LECTURE 5
LOGIC PROGRAMMING
PREDICATE LOGIC

PROPOSITIONAL LOGIC:

- basic elements are atoms (propositional symbols)
- those atoms are not variable dependent

Many complex ideas cannot be translated into PROPOSITIONA LOGIC.

Consider:

EVERY MAN IS MORTAL
SINCE CONFUCIUS IS A MAN, HE IS MORTAL.

This seems intuitively correct.

If we denote

P: Every man is mortal
Q: Confucius is a man
R: Confucius is mortal

then R cannot be "inferred" within PROPOSITIONAL LOGIC framework.

LECTURE 5
LOGIC PROGRAMMING

More powerful LOGIC is needed:

e.g. **PREDICATE LOGIC**

Again as for PROPOSITIONAL LOGIC:

- language
- semantics
- inference rule(s)

are needed. We introduce

- a first-order language
- generalized semantics
- for PROLOG use, generalized notion of RESOLUTION.

Example 1: Suppose we want to represent:

(i) " x is greater than 3 "

We define <u>a predicate:</u>

greater(X, Y)

to mean

" x is greater than y ".

LECTURE 5
LOGIC PROGRAMMING

Thus (*) is represented by:

greater (X, 3).

This is **syntax** of (*). The meaning i.e. semantics – discussed later, but it is clear that it depends on "**variable substitution**" for X. (X is <u>a variable</u>) 3 is <u>a constant</u> symbol.

(ii) "x loves y"

introduce predicate **love (X, Y)**.

"John loves Mary" reads then:

love (john, mary)

Here **love** is <u>a predicate</u>

john, mary are <u>constant symbols</u>

(iii) we can use also <u>**function symbols**</u>.

plus(X, Y) denotes "x + y"

father (X) to mean "father of x".

Then the sentences:

LECTURE 5
LOGIC PROGRAMMING.

" x+1 is greater than x"

" x is greater than x+1"

" John's father loves John"

read as:

greater(plus(X,1),X).
greater(X, plus(X,1)).
(**) love(father(john), john).

Note at this stage whether they are true or false is irrelevant – this is due to semantics. □

In the above example we have:

a) predicate symbols:

greater/2 (binary – of arity 2)

love/2

b) function symbols:

father/1 (unary – of arity 1)

plus /2

c) constant symbols:

1, 3, john, mary

d) variable symbols:

X, Y

LECTURE 5
LOGIC PROGRAMMING

Arities of predicates & functions:

P/n $\qquad$ f/m

(n, m) - arguments

PROLOG notational convention:

a) variables:

upper-case 1st letter e.g. X, Y.

b) constants:

lower-case 1st letter e.g. john, mary.

c) functions & predicates:

lower-case 1st letter e.g. greater father

Thus for clause

(*) love(father(John), John).

PROLOG treats John as variable & read (*) as

love(father(X), X).

Not what we wanted.

LECTURE 5
LOGIC PROGRAMMING

The **alphabet** of PREDICATE LOGIC:

1. **variable** symbols
2. **constant** symbols
3. **function** symbols
4. **predicate** symbols
5. the **connectives** $\sim, \wedge, \vee, \leftarrow$ and
6. the **quantifiers** $\forall$ & $\exists$ (universal, existential)
7. the **improper** symbols "(", ")", ",", ".", ";"

Note: p/0 **predicate** of arity 0 (**variable** free) is a propositional **symbol**.

Thus 4), 5), & 7) is exactly the alphabet for Propositional Logic. 1-7 is an extention of such alphabet.

LECTURE 5
LOGIC PROGRAMMING

Before introducing SYNTAX:

DEFINITION 1: A **term** is defined as:

1. a variable is a term
2. a constant is a term
3. (inductive part)
   for f/n (function of arity n)
   &
   $t_1, t_2, \ldots, t_n$ terms

$$\Downarrow$$

$f(t_1, t_2, \ldots, t_n)$ is a term.

DEFINITION 2: A term is called **GROUND** if it contains no variables.

Note: the improper symbols force precedence in constructing well-formed formulas. Without parentheses default precedence:

LECTURE 5
LOGIC PROGRAMMING

$\sim, \vee, \exists$

$\vee$

$\wedge$

$\leftarrow$

Example 2: (i) a, $f(a,b)$, $g(a)$, $f(g(a), b)$ etc. GROUND TERMS

(ii) X, $f(X,a)$, $f(X,Y)$, $f(g(a),Y)$, $g(g(X))$ etc

Not GROUND TERMS.

Well-formed formulas (PREDICATE LOGIC):

DEFINITION 3: A well-formed formula is defined as follows:

1. if p is a predicate of arity n & $t_1, t_2, \ldots, t_n$ are terms then

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

is a legal atomic formula (an atom)

2. if f_1 & f_2 are legal formulas then

$\sim(f_i)$ $i=1,2$

$(f_1) \wedge (f_2)$

$(f_1) \vee (f_2)$

are legal.

A.T

$(f_1) \leftarrow (f_2)$

also $(f) \leftarrow$ op. if $f \equiv (x)$

3. if f is a legal formula then

$$\forall x(f) \quad \text{and} \quad \exists x(f)$$

are **legal** (where x is a variable).

Example 3:

Let language $\mathcal{L}$ has **constants:**

bill, mary, jennifer, jack

& **predicate** called

married.

& **variables** X, Y.

Then all formulas:

married (jennifer, jennifer)
married (jennifer, bill)
$\forall x$ married (X, jack)
$\exists x$ married (X, mary)
$\forall x \exists y$ married (X, Y)
married (bill, jack)

are formally **legal** within Predicate Logic framework (SYNTAX). Whether they are true or false – it is addressed by SEMANTICS. □

LECTURE 5
LOGIC PROGRAMMING

Note: again as p/o ≡ propositional symbol.

the alphabet of PROPOSITIONAL LOGIC & set of well-formed formulas (1 & 2 from the last def.) is <u>a special case</u> of PREDICATE LOGIC.

So we have derived

- much stronger symbolic language
- it generalizes the previous one (very primitive).

We now extend the notion of <u>a CLAUSE.</u>

<u>DEFINITION 4</u>: <u>A LITERAL</u> is an atom or its negation i.e

P/n

$p(t_1, t_2, \ldots, t_n)$ or $\sim p(t_1, t_2, \ldots, t_n)$

<u>positive</u> <u>negative</u>

LECTURE 5
LOGIC PROGRAMMING

DEFINITION 5: **A Clause** is a formula

(DNF) $\forall x_1 \forall x_2 \ldots \forall x_n \; A_1 \vee A_2 \vee \ldots \vee A_k \vee \sim B_1 \vee \ldots \vee \sim B_m$

where $\{x_1, x_2, \ldots, x_m\}$ are all variables from **positive atoms** $A_1, A_2, \ldots, A_k, B_1, \ldots, B_m$.

We denote **clause** in **clausal form**

(CF) $\boxed{A_1, A_2, \ldots, A_k \leftarrow B_1, B_2, \ldots, B_m}$

we omit in notations $\forall \ldots$

Similarly, **Horn clauses** are:

either $A_1 \leftarrow B_1, B_2, \ldots, B_m$

or $\leftarrow B_1, B_2, \ldots, B_m$

with **universal quantifiers** in notation omitted.

Now we **define a Logic Program**:

i.e. $\boxed{\text{file.pl}}$

LECTURE 5
LOGIC PROGRAMMING

DEFINITION 6: A program clause: is a clause with precisely one positive literal i.e

$$A \leftarrow B_1, B_2, \ldots, B_k \qquad \text{Rule}$$

OR

$$A \leftarrow \qquad \text{Fact}.$$

A definite PROLOG logic program:

finite set of program clauses.

A definite goal is a negative clause i.e.:

$$\leftarrow B_1, B_2, \ldots, B_n$$

The scope of $\forall x$ in $\forall x F$ is F.
$\exists x$ $\qquad$ $\exists x F$

Bound occurance of x in F is the occurance following the quantifier.
Any other occurance is free.

LECTURE 5
LOGIC PROGRAMMING

$\exists x\ p(x,Y) \wedge q(x)$

free occurance.

bound occurance

$\exists x(p(x,Y) \wedge q(x))$

bound occurances

A closed formula $\equiv$ no free occurances of variable

Abbreviation :

$\forall F \equiv \forall x_1 \forall x_2 \ldots \forall x_n (F)$
$\exists F \equiv \exists x_1 \exists x_2 \ldots \exists x_n (F)$

x_i variables in F.

FOR PREDICATE LOGIC

- we also use REFUTATION
- so $\sim Q$ is to be generalized clause

LECTURE 5
LOGIC PROGRAMMING

Before passing to SEMANTICS in PREDICATE LOGIC, where:

- "general notion" of model, resolution, refutation, logical consequence & satisfiability

is still needed we explain:

why in PROLOG REFUTATION is used, why Forward Chaining Inference (FCI) is abandoned?

a) for FCI:
as "general resolution" produces a clause

$$\Delta \models Q$$

this must be a clause

But clause is $\forall x_1, x_2, \ldots, x_n [Q(x_1, \ldots, x_n)]$
$\equiv A \leftarrow B_1, B_2, \ldots, B_n$

- Q is not a very good Query
- rarely facts/rules happen for all variables involved!
- they happen for some only – how to find them?

LECTURE 5
LOGIC PROGRAMMING

b) for REFUTATION:

if $\boxed{Q = \exists x_1, \dots, x_n (B_1 \wedge B_2 \wedge \dots \wedge B_n)}$

then due to "tautology" validity (shown later)

$$\sim \exists x_1, \dots, x_n (B_1 \wedge \dots \wedge B_n)$$
$$\equiv$$
$$\forall x_1, \dots, x_n \sim B_1 \vee \dots \vee \sim B_n$$

$$\boxed{\sim Q \equiv \forall x_1, \dots, x_n (\sim B_1 \vee \sim B_2 \vee \dots \vee \sim B_n)}$$

↑

a clause (negative)

PROLOG by:

$$\boxed{\Delta \models Q \equiv \square \in (\Delta \cup \{\sim Q\})^*}$$

(i) "searches" for contradiction $\square$

(ii) if yes also finds for which $x_1, \dots, x_n$ $\square$ is obtained!

how? later shown.

LECTURE 5
LOGIC PROGRAMMING

Example 4:

```
p(X):-q(X).
q(a).
```

file.pl

$$\equiv \left.\begin{array}{l} p(x) \leftarrow q(x). \\ q(a). \end{array}\right\} \Delta$$

$$\equiv$$

$$\forall x\,(p(x) \leftarrow q(x)).$$

$$\cancel{\forall x\, q(x).}$$

If we want to ask: $\boxed{\Delta \models p(a)}$?

```
>> p(a).  (+ RTN)
>> yes
```

(so like in Propositional Logic)

But if we want to find the value of x for which p(x) is true:

$$\boxed{Q \equiv \exists x\, p(x)} \quad (**)$$

$$\Downarrow$$

$$\sim Q \equiv \forall x(\leftarrow p(x)).$$

```
>> p(X).  (+ RTN)
X = a.
```

LECTURE 5
LOGIC PROGRAMMING

What happens:

- the system responded that

$$\Delta \cup \{ \forall x \leftarrow p(x) \} = \tilde{\Delta} \leftarrow \text{clauses}$$

is <u>unsatisfiable</u>* for $X = a$.

- still we do not know the algorithm how to find $X = a$?

Query (**) is much more interesting! than (*)

We close the lecture with

2 examples:

"how to translate real-life problems into PREDICATE LOGIC language?"

* ← note that the general notion of unsatisfiability for PREDICATE LOGIC will be introduced next lecture

LECTURE 5
LOGIC PROGRAMMING

Example 5:

Let us consider **English statements:**

(a) every rational number is a real one.

(b) there exists a number that is prime.

(c) for every number x, there exists number y s. that $x < y$.

(d) every irrational number is not a real number.

Note: we do not consider the issue of validity of a)–d)!

Denote:

"x is a prime number" by $P(X)$

"x is a rational number" by $Q(X)$

"x is an irrational number" by $IQ(X)$

"x is a real number" by $R(X)$

"x is less then y" by $Less(X,Y)$

(a) $\forall X\ (R(X) \leftarrow Q(X))$

(b) $\exists X\ P(X)$

(c) $\forall X\ \exists Y\ Less(X,Y)$

(d) $\forall X\ (\sim R(X) \leftarrow IQ(X))$

symbolic representation

LECTURE 5
LOGIC PROGRAMMING

Example 6: Back to the Confucius example

"x is a man" man(X)

"x is mortal" mortal(X)

then

"every man is mortal"

is represented by:

$\forall X\,(\text{mortal}(X) \leftarrow \text{man}(X))$.

&

"Confucius is a man"

man(confucius).

&

"Confucius is mortal"

mortal(confucius).

So the whole problem reads now:

LECTURE 5
LOGIC PROGRAMMING

$$\Delta = \begin{cases} man(confucius). \\ \forall x\,(mortal(X) \leftarrow man(X)). \end{cases}$$

$$Q \equiv mortal(confucius)$$

$$\boxed{\Delta \models Q} \quad ?$$

PROLOG PROGRAM:

```
man(confucius).
mortal(X) :- man(X).
```

file.pl

and QUERY

```
>> mortal(confucius).   (+RTN)

   yes
>>
```

Next lecture: SEMANTICS FOR PREDICATE LOGIC