520 — Principles of Programming Languages

14: Prolog III

Christian Collberg

collberg@cs.arizona.edu

Department of Computer Science
University of Arizona

Matching

The rule:

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

?- deriv(x ^ 3, x, D).
\[ D = 1 \ast 3 \ast x^2 \]

The goal:

\[ x ^ 3 \text{ matches } U ^ C \]
\[ x = U, C = 3 \]
\[ x \text{ matches } X \]
\[ D \text{ matches } C \ast U ^ L \ast DU \]

Matching...

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

?- deriv(x^3 + x^2 + 1, x, D).
\[ D = 1 \ast 3 \ast x^2 + 1 \ast 2 \ast x^1 + 0 \]

\[ x ^ 3 + x^2 + 1 \text{ matches } U + V \]
\[ x ^ 3 + x^2 \text{ is bound to } U \]
\[ 1 \text{ is bound to } V \]

Matching...

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 – Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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 – Example...

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>variable subst.</th>
</tr>
</thead>
<tbody>
<tr>
<td>likes(c, X)</td>
<td>likes(a, X)</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>likes(c, X)</td>
<td>likes(c, Y)</td>
<td>yes</td>
<td>$\theta = {X=Y}$</td>
</tr>
<tr>
<td>likes(X, X)</td>
<td>likes(c, Y)</td>
<td>yes</td>
<td>$\theta = {X=c, X=Y}$</td>
</tr>
<tr>
<td>likes(X, X)</td>
<td>likes(c, _)</td>
<td>yes</td>
<td>$\theta = {X=c, X=_{47}}$</td>
</tr>
<tr>
<td>likes(c, a(X))</td>
<td>likes(V, Z)</td>
<td>yes</td>
<td>$\theta = {V=c, Z=a(X)}$</td>
</tr>
<tr>
<td>likes(X, a(X))</td>
<td>likes(c, Z)</td>
<td>yes</td>
<td>$\theta = {X=c, Z=a(X)}$</td>
</tr>
</tbody>
</table>

### Matching...

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

```plaintext
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
            END IF
        END FOR
        RETURN TRUE;
```
Matching – Example

- 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 and \( p \), respectively.

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 \( \theta \)).
Matching – Example...

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.

Matching – Example II (c)

Finally, we match \( A \) and \( Y \). Since \( A = X \) and \( X = c \), this means that \( Y = c \) as well.

Matching – Example II (d)
Executing Prolog

```prolog
FUNC Execute (G = G_1, G_2, \ldots, G_m; Result);
    IF Is_Empty(G) THEN Result := Yes
    ELSE Result := No; i := 1;
        WHILE Result=No & i \leq \text{NoOfClauses} DO
            Clause := H_i : B_1, \ldots, B_n;
            IF Unify(G_1, Clause, \theta) THEN
                G' := substitute(B_1, \ldots, B_n, G_2, \ldots, G_m, \theta);
                Execute(G', Result);
            ENDIF; i := i + 1;
        ENDDO ENDIF
```

Execution Example

% From the Northern Exposure FAQ
friend(maggie, person(eve, yes)).
friend(maggie, person(moose(morty, yes))).
friend(maggie, person(harry, no)).
friend(maggie, person(bruce, no)).
friend(maggie, person(glenn, no)).
friend(maggie, person(dave, no)).
friend(maggie, person(rick, no)).
friend(maggie, person(mike, yes)).
friend(maggie, person(joel, yes)).

cause_of_death(morty, copper_deficiency).
cause_of_death(harry, potato_salad).
cause_of_death(bruce, fishing_accident).
cause_of_death(glenn, missile).
cause_of_death(dave, hypothermia).
cause_of_death(rick, hit_by_satellite).
cause_of_death(mike, none_yet).
cause_of_death(joel, none_yet).

male(morty). male(harry). male(bruce).
male(glenn). male(dave). male(rick).
male(mike). male(joel). female(eve).
Execution Example...

\[
\text{alive}(X) :\neg \text{cause_of_death}(X, \text{none,yet}).
\]

\[
\text{pastime}(\text{eve, hypochondriac}).
\]

\[
\text{pastime}(\text{mike, hypochondriac}).
\]

\[
\text{pastime}(X, \text{golf}) :\neg \text{job}(X, \text{doctor}).
\]

job(mike, lawyer).  job(adam, chef).

job(maggie, pilot).  job(joel, doctor).

\text{?- friend(maggie, person(B, yes)), male(B), alive(B), pastime(B, golf).}

\text{female(eve).}

\text{pastime(X, golf)} :\neg \text{job}(X, \text{doctor}).

\text{job(adam, chef).}

\text{job(joel, doctor).}

\text{job(mike, doctor).}

\text{job(joel, doctor).}

\text{alive(X)} :\neg \text{cause_of_death}(X, \text{none,yet}).

\text{pastime(X, golf)} :\neg \text{job}(X, \text{doctor}).

\text{job(adam, chef).}

\text{job(joel, doctor).}
Execution Example...

Match? Unify(Hi, G1) Scan database Empty? Yes No more Hi fail

Hi :− X1, ..., Xn Replace G1 by <empty>
that = {} Substitute vars from θ
alive(mike), pastime(mike, golf).

Hi :− X1, ..., Xn Replace G1 by <cause_od(X, none)>
that = {X=mike}
cause_od(mike, none), pastime(mike, golf).

Hi :− X1, ..., Xn Replace G1 by <empty>
pastime(mike, golf).

Execution Example...

We skip a step here.

`pastime(mike, golf)` unifies with

```
pastime(X, golf) :- job(X, doctor).
```

However, `job(mike, doctor)` fails, and we backtrack all the way up to the original query.
Execution Example...

friend(m,p(eve,yes)).
frend(m,m(morty,yes)).
frend(m,p(harry,no)).
frend(m,p(mike,yes)).
friend(m,p(joel,yes)).
cause_od(mike,none).
cause_od(joel,none).
alive(X):-cause_od(X, none).
male(mike).
male(joel).
female(eve).
pastime(eve, hypoc).
pastime(mike, hypoc).
pastime(X, golf):-job(X, doctor).
job(adam, chef).
job(joel, dentist).

Prolog So Far...

- 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.