Lists (13A)

Young W. Lim
4/19/14

based on the following document:
http://www.learnprolognow.org/ Learn Prolog Now!

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice/OpenOffice.

Lists

- the elements enclosed by **square brackets** ([and])
- the elements are separated by **commas**
- the elements can be all kinds Prolog objects
- the elements can be another lists
- the number of elements : the **length** of a list
- the **zero element list** : the **empty list** ([])

[a, b, c, d]

[a, pred(b), X, 2, c]

[]

[d, [e, f], [e, pred(f)]]

[[], pred(a), [1, [b, c]], [], Z, [2, [b, c]]]

Head and Tail

Any **non-empty list** has two parts: the head and the tail.
the **head** is simply **the first element**
the **tail** is everything else

the **empty list** has neither a head nor a tail.
has no internal structure
is a special and simple list

the special built-in operator **| (vertical bar)**
to **decompose** a list into the head part and tail part
to get information from a list.
used together **with unification**.

```
?-  [Head|Tail]  =  [a,  b,  c,  d].

Head  =  a
Tail  =  [b,c,d]
yes


?-  [X|Y]  =  [].

no
```

Internal Variable Binding

X Y

?- [X|Y] = [[], pred(a), [2, [b, c]], [], Z].

the internal variable

X = []
Y = [dead(a),[2,[b,c]],[],_7800]
Z = _7800
yes

X Y T

?- [X,Y|T] = [[], pred(a), [2, [b, c]], [], Z].

the internal variable

X = []
Y = [dead(a),[2,[b,c]],[],_7800]
Z = _7800
yes

Anonymous Variable Binding

X1 X2 X3 X4

?- [X1,X2,X3,X4 | Tail] = [[], dead(z), [2, [b, c]], [], Z].

X1 = []
X2 = dead(z)
X3 = [2,[b,c]]
X4 = []
Tail = [_8910]
Z = _8910
yes

the anonymous variable
a variable, but the instantiated value is immaterial
Prolog does not tell its instantiated value
each occurrence of _ is independent
each is bound to different instantiated value

_ X _ Y

?- [_,X,_,Y|_] = [[], dead(z), [2, [b, c]], [], Z].

X = dead(z)
Y = []
Z = _9593
yes

Recursive Binding

_ _ [_|X] _

?- [_,_,[_|X]|_] = [[], dead(z), [2, [b, c]], [], Z, [2, [b, c]]].

_ X

X = [[b,c]]
Z = _10087
yes

_ _ [_|X] _

?- [_,_,[_|[X]]|_] = [[], dead(z), [2, [b, c]], [], Z, [2, [b, c]]].

_ [X]

X = [[b,c]]
Z = _10087
yes

member predicate

member(?Elem, ?List)

True if Elem is a member of List.
The SWI-Prolog definition differs from the classical one.
SWI-Prolog definition avoids unpacking each list element twice
and provides determinism on the last element.
E.g. this is deterministic:

```
member(X, [One]).
X=One.
```

swiprolog predicate member/2

an object X is a member of a list if it is the head of that list.

```
member(X, [X|T]).
member(X, [H|T])  :-  member(X,T).

member(X, [One]).
X=One;
false.
```

Determinism

```
member(X, [X|T]).
member(X, [H|T])  :-  member(X,T).
```

```
?- member(X,[1,2]).
X = 1 ;
X = 2 ;
No
```

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

```
member(B, [C|A]) :-    member_(A, B, C).
member_(_, A, A).
member_([C|A], B, _) :-    member_(A, B, C).
```

```
?- member(X,[1,2]).
X = 1 ;
X = 2 .
```

Determinism

```
member(X, [X|T]).
member(X, [H|T])  :-  member(X,T).
```

```
?- member(X,[1,2]).
X = 1 ;
X = 2 ;
No
```

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

```
member(B, [C|A]) :-    member_(A, B, C).
member_(_, A, A).
member_([C|A], B, _) :-    member_(A, B, C).
```

```
?- member(X,[1,2]).
X = 1 ;
X = 2 .
```

append predicate

```
?- append([a, b, c], [d, e], [a, b, c, d, e]).
Yes

?- append([a, b], [c, d], [e, f, g]).
No

?- append([a, b, c], [d, e], L).
L = [a, b, c, d, e]

?- append(L, [c, d], [a, b, c, d]).
L = [a, b]

?- append(L1, L2, [a, b, c]).
L1 = [], L2 = [a, b, c] ;
L1 = [a], L2 = [b, c] ;
L1 = [a, b], L2 = [c] ;
L1 = [a, b, c], L2 = [] ;
```

```
append([], L, L).

append([X | XS], YS, [X | ZS]) :-
    append(XS, YS, ZS).
```

delete predicate

```
delete([], _, []).

delete([E | List], E, ListWithoutE):-
        !,
        delete(List, E, ListWithoutE).

delete([H | List], E, [H | ListWithoutE]):-
        H \= E,
        !,
        delete(List, E, ListWithoutE).
```

intersection predicate

```
?- intersection([a, b, c], [d, b, e, a], L).
L = [a, b]
```

when no duplicates within the two input lists.
otherwise, intersection is obtained from the first input.

```
?- intersection([a, b, c, a], [d, b, e, a], L).
L = [a, b, a]
```

```
% Termination case
intersection([], _, []).

% Head of L1 is in L2
intersection([X | L1], L2, [X | L3]) :-
      member(X, L2),
      !,
      intersection(L1, L2, L3).

% Head of L1 is not in L2
intersection([X | L1], L2, L3) :-
      \+ member(X, L2),
      !,
      intersection(L1, L2, L3).
```

reverse predicate

```
reverse([],[]).

reverse([X | XS], YS] :-
        reverse(XS, RX),
        append(RX, [X], YS).
```

```
reverse(X, Y) :-
        reverse(X, [], Y).

reverse([], YS, YS).

reverse([X | XS], Accu, YS):-
        reverse(XS, [X | Accu], YS).
```

length predicate

```
?- length([a, b, c], 3).
Yes

?- length([a, [a, b], c], N).
N = 3


length([],0).

length([X | XS], N) :-
      length(XS, N1),
      N is N1 + 1.
```

Argument Modes

```
% quicksort(+InputList, -SortedList)

quicksort([], []) :- !.

quicksort([H | T], LSorted) :-

    split(H, T, LSmall, LBig),

    quicksort(LSmall, LSmallSorted),

    quicksort(LBig, LBigSorted),

    append(LSmallSorted, [H | LBigSorted], Lsorted).
```

Argument Modes

```
split(X, [Y | L], [Y | LSmall], LBig) :-
      before(Y, X),
      !,
      split(X, L, LSmall, Lbig).

split(X, [Y | L], LSmall, [Y | LBig]) :-
      !,
      split(X, L, LSmall, LBig).

split(_, [], [], []) :- !.


before(X, Y) :- X @< Y.
```

The **before**/2 predicate compares the list elements using the @</2 literal operator.

Argument Modes

(+) : an input, must be instantiated
(-) : an output, normally uninstantiated

multiple mode
  append(+L1, +L2, +L3)
  append(+L1, +L2, -L3)
  append(-L1, -L2, +L3)

(?) : can either be instantiated or not

  append(+L1,+L2, ?L3)
  append(?L1, ?L2, ?L3)

“@” a compound term argument that shall remain unaltered.

List Manipulation Predicates (1)

member	(?Elem, ?List)
append	(?List1, ?List2, ?List1AndList2)
append	(+ListOfLists, ?List)
prefix	(?Part, ?Whole)
select	(?Elem, ?List1, ?List2)
selectchk	(+Elem, +List, -Rest)
select	(?X, ?XList, ?Y, ?Ylist)
selectchk	(?X, ?XList, ?Y, ?YList)
nextto	(?X, ?Y, ?List)
delete	(+List1, @Elem, -List2)
nth0	(?Index, ?List, ?Elem)
nth1	(?Index, ?List, ?Elem)
nth0	(?N, ?List, ?Elem, ?Rest)
nth1	(?N, ?List, ?Elem, ?Rest)
last	(?List, ?Last)
proper_length	(@List, -Length)
same_length	(?List1, ?List2)
reverse	(?List1, ?List2)
permutation	(?Xs, ?Ys)
flatten	(+List1, ?List2)

List Manipulation Predicates (2)

flatten	(+List1, ?List2)
max_member	(-Max, +List)
min_member	(-Min, +List)
sum_list	(+List, -Sum)
max_list	(+List:list(number), -Max:number)
min_list	(+List:list(number), -Min:number)
numlist	(+Low, +High, -List)
is_set	(@Set)
list_to_set	(+List, ?Set)
intersection	(+Set1, +Set2, -Set3)
union	(+Set1, +Set2, -Set3)
subset	(+SubSet, +Set)
subtract	(+Set, +Delete, -Result)

List (13A)

References

[1] en.wikipedia.org
[2] en.wiktionary.org
[3] U. Endriss, “Lecture Notes : Introduction to Prolog Programming”
[4] http://www.learnprolognow.org/ Learn Prolog Now!
[5] http://www.csupomona.edu/~jrfisher/www/prolog_tutorial
[6] www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/intro.html
[7] www.cse.unsw.edu.au/~billw/dictionaries/prolog/negation.html