Unification & Matching

So far, when we’ve gone through examples, I have said simply that when trying to satisfy a goal, Prolog searches for a matching rule or fact.

What does this mean, to match?

Prolog’s matching operator or \( \equiv \). It tries to make its left and right hand sides the same, by assigning values to variables.

Also, there’s an implicit \( \equiv \) between arguments when we try to match a query

\[
?- f(x, y)
\]

to a rule

\[
f(A, B) :- ....
\]
Matching Examples

The rule:

\[
\text{deriv}(U \^C, X, C \times U \^L \times DU) :-
\]
\[
\text{number}(C), L \text{ is } C - 1,
\]
\[
\text{deriv}(U, X, DU).
\]

?- deriv(x \^3, x, D).
D = 1\times3\times x^2

The goal:

\bullet \ x \^3 \text{ matches } U \^C
\bullet \ x = U, C = 3
\bullet \ x \text{ matches } X
\bullet \ D \text{ matches } C \times U \^L \times DU
Matching Examples...

deriv(U+V, X, DU + DV) :-
deriv(U, X, DU),
deriv(V, X, DV).

?- deriv(x^3 + x^2 + 1, x, D).
D = 1*3*x^2+1*2*x^1+0

x^3 + x^2 + 1 matches U + V
x^3 + x^2 is bound to U
1 is bound to V
Matching Algorithm

Can two terms $A$ and $F$ be “made identical,” by assigning values to their variables?

Two terms $A$ and $F$ match if

1. they are identical atoms
2. one or both are uninstantiated variables
3. they are terms $A = f_A(a_1, \ldots, a_n)$ and $F = f_F(f_1, \ldots, f_m)$, and
   (a) the arities are the same ($n = m$)
   (b) the functors are the same ($f_A = f_F$)
   (c) the arguments match ($a_i \equiv f_i$)
## Matching – Examples

<table>
<thead>
<tr>
<th>$A$</th>
<th>$F$</th>
<th>$A \equiv F$</th>
<th>variable subst.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>$\sin(X)$</td>
<td>$\sin(a)$</td>
<td>yes</td>
<td>$\theta = {X=a}$</td>
</tr>
<tr>
<td>$\sin(a)$</td>
<td>$\sin(X)$</td>
<td>yes</td>
<td>$\theta = {X=a}$</td>
</tr>
<tr>
<td>$\cos(X)$</td>
<td>$\sin(a)$</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>$\sin(X)$</td>
<td>$\sin(\cos(a))$</td>
<td>yes</td>
<td>$\theta = {X=\cos(a)}$</td>
</tr>
</tbody>
</table>
### Matching – Examples...

<table>
<thead>
<tr>
<th>( A )</th>
<th>( F )</th>
<th>( A \equiv F )</th>
<th>variable subst.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{likes}(c, X) )</td>
<td>( \text{likes}(a, X) )</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>( \text{likes}(c, X) )</td>
<td>( \text{likes}(c, Y) )</td>
<td>yes</td>
<td>( \theta = {X=Y} )</td>
</tr>
<tr>
<td>( \text{likes}(X, X) )</td>
<td>( \text{likes}(c, Y) )</td>
<td>yes</td>
<td>( \theta = {X=c, X=Y} )</td>
</tr>
<tr>
<td>( \text{likes}(X, X) )</td>
<td>( \text{likes}(c, __)</td>
<td>yes</td>
<td>( \theta = {X=c, X=47} )</td>
</tr>
<tr>
<td>( \text{likes}(c, \text{a}(X)) )</td>
<td>( \text{likes}(V, Z) )</td>
<td>yes</td>
<td>( \theta = {V=c,Z=\text{a}(X)} )</td>
</tr>
<tr>
<td>( \text{likes}(X, \text{a}(X)) )</td>
<td>( \text{likes}(c, Z) )</td>
<td>yes</td>
<td>( \theta = {X=c,Z=\text{a}(X)} )</td>
</tr>
</tbody>
</table>
Matching Consequences

Consequences of Prolog Matching:

- An uninstantiated variable will match any object.
- An integer or atom will match only itself.
- When two uninstantiated variables match, they *share*:
  - When one is instantiated, so is the other (with the same value).
- Backtracking undoes all variable bindings.
Matching Algorithm

FUNC Unify (A, F: term) : BOOL;
IF Is_Var(F) THEN Instantiate F to A
ELSIF Is_Var(A) THEN Instantiate A to F
ELSIF Arity(F) \neq Arity(A) THEN RETURN FALSE
ELSIF Functor(F) \neq Functor(A) THEN RETURN FALSE
ELSE
    FOR each argument i DO
        IF NOT Unify(A(i), F(i)) THEN
            RETURN FALSE
    RETURN TRUE;
Visualizing Matching

- From *Prolog for Programmers*, Kluzniak & Szpakowicz, page 18.

- Assume that during the course of a program we attempt to match the goal \( p(X, b(X, Y)) \) with a clause \( C \), whose head is \( p(X, b(X, Y)) \).

- First we’ll compare the arity and name of the functors. For both the goal and the clause they are 2 \( p \), respectively.
Visualizing Matching...

\[ p(X, b(X, Y)) \]

\[ p(A, b(c, A)) : - \ldots \]
The second step is to try to unify the first argument of the goal ($x$) with the first argument of the clause head ($A$).

They are both variables, so that works OK.

From now on $A$ and $x$ will be treated as identical (they are in the list of variable substitutions $\theta$).
Visualizing Matching...

\[ p(\boxed{X}, \, b(X, \, Y)) \]

\[ p(\boxed{A}, \, b(c, \, A)) \,:= \, \ldots \]
\[ \theta = \{ A = X \} \]
Next we try to match the second argument of the goal \((b(X, Y))\) with the second argument of the clause head \((b(c, A))\).

The arities and the functors are the same, so we go on to try to match the arguments.

The first argument in the goal is \(X\), which is matched by the first argument in the clause head \((c)\). I.e., \(X\) and \(c\) are now treated as identical.
Visualizing Matching...

\[ p(X, b([X], Y)) \]

\[ p(A, b([\text{c}, A]) : \theta = \{A = X, X = \text{c}\} \]

Diagram:
- Query
- Caller
- Callee
- Head
Finally, we match $A$ and $Y$. Since $A = X$ and $X = c$, this means that $Y = c$ as well.
Visualizing Matching...

\[ p(X, b(X, [Y])) \]

\[ p(A, b(c, [A])) :\theta \]
\[ \theta = \{ A = X, X = c, A = Y \} \]
A term is either a
- a constant (an atom or integer)
- a variable
- a structure

Two terms match if
- there exists a variable substitution $\theta$ which makes the terms identical.

Once a variable becomes instantiated, it stays instantiated.

Backtracking undoes variable instantiations.

Prolog searches the database sequentially (from top to bottom) until a matching clause is found.