Unification

Two atomic formulae, A and B, with distinct variables, unify iff they can be made identical by replacing their variables by other terms.

- `loves(bob,Y)` unifies with `loves(bob,sue)` by replacing `Y` by `sue`.
- `loves(bob,Y)` unifies with `loves(X,santa)` by replacing `Y` by `santa` and `X` by `bob`. Both formulas become

    `loves(bob,santa)`

Formally, we used a *substitution* {Y \ santa, X \ bob} which is called a *unifier* of `loves(bob,Y)` and `loves(X,santa)`.

- Note that `loves(bob,X)` does *not* unify with `loves(tony,Y)`: no substitution for `X,Y` can make the two formulae syntactically equal.

More examples

`a(X,X)` unifies with `a(b,b)` and `a(c,c)` but *not* with `a(b,c)` because each occurrence of `X` must be replaced by the `same` term.

In `love(sue,tony)`, `sue` and `tony` are *constant symbols* (very simple data structures). We can construct more complex data structures using *function terms*, e.g.

`owns(john, car(blue,corvette))`

Database:

```
owns(john, car(red,corvette))
owns(john, cat(black,siamese,sylvester))
owns(elvis,copyright(song,"jailhouse rock"))
owns(tolstoy,copyright(book,"war and peace"))
owns(elvis,car(red,cadillac))
```


Yet More Examples

Queries:

- "retrieve everything that john owns"
  `owns(john,X)`

answers: `X = car(red,corvette)`
`X = cat(black,siamese,sylvester)`

- "retrieve the color and make of john's car"
  `owns(john,car(Color,Make))`
  answer: `Color = red`
  `Make = corvette`
- "who owns the copyright on the song "jailhouse rock"?"
  `owns(Who,copyright(song,"jailhouse rock"))`
  answers: `Who = elvis`

Examples (Cont'd)

Note that variables are *existentially* quantified:

$\exists$`Make, owns(Who,car(red,Make))`
answer: `Who = john`
`Who = elvis`

`Make` is *bound* to the quantifier, whereas `Who` is free. These are included in answers to queries.

- "Who owns a copyright?"
  $\exists$`X,Y owns(Who,copyright(X,Y))`

answers: `Who = elvis`
`Who = tolstoy`

In Prolog, existentially quantified variable is denoted by `_`:

`owns(Who, _)` means $\exists$`X, owns(Who,X)`
`owns(Who, copyright(_,_))` means
$\exists$`X,Y owns(Who,copyright(X,Y))`

Examples (Cont'd)

`owns(Who,car(red,Make))` unifies with `owns(elvis, car(red,cadillac))` using substitution

{Who \ elvis, Make \ cadillac}.

`owns(Who,car(red,Make))` also unifies with `owns(john, car(red,corvette))` using substitution

{Who \ john, Make \ corvette}.

Substitution

A substitution is a set

$$\{v_1 \setminus t_1, \ldots, v_n \setminus t_n\}$$

where v_is are distinct variable names
t_is are terms which do not use <u>any</u> of the v_js, $j = 1, \ldots, n$.

Positive Examples:

$\{X\backslash a,\ Y\backslash b,\ Z\backslash f(a,b)\}$
$\{X\backslash W,\ Y\backslash f(W,V,a),\ Z\backslash W\}$

Negative Examples:

$\{f(X)\backslash a\}$
$\{X\backslash a,\ X\backslash b\}$
$\{X\backslash f(X)\}$
$\{X\backslash f(Y),\ Y\backslash g(q)\}$

Applying a Substitution

Notation: t is a term
σ is a substitution

then $t\sigma$ is the term resulting from performing the substitution σ on t.

Examples:

$$\begin{array}{ll} X\{X\backslash a\} & = \\ f(X)\{X\backslash a\} & = \end{array}$$

Let $t = f(a, h(Y,b), X)$

$$\begin{array}{ll} t\{X\backslash b\} & = \\ t\{X\backslash b,\ Y\backslash f(Z)\} & = \\ t\{X\backslash Z,\ Y\backslash f(Z,a)\} & = \\ t\{W\backslash Z\} & = \end{array}$$

Note:

- σ need not include all variables in t.
- σ can include variables not in t.

Composing Substitutions

We can compose substitutions (barring variable name clashes).

If σ and θ are substitutions, then so is $\sigma \circ \theta$.

DEFN: For any term t, $t[\sigma \circ \theta] = (t\sigma)\theta$.

For example,

$$\begin{array}{ll} & f(X)\ \ [\{X\backslash Z\}\ \circ\ \{Z\backslash a\}] \\ = & (f(X)\ \{X\backslash Z\})\ \ \{Z\backslash a\} \\ = & f(Z)\ \ \{Z\backslash a\} \\ = & f(a) \end{array}$$

Also,

$$\begin{array}{llll} \{X\backslash a\} & \circ & \{Y\backslash b\} & = \\ \{X\backslash Z\} & \circ & \{Z\backslash a\} & = \\ \{X\backslash a\} & \circ & \{X\backslash b\} & = \end{array}$$

Unifiers

DEFN: t_1 and t_2 are *unified* by substitution σ
iff $t_1\sigma = t_2\sigma$.

If so, σ is called a *unifier*,
and t_1 and t_2 are said to be unifyable.

Examples:

t_1	t_2	unifier σ	$t_1\sigma$ $(= t_2\sigma)$
f(a)	f(X)		
f(X,b)	f(a,Y)		
f(b,c)	f(b,c)		
f(b,c)	g(X,c)		
f(a,b)	f(c,d)		
f(X,Y)	f(W,Z)		
f(a,b)	f(X,X)		
f(X,a)	f(Y,Y)		
f(g(U),d)	f(X,U)		
f(X)	f(g(Y))		
f(X)	f(Y)		
f(X)	f(g(X))		
f(X,g(X))	f(Y,Y)		

Multiple Unifiers

Assume $t_1 = f(X)$ and $t_2 = f(Y)$.

σ	$t_1\sigma$ $(= t_2\sigma)$
{ X \ Y}	
{ Y \ X}	
{ Y \ Z, X \ Z}	
{ Y \ a, X \ a}	
{ Y \ g(b,Z), X \ g(b,Z)}	
{ X \ Y, W \ h(q,Z)}	

Q: Is there a "best" unifier?

Wish list:

- no irrelevant constants
  {Y \ X} preferred over {X \ a, Y \ a}
- no irrelevant bindings
  {Y \ X} preferred over {Y \ X, W \ h(q,Z)}

In Search of the "Best" Unifier

Assume σ_1 and σ_2 are both unifiers for some pair of terms.

- Suppose σ_1 has a constant where σ_2 has a variable. Then $\exists$ subst. θ such that $\sigma_2 \circ \theta = \sigma_1$
  e.g. $\sigma_1 = \{X\backslash a, Y\backslash a\}$ $\sigma_2 = \{X\backslash Z, Y\backslash Z\}$
  then $\theta = \{Z\backslash a\}$
  and $\{X\backslash Z, Y\backslash Z\} \circ \{Z\backslash a\} = \{X\backslash a, Y\backslash a\}$
- Suppose σ_1 has an extra binding over σ_2. Then $\exists$ subst. θ such that $\sigma_2 \circ \theta = \sigma_1$
  e.g. $\sigma_1 = \{X\backslash a, W\backslash b\}$ $\sigma_2 = \{X\backslash a\}$
  then $\theta = \{W\backslash b\}$
  and $\{X\backslash a\} \circ \{W\backslash b\} = \{X\backslash a, W\backslash b\}$

Moral:
inferior unifier = better unifier composed with another subst.

"Most General Unifier" (MGU)

DEFN: σ is an mgu for t_1 and t_2 iff
(1) σ unifies both t_1 and t_2, and
(2) $\forall$ subst. μ that unify t_1 and t_2,
$\exists$ subst. θ such that $\sigma \circ \theta = \mu$.

i.e. $t_1\mu = (t_1\sigma)\theta$
and $t_2\mu = (t_2\sigma)\theta$.

Basic idea: An mgu has

- no unnec./irrelevant constants
  (uses constants only when necessary)
- no extraneous bindings

An Example

$$\boxed{t_1 = f(X) \quad t_2 = f(Y)}$$

then $\sigma = \{X \backslash Y\}$ is an mgu.

Why?

Consider $\mu = \{X \backslash a, Y \backslash a\}$, another unifier for t_1 and t_2.

Then $\exists\ \theta = \{Y \backslash a\}$ such that $\sigma \circ \theta = \mu$.

Note also that $\{Y \backslash X\}$ and $\{Y \backslash Z, X \backslash Z\}$ are also mgus for t_1 and t_2.

Another Example

$$t_1 = f(W,\ g(Z),\ Z)$$

$$t_2 = f(X,\ Y,\ h(X))$$

$$\text{Need} \quad \begin{array}{ccc} W & \longleftrightarrow & X \\ g(Z) & \longleftrightarrow & Y \\ Z & \longleftrightarrow & h(X) \end{array}$$

so mgu $\sigma =$

More MGU Examples

t_1	t_2	MGU σ
f(X,a)	f(a,Y)	
f(h(X,a),b)	f(h(g(a,b),Y),b)	
g(a,W,h(X))	g(Y,f(Y,Z),Z)	
f(X,g(X),Z)	f(Z,Y,h(Y))	
f(X,h(b,X))	f(g(P,a),h(b,g(Q,Q)))	

Prolog Terms

BNF:

<term>	::=	<constant> | <variable> | <functor>
<constant>	::=	<atom starting with l.case letter>
<variable>	::=	<atom starting with u.case letter>
<functor>	::=	<constant> '[' (<term> {, <term> }) ']'

Examples:

`a1   b   fred   married`	$\rightarrow$ constants
`X   Fred   Yc3`	$\rightarrow$ variables
`found   married(fred)   f(g(Y),X)`	$\rightarrow$ functors

DEFN: a *ground term* is a term with no variables.

f(q)   g(f(w),w1(b,c))   are ground terms.
f(A)   g(f(w),w1(B,c))   are not.

Prolog's Syntax

We would like to have full first-order logic ($\wedge, \vee, \neg, \Rightarrow, \forall, \exists$), but we settle for a subset:
$\longrightarrow$ positive Horn clauses.

$\langle$ *Horn clause* $\rangle$ ::= $\langle$ *term* $\rangle$.
| $\langle$ *term* $\rangle$: − $\langle$ *term* $\rangle$ {, $\langle$ *term* $\rangle$ }.

Examples:

1. `father(john,sue).`
2. `ancestor(odin,X).`
3. `parent(X,Y) :- mother(X,Y).`
4. `grandparent(X,Y) :- parent(X,Z), parent(Z,Y).`

Reading Prolog as Predicate Calculus

"," means AND / **&**
": −" means IMPLIES / $\Leftarrow$

Variables are assumed to be quantified by $\forall$.

Interpretations of examples:

1. `father(john,sue)`
2. $\forall$ `X . ancestor(odin,X)`
3. $\forall$ `X, Y . mother(X,Y)` $\Rightarrow$ `parent(X,Y)`
4. $\forall$ `X, Y, Z . ( parent(X,Z)` $\wedge$ `parent(Z,Y) )`
   $\Rightarrow$ `grandparent(X,Y)`

food for slide eater

food for slide eater