOG'S REFUTATION</u>.

Consider **Example 4** from **LECTURE 4** or **Lab 1**.

Example 1:

$$\Delta = \{p;\ q;\ r;\ w \leftarrow p, r;\ v \leftarrow w, q, s;\ s \leftarrow w\}$$

$$\boxed{\Delta \models v}\ ?$$

How does PROLOG find REFUTATION

We shall outline some aspects of SLD - REFUTATION

LECTURE 8
LOGIC PROGRAMMING

$\sim v$ is added to Δ.

1 Step: $\boxed{\text{root goal} \equiv \sim v}$
resolved with head of clause $\in \Delta$

↑ 1st clause that maches!

$$\frac{\begin{array}{c}\sim v \\ v \leftarrow w, q, s\end{array}}{\leftarrow w, q, s}$$

Resolvent_1

2 Step: the current goal is the last RESOLVENT (i.e. RESOLVENT_1)

QUESTION: which atom to choose?
w? or q? or s?

PROLOG selects the 1st left-most one. So

$$\frac{\begin{array}{c}\leftarrow \cancel{w}, q, s \\ \cancel{w} \leftarrow p, r\end{array}}{}$$

← 1st clause $\in \Delta$ that matches w

Next question: which RESOLVENT?

$\leftarrow p, r, q, s.$? or 1)

$\leftarrow q, s, p, r.$? 2)

LECTURE 8.
LOGIC PROGRAMMING

PROLOG implements <u>version 1)</u>
i.e. moves all remaining atoms from the RESOLVENT_1 (current goal) to the right of the program clause body. Thus

$$\leftarrow \cancel{w}, q, s.$$
$$\cancel{w} \leftarrow p, r.$$
$$\overline{\qquad\qquad\qquad\qquad}$$
$$\leftarrow \underline{p}, r, q, s. \qquad \text{RESOLVENT\_2 current goal}$$

<u>3. Step:</u>

$$\leftarrow \cancel{p}, r, q, s.$$
$$\cancel{p}$$
$$\overline{\qquad\qquad\qquad\qquad}$$
$$\leftarrow \underline{r}, q, s. \qquad \text{RESOLVENT\_3 current goal}$$

<u>4. Step:</u>

$$\leftarrow \cancel{r}, q, s.$$
$$\cancel{r}$$
$$\overline{\qquad\qquad\qquad\qquad}$$
$$\leftarrow q, s. \qquad \text{RESOLVENT\_4 current goal}$$

LECTURE 8
LOGIC PROGRAMMING

5. Step:

$$\begin{array}{c} \leftarrow \cancel{q}, s \\ \cancel{q} \\ \hline \leftarrow s \end{array}$$

RESOLVENT_5
current goal

6. Step:

$$\begin{array}{c} \leftarrow \cancel{s} \\ \cancel{s} \leftarrow w \\ \hline \leftarrow w \end{array}$$

RESOLVENT_6
current goal

7. Step:

$$\begin{array}{c} \leftarrow \cancel{w} \\ \cancel{w} \leftarrow p, r \\ \hline \leftarrow p, r \end{array}$$

RESOLVENT_7
current goal

8. Step:

$$\begin{array}{c} \leftarrow \cancel{p}, r. \\ \cancel{p} \\ \hline \leftarrow r \end{array}$$

RESOLVENT_8
current goal

9. Step:

$$\begin{array}{c} \leftarrow r \\ r \\ \hline \square \end{array}$$

RESOLVENT_9
contradiction

LECTURE 8
LOGIC PROGRAMMING

Of course, there may exists other refutations which are not:

SLD-refutations e.g.

$$
\begin{array}{c}
\leftarrow v \\
v \leftarrow w, q, s \\
\hline
\leftarrow w, q, \cancel{s} \\
\cancel{s} \leftarrow w \\
\hline
\leftarrow w, \cancel{q} \\
\cancel{q} \\
\hline
\leftarrow \cancel{w} \\
\cancel{w} \leftarrow p, r \\
\hline
\leftarrow \cancel{p}, r \\
\cancel{p} \\
\hline
\leftarrow \cancel{r} \\
\cancel{r} \\
\hline
\square
\end{array}
$$

2 STEPS are not part of SLD-REFUTATIO

A CONTRADICTION

$\square$.

Lecture 8
Logic Programming

SLD-Refutation satisfies the following:

$\boxed{a)}$ always the resolution takes:

RESOLUTION (current goal, clause)

resolvent

$\underbrace{\leftarrow A_1, A_2, \ldots, A_n}_{\text{negative goal clause}}$

1st program clause which head "matches" A_1

$\boxed{b)}$ in selecting atoms from the body of current goal we select first left-most one i.e. A_1

LECTURE 8.

LOGIC PROGRAMMING

$\boxed{c)}$ once we resolve

$$\leftarrow \cancel{A_1}, A_2, \ldots, A_n. \leftarrow \text{current goal}$$

$$\cancel{A_1} \leftarrow B_1, B_2, \ldots, B_m.$$

program clause

we put $A_2, \ldots, A_n$ on the right-hand side of $B_1, B_2, \ldots, B_m$:

$$\leftarrow B_1, B_2, \ldots, B_m, A_2, \ldots, A_n.$$

RESOLVENT

$\boxed{d)}$ if a given <u>selected atom</u> does not yield for a given program clause a $\square$ (contradiction) then we take the 2nd clause in Δ which "matches" selected atom, and so on – <u>BACKTRACKING</u>.

$\boxed{e)}$ Recall: $\leftarrow B_1, \ldots, B_n$ or

$A_1, \ldots A_k \leftarrow C_1 \ldots C_m$

<u>clauses not allowed</u> in Δ. $k \geq 2$

LECTURE 8
LOGIC PROGRAMMING

REMARK:

- so-far A_i were variable free
- in general case we need SLD-RESOLUTION to be covered (later, UNIFICATION & SUBSTITUTION)

SLD-REFUTATION

OUTLINED IN a) – d) + e)
is a restriction to general logic programming REFUTATION.

(i) Δ – with different orders of clauses is semantically equivalent

(ii) Δ – with clauses having different order of atoms in their bodies is also semantically equivalent.

But for

$$\underset{\boxed{\text{LOGIC PROGRAMMING PROGRAM}}}{\Delta} \longrightarrow \Delta_P \nwarrow \boxed{\text{PROLOG PROGRAM}}$$

above (i) & (ii) have impact on execution of the program.

So:

- order of program clauses
&
- order of atoms in program clauses

$\boxed{\text{matter!}}$

This is the 1st difference between

- theoretical logic programming.
&
- PROLOG programming.

LECTURE 8
LOGIC PROGRAMMING

Example 2:

$$\Delta_{P_1} = \{ q \leftarrow p;\ p \leftarrow q;\ p \leftarrow r;\ r \}$$

$$\Delta_{P_2} = \{ r;\ p \leftarrow r;\ q \leftarrow p;\ p \leftarrow q \}$$

Both Δ_{P_1} & Δ_{P_2} differ only by order of clauses — semantically equivalent

Thus for $\boxed{\begin{array}{c} \Delta_{P_1} \models q \\ \Delta_{P_2} \models q \end{array}}$

we should have <u>either No</u> (for both queries) <u>or YES</u> (for both queries).

For Δ_{P_1} we have:

$$\begin{array}{c} \sim q \\ \underline{q \leftarrow p} \\ \leftarrow p \\ \underline{p \leftarrow q} \\ \leftarrow q \end{array}$$

an infinite loop

in PROLOG: "out of local stack"

LECTURE 8
LOGIC PROGRAMMING

For Δ_{P_2}

$$\frac{\dfrac{\dfrac{\leftarrow q \qquad q \leftarrow p}{\leftarrow p} \qquad p \leftarrow r}{\leftarrow r} \qquad r \leftarrow}{\square}$$

$\Rightarrow \Delta_{P_2} \models q$

PROLOG: yes

$\square$.

To see potential ∞-loops one may draw the corresponding search space for a given Δ_P & Q

SLD - TREE*

* - formal definition later

LECTURE 8.
LOGIC PROGRAMMING

Example 3:

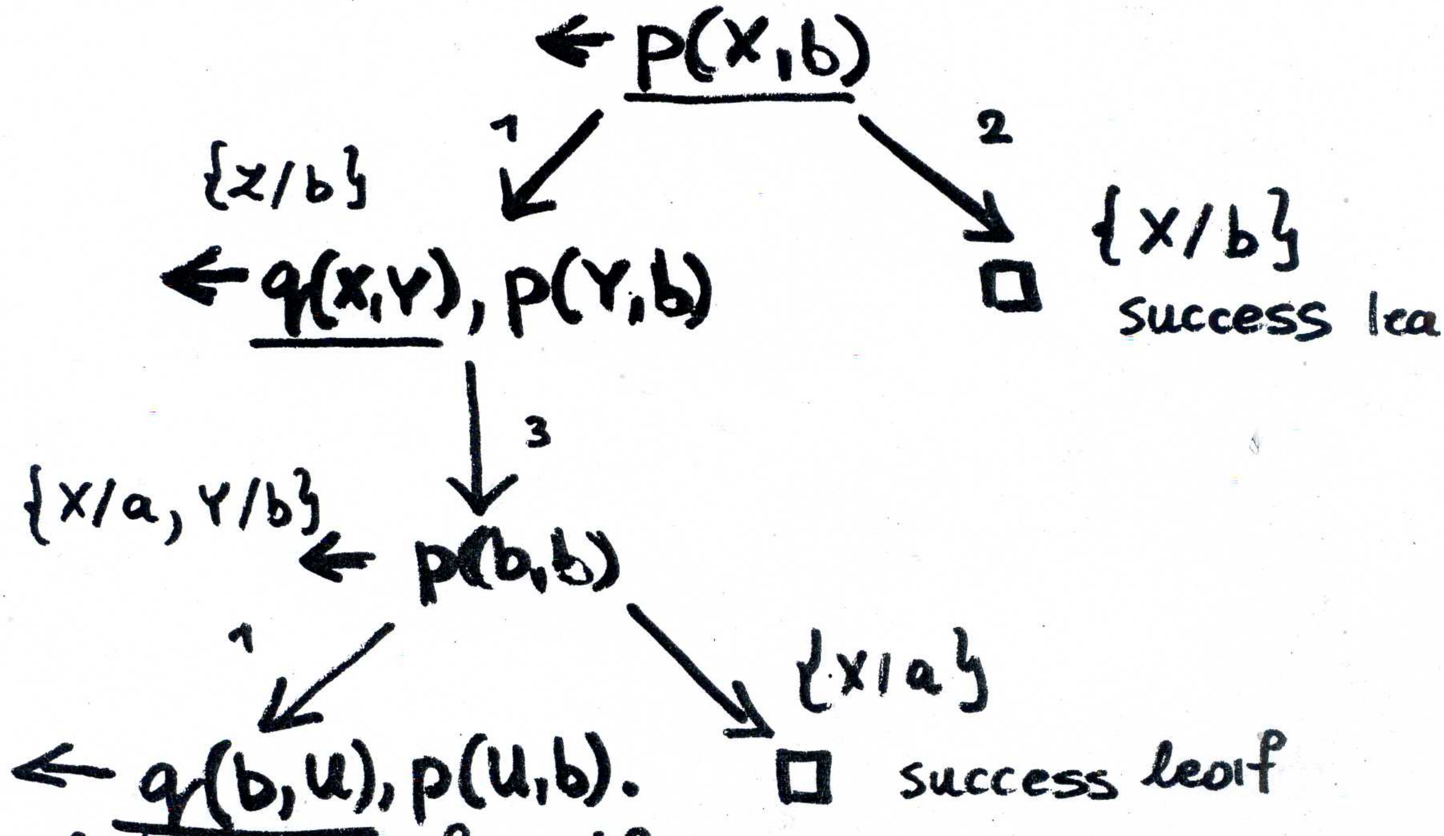
$$\Delta_P = \begin{cases} p(X,Z) :- q(X,Y),\ p(Y,Z). & 1) \\ p(X,X). & 2) \\ q(a,b). & 3) \end{cases}$$

Note that 1) e 2) are UNIVERSALLY QUANTIFIED

QUERY: $\Delta_P \stackrel{?}{\models} p(X,b).$

$\equiv \sim (\exists x\ p(X,b))$

SLD-TREE



This is the whole search space for PROLOG & Δ_p with $\boxed{Q \equiv \exists x\, p(X,b).}$

This tree is searched* by **depth-first search** from left to right.

Once 1st success is reached with substitution $\{X/a\}$

PROLOG will reply: $X = a$ yes

and waits:

- if RTN is hit ⇒ we terminate searching the SLD tree. PROLOG waits for the next QUERY.
- if ";" semicolon is hit the search for □ resumes and we get 2nd success with
  $X=b$ yes

* later explained more.

- again if RTN is hit $\Rightarrow$ PROLOG teminates the search for next successes in SLD tree, and waits for the next QUERY.
- if ; is hit then PROLOG resumes searching SLD tree. There are no more $\Rightarrow$ it reports

No

& waits for the next QUERY.

REMARK:

It is easy to prove that:

$$\overset{(ii)}{\exists y\, p(f(y))} \leftarrow \overset{(i)}{\forall x\, p(x)} \quad (\blacktriangle)$$

is **VALID** – i.e. true for each interpretation.

The "would-be counterexample"

$\begin{cases} D = [0,1] \quad f(y) = y-1 \quad p(x) \text{ is true if } x \geq 0 \\ \text{Then as (i) is true} \\ \quad\quad\quad \text{(ii) is false} \end{cases} \Rightarrow (\blacktriangle) \text{ is false}$

fails as $f: D \not\to D$.

Lecture 8
Logic Programming

Remark:

(i) $p(X) :- q(X).$ $\equiv \forall X\, p(X) \leftarrow q(X).$

(ii) $q(X).$ $\equiv \forall X\, q(X).$

It is clear that:

- X appearing in 2 different clauses (i) & (ii) are in fact not the same as (i) & (ii) $\equiv$

(i) $p(X) :- q(X).$

(ii) $q(Y).$

- X appearing in the same clause cannot be changed i.e

$$p(X,Y) \leftarrow q(X).$$
$$\not\equiv$$
$$p(X,Y) \leftarrow q(Z)$$

as they are bound by the same universal quantifier.

LECTURE 8

LOGIC PROGRAMMING

The last remark is important when the so-called

"occur check"

for

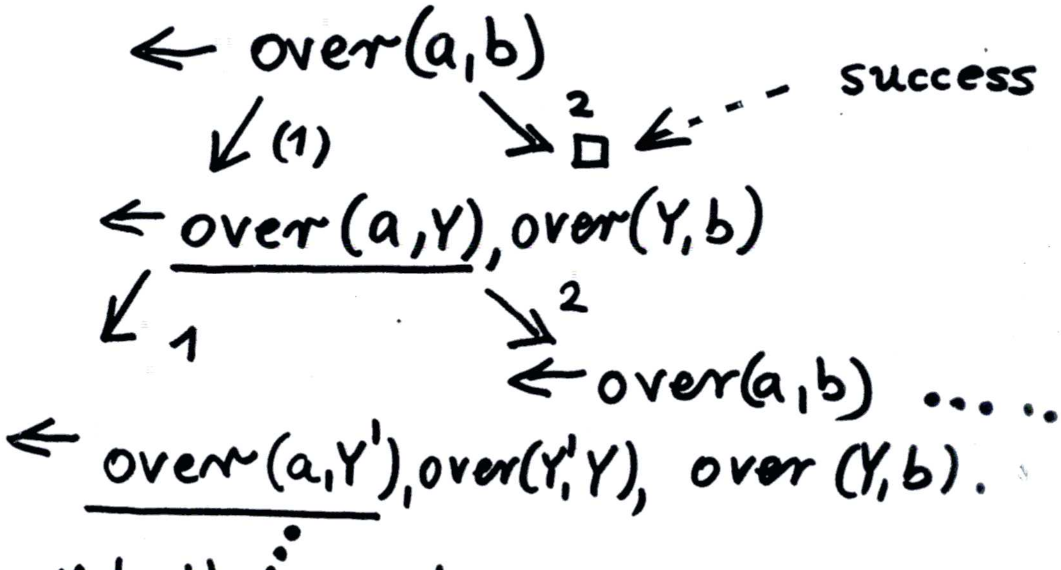
UNIFICATION ALGORITHM

is discussed.

Example 4:

1) over(X,Z) :- over(X,Y), over(Y,Z)
over(a,b).
over(b,c). over(c,d).

For ■ in Example 8 (LECTURE 7)



so we build the goal

"out of local stack"

LECTURE 8
LOGIC PROGRAMMING

One method is to move boundary conditions in front of 1).

But it may not help.

Check also an alternative !

from Example 8 (LECTURE 7).

$\square$

Before we pass in the next lecture to

a) SUBSTITUTION
b) UNIFICATION

let us sum-up the overall procedure in logic programming (including PROLOG SLD-refutation)

LECTURE 8
LOGIC PROGRAMMING

- we may ask to resume searching for next successes by hitting ";"
- by proving $\square \in (\Delta \cup \{\sim Q\})^{*}$ $\equiv$ we use the result that $\Delta \models Q$.

Proving <u>Validity</u> of a given formula requires <u>a formal proof</u>.

e.g $$\boxed{\sim \exists x\, Q(x) \equiv \forall x \sim Q(x)}$$
$$\underbrace{\sim \exists x\, Q(x)}_{\text{LEFT}} \equiv \underbrace{\forall x \sim Q(x)}_{\text{RIGHT}}$$

'$\equiv$' is equivalent to

LEFT $\underset{(i)}{\Leftarrow}$ RIGHT & RIGHT $\underset{(ii)}{\Leftarrow}$ LEFT

So we prove now both (i) & (ii).

Assume $\mathcal{I}$ is an arbitrary interpretation:

LECTURE 8
LOGIC PROGRAMMING

a) D is arbitrary

b) constant symbols $\rightarrow$ specific symbols from D

$$a_i \rightarrow a_i' \quad \text{arbitrary}$$

c) function symbols $\Rightarrow$ specific symbols of functions

$$f_i/n_i \rightarrow f'/n_i \quad \text{arbitrary}$$

d) predicate symbols $\Rightarrow$ specific symbols of predicates

$$P_j/n_j \rightarrow P_j'/n_j$$

(i) Assume $\boxed{\mathcal{I}(\forall x \sim Q(x)) = \text{true.}}$

For each $e \in D$ $\sim Q'(e)$ is true

For each $e \in D$ $Q'(e)$ is false

$\exists x\ Q(x)$ is false

$\underline{\sim \exists x\ Q(x) \text{ is true}}$. Thus LEFT is true

(ii) Assume $\boxed{\mathcal{I}(\sim\exists x\, Q(x)) = true}$

$\exists x\, Q(x)$ is false

for each $e \in D$ $\quad Q'(e)$ is false

for each $e \in D$ $\quad \sim Q'(e)$ is true

$\forall x \sim Q(x)$ is true

Thus RIGHT is true.

The PROOF is complete.

LECTURE 8
LOGIC PROGRAMMING

- program Δ consists of non-negative Horn clauses

$$A \leftarrow A_1, A_2, \ldots, A_n. \qquad \text{(rules)}$$

$$A \leftarrow \qquad \text{(facts)}$$

- the query is

(*) $\exists y_1 \exists y_2 \ldots \exists y_n \,(B_1 \wedge B_2 \wedge \ldots \wedge B_m)$

$$\overset{?}{\models_{\Delta}}$$

- PROLOG uses SLD-refutation & negates (*) to add negative clause

$$\forall y_1 \forall y_2 \ldots \forall y_n \sim B_1 \vee \sim B_2 \vee \ldots \vee \sim B_m$$

& shows whether there exists or not SLD-refutation

- if YES it reports the <u>successful</u> SUBSTITUTION
- if NO it reports NO.