LECTURE 6
LOGIC PROGRAMMING

We have defined **well-formed** strings for PREDICATE LOGIC (SYNTAX)

How to attach the **meaning** to:

$$\text{string} \longrightarrow \{0, 1\}$$

0: false, 1: true

DEFINITION 1:

Relation of arity n (R/n) over set $\mathbb{D}$ is some **subset**

$$\tilde{\mathcal{R}} \subset \mathbb{D}^n = D \times \dots \times \mathbb{D}$$

We can represent R as **a function**:

$$R: \underbrace{\mathbb{D} \times \dots \times \mathbb{D}}_{n\text{-times}} \longrightarrow \{0, 1\}$$

such that for a n-tuple $(d_1, d_2, .., d_n)$ $d_i \in D$:

$$R(d_1, d_2, \dots, d_n) = 1$$

iff

$$(d_1, d_2, \dots, d_n) \in \tilde{\mathcal{R}} \subset \mathbb{D}^n$$

LECTURE 6
LOGIC PROGRAMMING

Example 1:

(i) $Pr/1$: $\widetilde{Pr}(x) \subset \mathbb{N}$ is true iff x is prime

$$\widetilde{Pr}(x) = \{2, 3, 5, 7, 11, \dots\} \subseteq \mathbb{N}$$

(ii) $Sq/2$: $\widetilde{Sq}(x,y) \subset \mathbb{N} \times \mathbb{N}$ iff $y = x^2$

$$\widetilde{Sq}(x,y) = \{(0,0), (1,1), (2,4), \dots\} \subseteq \mathbb{N}^2$$

We introduce now for **a language**:

$$\mathcal{L} = \{\underbrace{a_1, \dots, a_n}_{\text{constants}}, \underbrace{x_1, \dots, x_m}_{\text{variables}}, \underbrace{f_1/n_1, \dots, f_k/n_k}_{\text{functions}}, \underbrace{P_1/n_1, \dots P_l/n_l}_{\text{predicates}}\}$$

the definition of **any interpretation**:

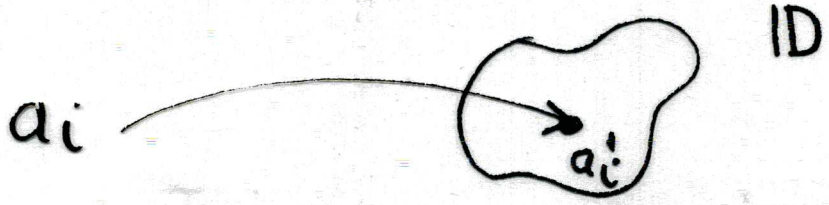
LECTURE 6
LOGIC PROGRAMMING

DEFINITION 2:

Let D be a non-empty set, called the **domain of discourse**.
Then **an interpretation** $\mathcal{I}$ (for language $\mathcal{L}$) :

a) for each **constant** $a_i \in \mathcal{L}$ we assign:

$$a_i \longrightarrow a_i' \in \mathbb{D}$$



b) for each f_i/n_i (function symbol) we assign **a function**

$$f_i \to f_i' : \mathbb{D}^{n_i} \longrightarrow \mathbb{D}$$

c) for each **predicate symbol** P_i/n_i we assign **a relation** P'

$$P_i \longmapsto P_i' : \mathbb{D}^{n_i} \longrightarrow \{0,1\}.$$

LECTURE 6
LOGIC PROGRAMMING

An **pre**interpretation is not sufficient.
(for semantics)

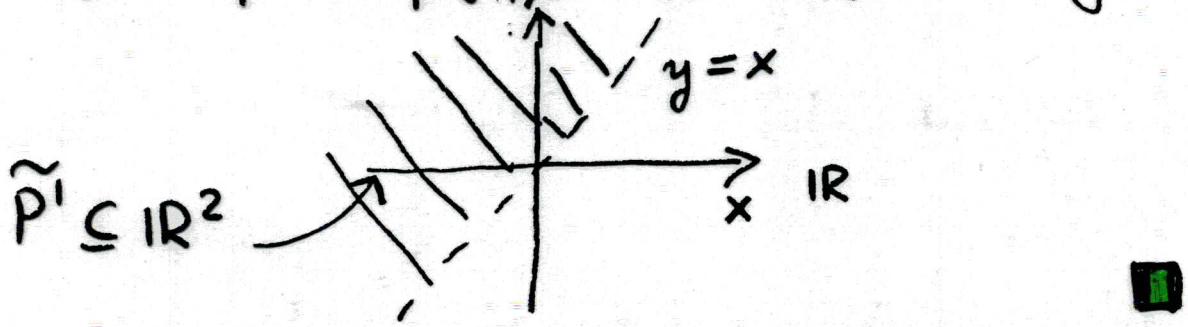
Example 2: $\mathcal{L} = \{a_1, a_2, f/1, p/2\}$
↖ language

Let $\mathbb{D} = \mathbb{R}$ (set of real numbers)

For a) $a_1 \rightarrow 0 \in \mathbb{R}$
$a_2 \rightarrow 1 \in \mathbb{R}$

b) $f \rightarrow f^I : x \rightarrow x^2 \in \mathbb{R}$

c) $p \rightarrow p^I(x,y)$ true iff $x < y$



- constants refer to specific elements in $\mathbb{D}$.
- functions f/n refer to actual functions on $\mathbb{D}^n \rightarrow \mathbb{D}$.
- predicates p/m represent real-world relationship of the objects in $\mathbb{D}^m$.

LECTURE 6
LOGIC PROGRAMMING

Example 3:

For $\mathcal{L} = \{$ jack, bill, jennifer, mary, married/2 $\}$

Let $\mathbb{D} = \{$ Jack, Bill, Jennifer, Mary $\}$

a) jack → Jack
bill → Bill
jennifer → Jennifer
mary → Mary

b) no function symbols here

c) p → p'(X,Y) is true iff
|||
married
|||
married'
X is married to Y.

Assume that only Mary is married to Bill.

Note that only for **closed formulas** we can determine **semantics**:

married'(Bill, Mary) = true
married'(Bill, Bill) = false

This can be done from relation R.
But even those **closed** formulas:

(•) $\exists x\ married'(x, Bill) = true$
$\forall x\ married'(x, Bill) = false$

have the **unique meaning**! (to be soon defined).

But **open formula** (with free variable)

(••) $married'(X, Mary) \stackrel{?}{=} \begin{cases} true & X = Bill \\ false & X \neq Bill. \end{cases}$

free variable

gives problem!

■.

To deal with the above:

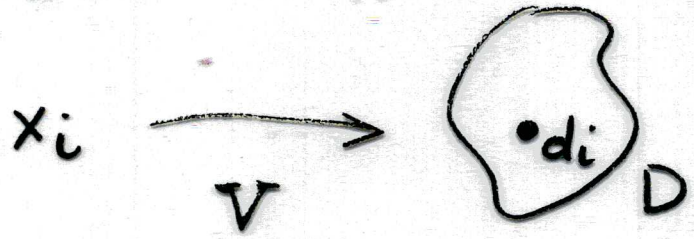
we define **a variable assignment**

That would cover (••).
Definition of interpretation & variable assignment covers (•).

LECTURE 6
LOGIC PROGRAMMING

DEFINITION 3:

A variable assignment V with respect to an interpretation $\mathcal{I}$ is an assignment to each variable in language an element from D:



Once V is given:

$\Rightarrow$ any term t represents $t' \in D$

So given:

- an interpretation $\mathcal{I}$
- = a variable assignment V based on $\mathcal{I}$

one can introduce a **full semantics** in PREDICATE LOGIC irrespectively whether f is closed or open.

LECTURE 6
LOGIC PROGRAMMING

DEFINITION 4:

Let I be an interpretation for language L with domain D.
Let V be a variable assignment based on D.
Then any formula from L is given a truth value as follows:

1*. for $f \equiv p(t_1, t_2, \ldots, t_n)$ being an atom the truth value is:

$$p'(t_1', t_2', \ldots, t_n')$$

2*. if f_1 & f_2 are **well-formed** then

f_1	f_2	$\sim(f_1)$	$(f_1) \vee (f_2)$	$(f_1) \wedge (f_2)$	$f_2 \leftarrow f_1$
true	true	false	true	true	true
true	false	false	true	false	false
false	true	true	true	false	true
false	false	true	false	false	true

Note: 1* & 2* are extensions of semantics of PROPOSITIONAL LOGIC.

3. for

$$f \equiv \exists x P$$

then the truth value of f is true if there exists some element $d \in D$ such that P is true when all occurances of variable x are assigned to value d. All other variables retain their variable assignment.

4. for

$$f \equiv \forall x P$$

then the truth value of f is true if for every $d \in D$, P is true when all occurances of the variable x are assigned to the value d. All other variables retain their variable assignment.

The truth value of f being **CLOSED** formula **does not depend** on **variable assignment!**

LECTURE 6
LOGIC PROGRAMMING

From now on we assume to deal only with

CLOSED FORMULAS

e.g., terms are closed & $\forall \ldots \exists$ f have no free variables

Semantics depends now merely on

Interpretation !

Example 4:

Let $\mathcal{L} = \{ \underbrace{a, b}_{\text{constants}} ; \underbrace{s/1}_{\text{function}} ; \underbrace{p/2, q/1, r/2}_{\text{predicates}} \}$

Consider the following interpretation:

$D = \mathbb{Z}$ (set of integeres)

a) $a \to 0 \in \mathbb{Z}$
$b \to 1 \in \mathbb{Z}$

b) $s \to s' : x \to x+1 \qquad s' : \mathbb{Z} \to \mathbb{Z}$.

c) $p \to$ ".>." $\equiv P'(x,y)$ true if $x > y$

$q \to$ ".>0" $\equiv q'(x)$ true if $x > 0$

$r \to r'(x,y)$ true if x divides y

LECTURE 6

LOGIC PROGRAMMING

From symbolic formulas we can read back English sentences (under this interpretation I):

$\forall x \exists y\ p(x,y)$: For any integer x, there is an integer y such that y is less than x.
TRUE under I.

$\exists x \forall y\ p(x,y)$: There is an integer x such that x is greater than every integer y.
False under I.

$p(s(a), b)$: 1 is greater than 1
False under I.

$\forall x \left(q(x) \leftarrow p(x,a)\right)$: Any integer x greater than 0 satisfies $x > 0$.
TRUE.

LECTURE 6
LOGIC PROGRAMMING

Example 5:

Testing relationship between people.

Consider predicates:

parent/2, male/1, female/1

We start with simple facts:

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

female(pam).
male(tom).
male(bob).
female(liz).
female(ann).
female(pat).
male(joe).
male(jim).
```

← Pam is the parent of Bob. etc.

← gender of Pam.

← part of the file.pl

LECTURE 6
LOGIC PROGRAMMING

>> male(tom). (+RTN)

yes

>> male(X). (+RTN) "who is male"

X = tom

$Q \equiv \exists x\, male(X)$

Now - there are more than one solution.

(i) if we hit (+RTN) end of search

(ii) if we hit ";" a semi-colon another X is found by PROLOG

X = bob + (RTN)

↖ search for solution resumes

We can express other relationships: in terms of the previous ones.

child(X,Y) :- parent(Y,X).

↑

"A is child of B provided B is a parent of A."

(▲) mother(X,Y) :- parent(X,Y), female(X).

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

& even go analogously to more distant relations:

LECTURE 6
LOGIC PROGRAMMING

grandparent(X,Y) :- parent(X,Z), parent(Z,Y)

Now

>> child(bob, A).

A = pam

" Bob is a child of Pam".

(RTN +) from now on we omit it.

$Q \equiv \exists x\, child(bob, X)$.

>> mother(X, jim).

$$\Downarrow$$

$Q \equiv \exists x\, mother(X, jim)$..

" who is the mother of Jim ?"

here as we know PROLOG takes

$$\sim Q \equiv \forall x \sim mother(X, jim)$$

$$\Downarrow$$

$\leftarrow mother(X, jim)$ clausal form

original goal ↑

& repeatedly tries "all possible ways" to resolve it in order to **produce an empty clause** □.

1. it looks down the data base & finds:

(*) $mother(X,Y) \leftarrow parent(X,Y), female(X)$.

2. it tries to find **an instance of (*)** so that literals in the **goal clause** & **program clause (*)** **match**

this is called **UNIFICATION***

Here instance of (*) is:

mother(X, jim) ← parent(X, jim), female(X)

& goal

← mother(X, jim)

Now we can "resolve"** & get a new goal:

(▲) ← parent(X, jim), female(X).

current new goal

3. again **first literal in** (▲) is chosen & PROLOG tries to **match** with **the head** of some **program clause** (**1st one!**)

Here the 1st 5 do not match.

But the **sixth** one — Yes.

We do now **substitution** for X = joe in (▲).

* formally defineol later

** formally for PREDICATE LOGIC RESOLUTION defined later

← parent(joe, jim), female(joe).

parent(joe, jim).

← female(joe). Resolvent

↖ current goal

4. Now PROLOG cannot find in database the clause it matches current goal.

So far no empty clause is found!

5. PROLOG now "backtracks" to the first place, where there is more than one choice of selecting the program clause which:

"head matches the left-most literal of current goal".

6. Now we take the 2th clause

← parent(pat, jim), female(pat).
parent(pat, jim).

← female(pat). Resolvent

↖ current goal

LECTURE 6
LOGIC PROGRAMMING

Finally if we resolve with clause 12)

$$\leftarrow \text{female}(\text{pat}).$$
$$\text{female}(\text{pat}).$$

$$\leftarrow$$
$$\Vdash$$
$$\square$$

So we proved:

$$\square \in \left(\underbrace{\Delta \cup \{\leftarrow \text{mother}(\tilde{x}, \text{jim})\}}_{\tilde{\Delta}}\right)^{*}. \qquad \tilde{x} = \text{pat}.$$

Thus by **Th.1** & **Th.2**. (LECTURE 4)

$\tilde{\Delta}$ is unsatisfiable $\equiv$ $\Delta \models Q$ for $\tilde{x} = \text{pat}$.

>> yes
$X = \text{pat}$

PROLOG

"Pat is a mother of Jim".

Again if we:

LECTURE 6
LOGIC PROGRAMMING

a) hit [+RTN] keystroke

PROLOG terminates search for successful $\tilde{x}$ for which

$$\Delta \models mother(X, jim).$$

?

$$\exists x\ mother(X, jim).$$

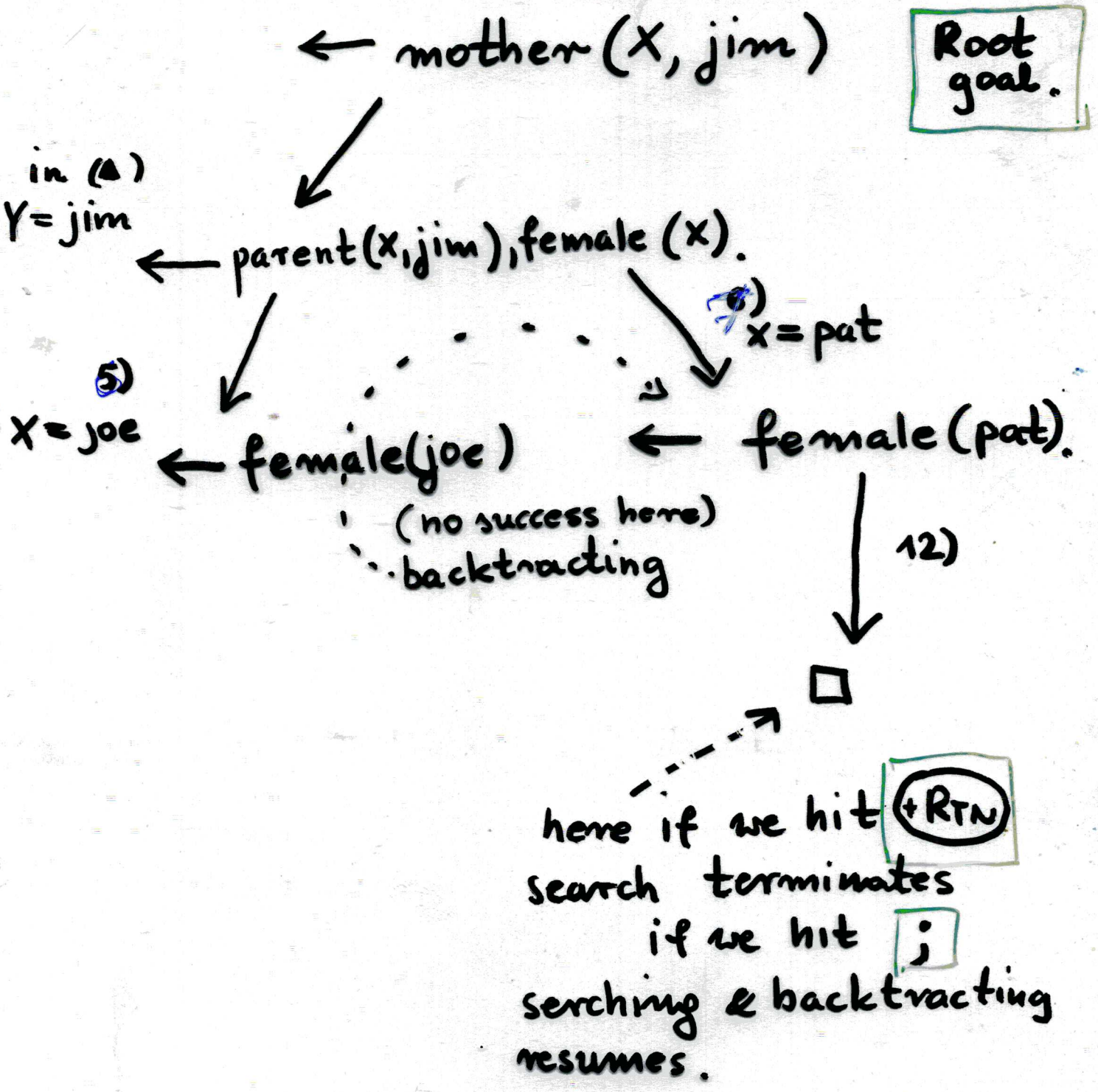
b) hit ';' PROLOG

- treats the last success for X=pat as failure
- resumes search for next possible successes.

One can represent the last execution of PROLOG program for query

$$\Delta \models \exists x\ mother(X, jim)$$

LECTURE 6
LOGIC PROGRAMMING



We can now show the

the RECURSIVE PROGRAMMING IN PROLOG:

LECTURE 6
LOGIC PROGRAMMING

Example 6:

For grandparents:

```
g-parent(X,Z):- parent(X,Y), parent(Y,Z).
```

But if we dig in in the genealogy then:

```
ggg-parent(X,Z):- parent(X,X1), parent(X1,X2),
                  parent(X2,X3), parent(X3,Z).
```

This is a bit

awkward !

The solution is

an RECURSIVE APPROACH

```
ancestor(X,Z):- parent(X,Z).
ancestor(X,Z):- parent(X,Y), ancestor(Y,Z).
```

Second clause is recursive.
First clause is "boundary conditions".

LOGIC PROGRAMMING

LECTURE 6

>> ancestor(X, pat)

$\Delta \models \exists x \, ancestor(X, pat).$

for previous example:

"What are the ancestors of Pat?"

X = bob ;

X = pam ;

X = tom;

No

So PROLOG asked to search for more finds all of them & once it cannot find it reports

No

& waits for the next QUERY

Try to draw a search space as an exercise!

LECTURE 6
LEGIC PROGRAMMING

Warning:

- because of the implementation issues of PROLOG interpreter*

though

```
ancestor(X,Z):-ancestor(Y,Z), parent(X,Y)
ancestor(X,Z):- parent(X,Z).
```

(**)

III semantically

(**) from Example 6

for query

```
>> ancestor(X,pat).
```

! out of local stack

[execution aborted]

Problem: recursion never reaches (***) the boundary conditions ≡ a cycle

* explained later

(***) it has to do with SLD-RESOLUTION implementation.

This shows the significance:

divergence

from the

ideal logic programming

to

its implementation.

Hint: try to write "a search tree like" space for (▲▲) to see

an infinite cycle

We come back later to this isse.