LECTURE 10
LOGIC PROGRAMMING
UNIFICATION ALGORITHM

We present an algorithm:

- to unify the head of some (i.e. E_1) program clause with the first left-most literal in the current goal clause (i.e. E_2)
- it either:
  - (i) outputs **MGU** if $S = \{E_1, E_2\}$ is unifiable
  - (ii) or it reports the fact that a set **is not unifiable**.

Example 1:

(i) $S = \{p(f(X), a), p(Y, f(W))\}$ is **not unifiable** as the 2nd arguments a & $f(W)$ cannot be unified: we need a mapping variable $\longmapsto$ term.

(ii) $S = \{p(X), q(X)\}$.

cannot be unified as for the PREDICATE LOGIC language predicates are "not free variables".

(iii) $S = \{p(f(X), Z), p(Y, a)\}$

is unifiable: i.e. MGU reads:

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

□.

We need some auxilliary notions:

DEFINITION 1:

Let S be a set of simple expressions. The **disagreement set of S** is defined

- locate the leftmost symbol position at which not all expressions in S have the same symbol.
- extract from each expression in S the subexpression beginning

at that symbol position. The set of all such subexpressions forms the disagreement set.

The above definition is general (i.e for arbitrary n).

Example 2:

(i) $S = \{ p(f(X), \underline{h(Y)}, a), p(f(X), \underline{Z}, a), p(f(X), \underline{h(Y)}, b) \}$

then the disagreement set is $\{h(Y), Z\}$ = D_S

(ii) $S = \{P(X, \underline{f(Y,Z)}), P(X, \underline{a}), P(X, \underline{g(h(K(X)))})\}$

then the disagreement set is

$D_S = \{ f(Y,Z), a, g(h(K(X))) \}$

(iii)

$S = \{p(x), q(x)\} \Rightarrow D_S = \{p(x), q(x)\}.$

$S = \{p(x), p(a)\} \Rightarrow D_S = \{x, a\}$

$S = \{p(x), p(x)\} \Rightarrow D_S = \emptyset.$ □

UNIFICATION ALGORITHM – UA

1. STEP: Put $k=0$ & $\alpha_0 = \epsilon$

2. STEP:
   IF $S\alpha_k$ is a singleton then STOP
   OTHERWISE find th disagreement set D_k of $S\alpha_k$

3. STEP:
   IF there exists v & t in D_k such that v is a variable that does not appear in t, then put
   $$\alpha_{k+1} = \alpha_k \circ \{v/t\}$$
   increment k and go to step 2
   OTHERWISE STOP:
   S is not unifiable

LECTURE 10.

LOGIC PROGRAMMING

REMARK: it is clear that UA terminates because S contains only finite number of variables & each application of STEP 3 eliminates 1 variable.

Example 3:

(i) Let $S = \{ p(X, X), p(Y, f(Y)) \}$

a) $\alpha_0 = \epsilon$

$S\alpha_0 = S \neq$ singleton

$D_0 = \{X, Y\}$.

b) $\alpha_1 = \epsilon \circ \{X/Y\} = \{X/Y\}$

$S\alpha_1 = \{ p(Y, Y), p(Y, f(Y)) \}$

$\neq$ singleton

$D_1 = \{Y, f(Y)\}$.

& UA fails here to find MGU as we cannot find in D_1 a variable v & a term t not involving v. So S is <u>not unifiable</u>.

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

a) $\alpha_0 = \epsilon$
$S\alpha_0 = S \neq$ singleton
$D_0 = \{f(a), Y\}$

b) $\alpha_1 = \epsilon \circ \{Y/f(a)\} = \{Y/f(a)\}$.
$S\alpha_1 = \{p(f(a), g(X)), p(f(a), f(a))\}$
$D_1 = \{g(X), f(a)\}$.

UA fails: no free variable available.
S **is not unifiable.**

(iii) Let $S = \{p(a, X, h(g(Z))), p(Z, h(Y), h(Y))\}$

a) $\alpha_0 = \epsilon$
$S\alpha_0 = S \neq$ singleton
$D_0 = \{a, Z\}$

b) $\alpha_1 = \epsilon \circ \{Z/a\} = \{Z/a\}$
$S\alpha_1 = \{p(a, X, h(g(a))), p(a, h(Y), h(Y))\}$
$\neq$ singleton
$D_1 = \{X, h(Y)\}$

c) $\alpha_2 = \alpha_1 \circ \{X/h(Y)\} = \{Z/a, X/h(Y)\}$
$S\alpha_2 = \{p(a, h(Y), h(g(a))), p(a, h(Y), h(Y))\}$
$\neq$ singleton $\quad D_2 = \{Y, g(a)\}$.

d) $a_3 = a_2 \circ \{Y/g(a)\}$

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

$Sa_3 = \{p(a, h(g(a)), h(g(a)))\}$

$=$ singleton

so S is unifiable & MGU $\equiv a_3$.

□

THEOREM 1: UNIFICATION TH.
see Lloyd pp. 25.

Let S be a finite set of simple expressions. If S is unifiable, then the UA terminates and gives an MGU for S. If S is not unifiable, the the UA terminates & reports this fact.

Note: in STEP 3 of UA we check whether v OCCURS in term t.

This is called **OCCUR CHECK**

LECTURE 10
LOGIC PROGRAMMING

If we admit substitutions

$$V \rightarrow t(V) = t(t(V)) = t(t(t(V))) \ldots$$

so we create an expression with infinite number of symbols!

↓

this would lead us out of the PREDICATE LOGIC LANGUAGE!

If implemented it would also recursively build infinite term.

OCCUR CHECK CAN BE VERY INEFFICIENT!

Example 4:

Let $S = \{p(x_1, x_2, \ldots, x_n), p(f(x_0, x_0), \ldots, f(x_{n-1}, x_{n-1}))\}$

$D_0 = \{x_1, f(x_0, x_0)\}$ $\quad \alpha_1 = \{x_2 / f(x_0, x_0)\}$

$D_1 = \{x_2, f(f(x_0, x_0), f(f(x_0, x_0))\}$

$\alpha_2 = \{x_1 / f(x_0, x_0)\} \circ \{x_2 / f(f(x_0, x_0), f(x_0, x_0))\}$

$= \{x_1 / f(x_0, x_0), x_2 / f(f(x_0, x_0), f(x_0, x_0))\}$.

& so on.

Easy to see that disagreement set D_k contains a term with 2^{k+1} occurances of X_0 & hence the OCCUR CHECK requires

EXPONENTIAL TIME!

□

Because of above:

MOST PROLOG INTERPRETERS OMIT "OCCUR CHECK"

- execution – the substitution $x \to f(x)$ will be accepted (formally)
- outputting – the program will crash when asked to print $X \to f(X)$

LECTURE 10
LOGIC PROGRAMMING

In theoretical LOGIC PROGRAMMING we LOSE SOUNDNESS as admitting ∞-terms. Indeed:

Example 5:

$$\Delta = \begin{cases} test \leftarrow p(X,X). \\ p(X, f(X)). \end{cases}$$

QUERY: $\boxed{\Delta \models test}$?

Without the occur check

a) PROLOG REPLIES "Yes"
It does not crash as test do not contain variable.
With any query of type $\exists x\, Q(x)$ it would crash!

b) but

$\Delta \not\models test$ indeed

LECTURE 10
LOGIC PROGRAMMING

Consider the $\mathcal{I}$:

a) D is arbitrary

b) $P/2 \to P'/2$ $\quad p'(x,y)$ is false iff $x = y$
(e.g. $p' \equiv ' \neq '$)

and assume $f \to f'$ such that

$f(X) \neq X$.

c) $test \to test' \equiv false$

Then $\mathcal{I}$ is a model for Δ as both formulas from Δ are true under $\mathcal{I}$. But test is false under $\mathcal{I}$. So $\Delta \not\models test$ □

Note that if

$\Delta = \{clauses, p(X)\}$ $\quad \leftarrow p(X).$ current goal

so $S' = \{p(X), p(X)\}$ (■)

↑ ↑ variables are the same!

only apparently But there is no OCCUR CHECK violation here.

In fact

$$\Delta = \{\text{clauses}, \forall x\, p(x)\}$$

$$\Vert\!\Vert$$

$$\{\text{clauses}, \forall x'\, p(x')\}.$$

Thus the disagreement set for

$$S = \{p(x'), p(x)\}$$

is $D = \{x', x\}$

& consequently substitutions

$$x \to x' \quad \text{or} \quad x \to x'$$

are legal.

Therefore $S = \{p(x), p(f(x))\}$ in the context of the above (■) can be unified!

However consider

$$S = \{ p(x, X),\ p(x, f(x)) \} \qquad (\blacktriangle)$$

$$\equiv$$

$$\{ p(x, X),\ p(x', f(x')) \}$$

$$\leftarrow \forall x'\ p(x', f(x')) \quad \text{from}$$

So the 1st disagreement set is

$$D_0 = \{ X, x' \} \Rightarrow a_1 = \{ X / x' \}$$

Hence

$$S_1 = \{ p(x', x'),\ p(x', f(x')) \}$$

Now we cannot change x' to x'' they are bound! More

$$D_1 = \{ x', f(x') \}$$

& therefore as $f(x')$ contains x' ($\blacktriangle$) is not unifiable.

LECTURE 10.
LOGIC PROGRAMMING

Example 6:

For multiplication:

"$0 \cdot X = 0$"

"$X \cdot Y = Q$ & $Y + Q = Z \Rightarrow (X+1) \cdot Y = Z$"

Assume that plus(X, Y, Z) is defined. (see next lecture!)

```
times(zero, X, zero).
times(s(X), Y, Z) :- times(X, Y, Q),
                     plus(Y, Q, Z).
```

>> times(s(s(zero)), s(s(zero)), X).

"$2 \cdot 2 = ?$"

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

yes

>> times(s(s(zero)), X, s(s(s(s(zero))))).

"$2 \cdot ? = 4$"

X = s(s(zero))

yes.

this is when there is a divisor!
But if not, we have <u>an infinite loop</u>

>> times(s(s(zero)), X, s(s(s(s(zero))))).
"2. ? = 3"

(*) ! out of local stack during execution
[execution aborted]. □

One has to be careful when defining the predicates.

If asked to do

predicate(ground1, X)

works that does not mean that

predicate(X, ground2).

To see the reason for (*) in the last example one has to draw the SLD-TREE for Δ & QUERY.