Unification in Prolog

Following statement pass in query.

* Qury: *f*(a, b) = *f*(X, Y).

**X** = a,

**Y** = b

* Qury: (*q*(Y,*g*(a,a))= *q*(*g*(X,X),Y)).

**X** = a,

**Y** = *g(*a,a)

* Qury: likes(X, [food, drink]) = likes(john, [Y, Z]).

X = john, Y = food, Z = drink

* Qury: f(X, g(Y, Z)) = f(1, g(2, 3)).

# X = 1, Y = 2, Z = 3

* Qury: f(X, g(Y)) = f(1, h(2)).

Faulse.

* Qury: p(X, q(Y, f(Z))) = p(a, q(b, f(c))).

**X** = a,

**Y** = b,

**Z** = c

Qury: 2\*3+4 = X+Y. X = 2\*3, Y = 4

Code: X = [1, 2, 3].

Qury: [X, Y | Z] = [1, 2, 3, 4].

Result: **X** = 1,

**Y** = 2,

# **Z** = [3, 4]

Other Codes.

?- a = a. yes

?- a = b. no

?- location(apple, kitchen) = location(apple, kitchen). yes

?- location(apple, kitchen) = location(pear, kitchen).

no

?- a(b,c(d,e(f,g))) = a(b,c(d,e(f,g))). yes

?- a(b,c(d,e(f,g))) = a(b,c(d,e(g,f))). no

Another simple form of unification occurs between a variable and a primitive. The variable takes on a value that causes unification to succeed.

?- X = a.

X = a

# ?- 4 = Y.

Y = 4

?- location(apple, kitchen) = location(apple, X). X = kitchen

In other cases multiple variables are simultaneously bound to values.

?- location(X,Y) = location(apple, kitchen). X = apple

Y = kitchen

?- location(apple, X) = location(Y, kitchen). X = kitchen

Y = apple

Variables can also unify with each other. Each instance of a variable has a unique internal Prolog value. When two variables are unified to each other, Prolog notes that they must have the same value. In the following example, it is assumed Prolog uses '\_nn,' where 'n' is a digit, to represent unbound variables.

# ?- X = Y. X = \_01 Y = \_01

?- location(X, kitchen) = location(Y, kitchen). X = \_01

# Y = \_01

Prolog remembers the fact that the variables are bound together and will reflect this if either is later bound.

?- X = Y, Y = hello.

X = hello

Y = hello

?- X = Y, a(Z) = a(Y), X = hello.

X = hello

Y = hello

Z = hello

The last example is critical to a good understanding of Prolog and illustrates a major difference between unification with Prolog variables and assignment with variables found in most other

languages. Note carefully the behavior of the following queries.

?- X = Y, Y = 3, write(X).

3

# X = 3

Y = 3

?- X = Y, tastes\_yucky(X), write(Y). broccoli

X = broccoli Y = broccoli

When two structures with variables are unified with each other, the variables take on values that make the two structures identical. Note that a structure bound to a variable can itself contain

variables.

?- X = a(b,c).

X = a(b,c)

?- a(b,X) = a(b,c(d,e)).

X = c(d,e)

?- a(b,X) = a(b,c(Y,e)).

X = c(\_01,e)

# Y = \_01

Even in these more complex examples, the relationships between variables are remembered and updated as new variable bindings occur.

?- a(b,X) = a(b,c(Y,e)), Y = hello. X = c(hello, e)

Y = hello

?- food(X,Y) = Z, write(Z), nl, tastes\_yucky(X), edible(Y), write(Z).

food(\_01,\_02)

food(broccoli, apple) X = broccoli

Y = apple

Z = food(broccoli, apple)

If a new value assigned to a variable in later goals conflicts with the pattern set earlier, the goal fails.

?- a(b,X) = a(b,c(Y,e)), X = hello. no

The second goal failed since there is no value of Y that will allow hello to unify with c(Y,e). The following will succeed.

?- a(b,X) = a(b,c(Y,e)), X = c(hello, e). X = c(hello, e)

Y = hello

If there is no possible value the variable can take on, then unification fails.

?- a(X) = a(b,c).

no

?- a(b,c,d) = a(X,X,d). no

The last example failed because the pattern asks that the first two arguments be the same, and they aren't.

?- a(c,X,X) = a(Y,Y,b).

no

Did you understand why this example fails? Matching the first argument binds Y to c. The second

argument causes X and Y to have the same value, in this case c. The third argument asks that X bind to b, but it is already bound to c. No value of X and Y will allow these two structures to unify.

The anonymous variable (\_) is a wild variable, and does not bind to values. Multiple occurrences of it do not imply equal values.

?- a(c,X,X) = a(\_,\_,b).

X = b

Unification occurs explicitly when the equal (=) built-in predicate is used, and implicitly when Prolog searches for the head of a clause that matches a goal pattern.

**Exercises**

Predict the results of these unification queries.

?- a(b,c) = a(X,Y).

?- a(X,c(d,X)) = a(2,c(d,Y)).

?- a(X,Y) = a(b(c,Y),Z).

?- tree(left, root, Right) = tree(left, root, tree(a, b, tree(c, d, e))).