LECTURE 9

LOGIC PROGRAMMING

SUBSTITUTION

We concentrate more now on some aspects of logic programming.

PROLOG "matches" a body atom in the current goal & the head of the program clause.

Example1: Consider clauses

C_1: $P(x) \vee Q(x)$

C_2: $\sim P(f(x)) \vee R(x)$

There is no literal in C_1 that is complementary to any literal in C_2

However, if we substitute $f(a)$ for X in C_1 & a for X in C_2:

LECTURE 9
LOGIC PROGRAMMING

C_1': $P(f(a)) \vee Q(f(a))$.
C_2': $\sim P(f(a)) \vee R(a)$.

} GROUND INSTANCES

Then we can resolve C_1' & C_2' to:

C_3': $Q(f(a)) \vee R(a)$.

More generally:

taking into account that

C_1: $P(X) \vee Q(X) \equiv P(X') \vee Q(X')$. *

if we substitute for X' with $f(X)$ in C_1 we obtain:

C_1^* $P(f(X)) \vee Q(f(X))$.
C_2 $\sim P(f(X)) \vee R(X)$.

new resolvent

C_3: $Q(f(X)) \vee R(X)$

* Later we define formally SUBSTITUTION where $X \to f(X)$ is illegal. □

Note that C_3' is an instance of C_3.

By substituting appropriate terms for variables in C_1 & C_2 one can "generalize" **RESOLUTION**

↖ so far defined for
a) PROPOSITIONAL LOGIC
b) ground instances of clauses in PREDICATE LOGIC.

Furthermore:

clause C_3 is the "most general clause".
- the other are special cases.

DEFINITION 1:

The process of matching same terms in two clauses (by SUBSTITUTION) is called

UNIFICATION

of two terms.

LECTURE 9
LOGIC PROGRAMMING

UNIFICATION ALGORITHM:

is the mechanism by which variables are bound & atoms are cancelled out when resolving two clauses.

DEFINITION 2: **A substitution** Θ is the finite set of the form

$$\{v_1/t_1, \ldots, v_n/t_n\}, \qquad (\blacktriangle)$$

where each v_i is a variable, each t_i is a term, & the variables $v_1, v_2, \ldots, v_n$ are distinct. Each element v_i/t_i of $(\blacktriangle)$ is called **binding** for v_i.

The substitution is called:

ground

if all t_i are ground terms;

variable-pure

if all t_i are variables.

LECTURE 9
LOGIC PROGRAMMING

Note: if v_i/t_i then t_i **cannot contain** v_i

In EX. 1 substitution was ground (the 1st one) & variable-pure (the 2nd one).

There are of course substitutions which are

neither **ground** nor **variable-pure**.

DEFINITION 3: **An expression** is either a term, a literal, or a conjunction or disjunction of literals.

A simple expression is either a term or an atom.

expression : term $\leftrightarrow$ variable, constant, f(term)

literal p( ) or ~p( )
+ conjunction & disjunctions

LECTURE 9
LOGIC PROGRAMMING

simple expression:

term → constant
term → variable
term → f(term)

atom p( )

DEFINITION 4:

Let

$$\Theta = \{ v_1/t_1, v_2/t_2, \ldots, v_n/t_n \}$$

be a substitution & E be an expression. Then $E\Theta$, the instance of E by Θ is the expression obtained from E by simultaneously replacing all occurances of the variable v_i in E by term t_i

If $E\Theta$ is ground the $E\Theta$ is called a ground instance of E.

Let us look at examples:

LECTURE 9
LOGIC PROGRAMMING

Example 2:

a) if $E = parent(X, Y).$ & $\theta = \{X/pam, Y/bob\}$

then $E\theta = parent(pam, bob)$

↑ GROUND INSTANCE

b) if $E = p(X, Y, f(a))$ & $\theta = \{X/b, Y/X\}$

then $E\theta = p(b, X, f(a)).$

↑ neither ground nor variable – pure

□

If $S = \{E_1, E_2, \ldots, E_n\}$ is a finite set of expressions then

$$S\theta = \{E_1\theta, E_2\theta, \ldots, E_n\theta\}.$$

In PROLOG $n = 2$:

i.e. $E_1\theta$

↑ an instance of the head of the program clause

$E_2\theta$

↑ left-most atom in the current goal (a negative clause)

LECTURE 9
LOGIC PROGRAMMING

Our interest is now
a composition of two substitutions:

Example 3:

Let $\Theta = \{ X/f(Y), Y/Z \}$

$\alpha = \{ X/a, Y/b, Z/Y \}$.

We want to find $\Theta \circ \alpha = \Theta\alpha$

(1) (2)

(1) this is first

(2) this is second

$$\left.\begin{array}{l} X \to f(Y) \to f(b) \\ Y \to Z \to Y \end{array}\right\} \quad \begin{array}{l} X \to f(b) \\ Y \to Y \\ Z \to Y \end{array} \text{ (the same!)}$$

So finally

$$\Theta \circ \alpha = \{ X/f(b), Z/Y \} \quad \square$$

Note that if

variable $\to$ the_same_variable

we don't treat this as a substitution.

Even more disallowed. variable $\to$ f(the_same_variable)

LECTURE 9
LOGIC PROGRAMMING

DEFINITION 5:

Let

$$\theta = \{ u_1/s_1, \dots, u_m/s_m \}$$

and

$$\alpha = \{ v_1/t_1, \dots, v_n/t_n \}.$$

Then the <u>composition</u> $\theta\alpha$ of two substitutions θ & α is the substitution obtained from the set

$$\{ u_1/s_1\alpha, u_2/s_2\alpha, \dots, u_m/s_m\alpha, v_1/t_1, \dots, v_n/t_n \}$$

(where $s_i\alpha$ is a substitution of s_i by α) by deleting any binding $u_i/s_i\alpha$ for which $s_i\alpha = u_i$ and deleting any binding v_j/t_j for which $v_j \in \{ u_1, \dots, u_m \}$.

REMARK:

- deletion of $s_i\alpha = u_i$ is due to the fact that variable cannot be mapped to the same variable.

•• deletion of v_j/t_j is due to the fact that if $v_i \notin \{u_1, \dots, u_m\}$ then it was not altered by the previous substitution Θ. If $v_i \in \{u_1, \dots, u_m\}$ then all changes v_i are grasped by (*).

Example 4:

If we look to the last Ex 3. then

$$\Theta a = \{ X/f(a), Y/Y, \underbrace{X/a, Y/b, Z/Y}_{v_i/t_i} \} \quad (\blacktriangle)$$

& we throw out the terms:

Y/Y – due to •

X/a, Y/b – due to ••

leaving ($\blacktriangle$) as

$$\Theta a = \{ X/f(b), Z/Y \}.$$

The same as in Ex. 3. □

LECTURE 9
LOGIC PROGRAMMING

The substitution given by an empty set is called **IDENTITY** & is denoted by ϵ.

This is intuitive: empty set means no variables are changed.

See Lloyd §4. p. 21.

Lemma 1: Let $\theta, \alpha,$ & γ be substitutions.

Then

1. $\theta\epsilon = \epsilon\theta = \theta$
2. $(E\theta)\alpha = E(\theta\alpha)$ for each Expression
3. $(\theta\alpha)\gamma = \theta(\alpha\gamma)$.

Note: $\alpha\theta \neq \theta\alpha$.

DEFINITION 6:

Let E & F be expressions. Then F & E are called **VARIANTS** of each other if there are substitutions θ & α s.t.

$E = F\theta$ & $F = E\alpha$

LECTURE 9
LOGIC PROGRAMMING

Example 5:

(a) $p(f(X,Y), g(Z), a)$ $\qquad$ $p(f(Y,X), g(U), a)$

$\qquad\qquad\quad\;\; \equiv \qquad\qquad\qquad\quad \equiv$

$\qquad\qquad\quad\;\; E \qquad\qquad\qquad\quad F$

E is a variant of F i.e $E = F\theta_2$

F is a variant of E i.e $F = E\theta_1$

where

$\theta_1 = \{X/Y, Y/X, Z/U\}$

$\theta_2 = \{Y/X, X/Y, U/Z\}$.

(b) however $p(X,Y)$ is not a variant of $p(X,X)$

$\qquad\qquad\quad\;\; \equiv \qquad\qquad\quad\;\; \equiv$

$\qquad\qquad\quad\;\; E \qquad\qquad\quad\;\; F.$

$E \neq F a_1$

$F = E a_2$ $\qquad$ □

DEFINITION 7:

Let E be an expression a let V be the set of variables occuring in E. Then

LECTURE 9
LOGIC PROGRAMMING

a RENAMING SUBSTITUTION

(*) $\{X_1/Y_1, X_2/Y_2, \ldots, X_n/Y_n\}$

is <u>a variable-pure</u> substitution (*) such that $\{X_1, X_2, \ldots, X_n\} \subseteq V$, the Y_i are distinct & $(V \setminus \{X_1, X_2, \ldots, X_n\}) \cap \{Y_1, \ldots Y_n\} = \emptyset$.

The last condition assures that <u>new variables</u> do not get mixed up with the <u>old ones</u>.

It is clear that if E differs from F via <u>renaming substitution</u> then E is <u>variant</u> of F & <u>vice versa</u>.

The converse is also true:

Lloyd pp. 22

Lemma 2: Let E & F be variant of each other. Then there exists RENAMING SUBSTITUTIONS θ & σ: $E = F\sigma$ & $F = E\theta$.

LECTURE 9
LOGIC PROGRAMMING

We are interested in substitutions that make each expression the same.

$$S = \{E_1, E_2, \ldots, E_n\}$$

↓ how to find it?

$$S\Theta = \{E_1\Theta, E_2\Theta, \ldots E_n\Theta\}$$

$$\Downarrow$$

$$S\Theta = \{E_i\Theta\}$$

i.e.: $E_1\Theta = E_2\Theta = \ldots = E_n\Theta$.

PROLOG: $n = 2$ as

- $i = 1$ E_1 is a head of program clause
- $i = 2$ E_2 is the left-most atom in the current program goal.

Such a substitution is called:

A UNIFIER

LECTURE 9
LOGIC PROGRAMMING

DEFINITION 8:

Let S be a finite set of simple expressions. A substitution θ is called a unifier for S if $S\theta$ contains only one expression.

A unifier θ for S is called a most general unifier (mgu) if for every unifier α of S there is a substitution γ such that: $\boxed{\alpha = \theta\gamma}$.

- a unifier: substitution that makes $S\theta$ a singleton
- a MGU: is "a minimal variable substitution" via which we can get all other unifiers

LECTURE 9
LOGIC PROGRAMMING

Example 6:

Let $S = \{ p(X, f(X, g(Y))), p(X, Z) \}$.

Then $\alpha = \{X/a, Y/b, Z/f(a, g(b))\}$

is <u>a unifier</u> as

$$S\alpha = \{ p(a, f(a, g(b))) \}.$$

But for another <u>unifer</u>

$$\Theta = \{ Z/f(X, g(Y)) \}$$

for which

$$S\Theta = \{ p(X, f(X, g(Y))) \}$$

we <u>cannot have</u>

$$\Theta = \alpha \cdot \gamma.$$

But clearly Θ is <u>a MGU</u>. Notice:

$$\begin{aligned} \alpha &= \Theta \circ \{X/a, Y/b\} \\ &= \{Z/f(X, g(Y))\} \circ \{X/a, Y/b\} \\ &= \{X/a, Y/b, Z/f(a, g(b))\}. \quad \text{YES} \end{aligned}$$

Note that if θ_1 & θ_2 are MGUs

$$\theta_2 = \theta_1 \tilde{\tau} \Rightarrow \theta_2 = \theta_2 \alpha \tilde{\tau}$$
$$\theta_1 = \theta_2 \alpha$$

So $\alpha \tilde{\tau} = \epsilon$.
α & τ are renaming substitutions.

Note now

$$\tilde{\theta} = \{X/X_1, Y/Y_1, Z/f(X_1, g(Y_1))\}$$

is also another MGU. Indeed

$$S\tilde{\theta} = \{p(X_1, f(X_1, g(Y_1))), p(X_1, f(X_1, g(Y_1)))\}$$
$$= \{p(X_1, f(X_1, g(Y_1)))\}. \leftarrow \text{so a unifier}$$
$$\tilde{\theta} = \theta \circ \gamma = \{Z/f(X, g(Y))\} \circ \{X/X_1, Y/Y_1\}$$
$$= \{X/X_1, Y/Y_1, Z/f(X_1, g(Y_1))\}$$

as θ is a most general unifier

On the other hand:

$$\Theta = \{X/X_1, Y/Y_1, Z/f(X_1, g(Y_1))\} \circ \{X_1/X, Y_1/Y\}$$

$$= \{X/X, Y/Y, Z/f(X, g(Y))\}$$

$$= \{Z/f(X, g(Y))\} \quad \text{yes}$$

renaming substitution

as $\tilde{\Theta}$ is a most general unifier.

Clearly Θ & $\tilde{\Theta}$ differ by renaming substitution. □

Warning:

$E = p(f(X), Y)$ & $\Theta = \{X/f(Y), Y/f(a)\}$

$E\Theta = p(f(f(Y)), f(a))$

not $E\Theta = p(f(f(f(a))), f(a))$.

Some PROLOG examples:

LECTURE 9
LOGIC PROGRAMMING

Example 7:

Arithmetic in pure LOGIC PROGRAMMING for IN

0 ⟶ zero

1 ⟶ S(zero) successor of zero

2 ⟶ S(S(zero)) —//—

⋮

```
isnumber(zero).
isnumber(S(X)) :- isnumber(X).
```
≡Δ

declarative description of what it means to be a number

" 0 is a number "

" x+1 is a number if x is a number "

>> isnumber(zero).

yes (RTN)

>> isnumber(S(S(S(zero)))).

yes (RTN)

>> isnumber(s(s(2))).

no (RTN)

>> isnumber(X).

↖ "what are the numbers"

≡ $Q \equiv \exists X$ isnumber(X).

X = zero ;

X = s(zero);

X = s(s(zero));

X = s(s(s(zero))) (RTN)

yes

We shall show later the corresponding SLD-trees for Δ & different QUERIES.

□

Example 8:

another mathematical operation:

" 0 is less than each $n \in \mathbb{N}$ "

" if $x \leq y \Rightarrow x+1 \leq y+1$ "

LECTURE 9
LOGIC PROGRAMMING

```
lessthanequal(zero, X):- isnumber(X).
lessthanequal(S(X), S(Y)):- lessthanequal(X,Y).
```

|||

Δ

>> lessthanequal(S(S(zero)), S(S(S(zero)))).

yes (RTN) " is $2 \leq 3$? "

>> lessthanequal(S(S(zero)), S(zero)).

" is $2 \leq 1$? "

No

>> lessthanequal(zero, X).

" $Q \equiv \exists x$ lessthan(S(zero), X). "

$\exists x \; 1 \leq x$?

X = S(zero); $1 \leq 1$

X = S(S(zero)) (RTN) $1 \leq 2$

>>

□

Again one can construct more such examples.