Slides to Accompany *Programming Languages and Methodologies*

R. J. Schalkoff

Chapter 3, Part 2: Prolog

Using Care with Recursion

```
/* inf-recur-ex.pro */

a:-a.

a.
```

```
?- ['inf-recur-ex.pro'].
% inf-recur-ex.pro compiled 0.01 sec, 0 bytes

Yes
?- listing(a).

a :-
a.
a.

Yes
?- a.
ERROR: Out of local stack
   Exception: (39,067) a ? abort
Yes
?- trace(a).
%         a/0: [call, redo, exit, fail]

Yes
[debug]  ?- a.
 T Call: (7) a
 T Call: (8) a
 T Call: (9) a
 T Call: (10) a
 T Call: (11) a
 T Call: (12) a
< some time later ...>
 T Call: (39,288) a
 T Call: (39,289) a
 T Call: (39,290) a
ERROR: Out of local stack
```

Prolog And Arithmetic (Example)

```
?- Y is (3/4+1).

Y = 1.75

Yes
?- Y is (3//4+1).

Y = 1

Yes
?- Y is (4/4+1).

Y = 2

Yes
?- Y is (4.0/4+1).

Y = 2
```

```
Yes
?- Y is (4.0/4+1.0).

Y = 2

Yes
?- Y is .21.
ERROR: Syntax error: Operator expected
ERROR: Y is .
ERROR: ** here **
ERROR: 21 .
?- Y is 0.21.

Y = 0.21

Yes
```

Lists in Prolog

Some important aspects of lists in Prolog, with examples:

- Lists in Prolog consist of elements separated by commas and enclosed in brackets `[]`, i.e.,

  ```
  [a, b, c, d]
  ```

  is a 4-element list of the elements a thru d.

- Lists may be used as arguments to predicates. A list is a single argument to a predicate, *regardless of the length of the list.*

- Lists may be comprised of variables, constants and other lists.

- Elementary list manipulation is based upon notation for the head (X) and tail (Y) of a list. These are denoted in Prolog statements as:

  ```
  [X | Y]
  ```

The head is the first element; the tail is the list with the head removed.

- The empty list is a list without any elements and denoted `[ ]`.

List Membership Example

We begin with a simple, recursive and complete definition of list membership:

- *X is a member of a list if it is the head (the first element) of the list or it is a member of the tail (the rest) of the list.*
- The first part of the description allows us to write

```
member(X, [X|_]).
```

- To test for membership in the tail of the list, we employ the second part of the description and use recursion in the form:

```
member(X, [_|Y]) :- member(X,Y).
```

Note: in what follows, we use the predicate name `member1` instead of simply `member`, since `member` is sometimes a built-in or system predicate.

```
/* member-ex.pro   */
/* example of lists, recursion */

member1(X,[X|_]).
member1(X,[_|Y]) :- member1(X,Y).
```

`member1` Use and Tracing

```
?- ['member-ex.pro'].
member-ex.pro compiled, 0.00 sec, 476 bytes.

Yes
?- listing(member1).

member1(A, [A|B]).
member1(A, [B|C]) :-
        member1(A, C).
```

```
Yes
?- member1(3, [1,3,4,5]).

Yes
?- member1(6, [1,3,4,5]).

No
?- trace(member1).
        member1/2: call redo exit fail

Yes
[debug]  ?- member1(3, [1,2,3,4,5]).
T Call:  (  7) member1(3, [1, 2, 3, 4, 5])
T Call:  (  8) member1(3, [2, 3, 4, 5])
T Call:  (  9) member1(3, [3, 4, 5])
T Exit:  (  9) member1(3, [3, 4, 5])
T Exit:  (  8) member1(3, [2, 3, 4, 5])
T Exit:  (  7) member1(3, [1, 2, 3, 4, 5])

Yes
```

```
?- member(3, [1,3,4,5]).

Yes
?- member(6, [1,3,4,5]).

No
?-
```

SWI-Prolog "member"

```
For help, use ?- help(Topic). or ?- apropos(Word).

?- help(member).
member(?Elem, ?List)
    Succeeds  when Elem can be unified with one of the members of List.
    The predicate can be used with any instantiation pattern.
```

'You often get more than you pay for'

```
?- member(What, [1,2,3]).

What = 1 ;

What = 2 ;

What = 3 ;

No
```