Prolog programming: a do-it-yourself course for beginners

Day 2

Kristina Striegnitz
Department of Computational Linguistics
Saarland University, Saarbrücken, Germany
kris@coli.uni-sb.de
http://www.coli.uni-sb.de/~kris
Day 2: Matching and Proof Search

Today:  
- recursive predicate definitions
- how Prolog answers queries

Reader: Lectures 2 and 3 of *Learn Prolog Now!*
Ancestors

Task: Define a predicate \texttt{ancestor_of}(X, Y) which is true if X is an ancestor of Y.
Ancestors (cont.)

\[
\text{grandparent}_\text{of}(X, Y) :- \text{parent}_\text{of}(X, Z), \text{parent}_\text{of}(Z, Y).
\]
\[
\text{greatgrandparent}_\text{of}(X, Y) :- \text{parent}_\text{of}(X, Z), \text{parent}_\text{of}(Z, A), \text{parent}_\text{of}(A, Y).
\]
\[
\text{greatgreatgrandparent}_\text{of}(X, Y) :- \text{parent}_\text{of}(X, Z), \text{parent}_\text{of}(Z, A), \\
\quad \text{parent}_\text{of}(A, B), \text{parent}_\text{of}(B, Y).
\]

Doesn’t work for \textit{ancestor}_\text{of}; don’t know “how many parents we have to go back”.

\[
\text{ancestor}_\text{of}(X, Y) :- \text{parent}_\text{of}(X, Y).
\]

People are ancestors of their children,

\[
\text{ancestor}_\text{of}(X, Y) :- \text{parent}_\text{of}(X, Z), \text{ancestor}_\text{of}(Z, Y).
\]

and they are ancestors of anybody that their children may be an-
cestors of (i.e., of all the descendants of their children).
Ancestors (cont.)

\[\text{grandparent_of}(X,Y) \leftarrow \text{parent_of}(X,Z), \text{parent_of}(Z,Y).\]
\[\text{greatgrandparent_of}(X,Y) \leftarrow \text{parent_of}(X,Z), \text{parent_of}(Z,A), \text{parent_of}(A,Y).\]
\[\text{greatgreatgrandparent_of}(X,Y) \leftarrow \text{parent_of}(X,Z), \text{parent_of}(Z,A), \text{parent_of}(A,B), \text{parent_of}(B,Y).\]

Doesn’t work for \text{ancestor_of}; don’t know “how many parents we have to go back”.

\[\text{ancestor_of}(X,Y) \leftarrow \text{parent_of}(X,Y).\]

People are ancestors of their children,

\[\text{ancestor_of}(X,Y) \leftarrow \text{parent_of}(X,Z), \text{ancestor_of}(Z,Y).\]

and they are ancestors of anybody that their children may be ancestors of (i.e., of all the descendants of their children).
Example 1

KB: wizard(harry).
    wizard(ron).
    wizard(hermione).
    muggle(uncle_vernon).
    muggle(aunt_petunia).
    chases(crookshanks, scabbars).

Query: ?- wizard(hermione).
    yes

Easy: wizard(hermione) is a fact in the knowledge base.
Example 2

KB: wizard(harry).
    wizard(ron).
    wizard(hermione).
    muggle(uncle_vernon).
    muggle(aunt_petunia).
    chases(crookshanks,scabbars).

Query: ?- wizard(X).

X = harry ;
X = ron ;
X = hermione ;
no

• The query wizard(X) matches the fact wizard(harry). This instantiates the variable X with harry.

• It also matches the facts wizard(ron) and wizard(hermione).
Matching

- Two atoms match if they are the same atom.
  \[\text{Ex.}: \text{harry} = \text{harry}, \text{but} \quad \text{harry} \not= \text{'Harry'}.\]

- A variable matches any other Prolog term. The variable gets instantiated with the other term.
  \[\text{Ex.}: X = \text{wizard(harry)}\]
  \[\text{Ex.}: X = Y\]

- Two complex terms match if they have the same functor and the same number of arguments and if all pairs of parallel arguments match.
  \[\text{Ex.}: \text{like(harry,hargrid)} = \text{like(harry,X)}\]
  \[\text{Ex.}: \text{like(harry,hargrid)} = \text{like(harry,X,Y)}\]
  \[\text{Ex.}: \text{like(harry,hargrid)} \not= \text{like(X,X)}\]
**Back to Example 2**

**KB:**
- wizard(harry).
- wizard(ron).
- wizard(hermione).
- muggle(uncle_vernon).
- muggle(aunt_petunia).
- chases(crookshanks, scabbars).

**Query:**

```
?- wizard(X).
X = harry ;
X = ron ;
X = hermione ;
no
```

- Prolog checks for facts that **match** the query. (There are three.)
- Prolog starts from the top of the knowledge base and, therefore, finds `wizard(harry)` first.
- Typing `;` forces Prolog to check whether there are other possibilities.
Example 3

KB:

- eating(dudley).
- happy(aunt petunia) :- happy(dudley).
- happy(uncle vernon) :- happy(dudley), unhappy(harry).
- happy(dudley) :- kicking(dudley, harry).
- happy(dudley) :- eating(dudley).

Query:

?- happy(aunt petunia).

yes

- Check for a fact or a rule’s head that match the query.
- If you find a fact, you’re done.
- If you find a rule, prove all goals specified in the body of the rule.
Example 4

KB:
- eating(dudley).
- happy(aunt_petunia):-happy(dudley).
- happy(uncle_vernon):-happy(dudley),unhappy(harry).
- happy(dudley):-kicking(dudley,harry).
- happy(dudley):-eating(dudley).

Query:  ?- happy(X).
father(albert, james).
father(james, harry).
mother(ruth, james).
mother(lili, harry).
wizard(lili).
wizard(ruth).
wizard(albert).
wizard(X) :-
  father(Y, X),
  wizard(Y),
  mother(Z, X),
  wizard(Z).
Ancestors (cont.)

parent_of(paul, petunia).
parent_of(helen, petunia).
parent_of(paul, lili).
parent_of(helen, lili).
parent_of(albert, james).
parent_of(ruth, james).
parent_of(petunia, dudley).
parent_of(vernondudley).
parent_of(lili, harry).
parent_of(james, harry).

ancestor_of(X, Y) :-
    parent_of(X, Y).
ancestor_of(X, Y) :-
    parent_of(X, Z),
    ancestor_of(Z, Y).

Day 2: Matching and Proof Search – p.13
Practical Session

- matching
- proof search
- recursion

http://www.coli.uni-sb.de/~kris/essllli04prolog
(Maybe it’s a good idea to bookmark it, if you haven’t done so already.)