CS571: Programming Languages

List

List

List in Prolog similar to list in Haskel

|?- [H|T] = [1,2,3].
H = 1
T = [2,3].
No

| ?- [1|T] = [1,2,3].
T = [2,3].
no

List in Haskell vs Prolog

- Haskell:
  - A template for a primitive recursive definition over lists is:
    - f [] = … (base case)
    - f (x:xs) =… (recursive case)
  - Question: how to define f (x:xs) from f xs.
- Prolog
  - A template for a primitive recursive definition over lists:
    - pred([], …) (base case)
    - pred([H|T],…) (recursive case)
  - Question: how to define pred([H|T],...) from pred(T,...).

member

- Is a given integer in an integer list.

Haskell:

```
member a [] = False
member a (b:xs)
   | a = = b = True
   | otherwise = member a xs
```

Prolog:

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

or

```
my_member(H, [H1|T]) :- H = H1.
my_member(X, [H|T]) :- my_member(X,T).
```


Example: my_member

- Is a given integer in a list.

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

- Testing membership

```
| ?- my_member(a, [a,b,a])
```

Example: my_member

- Is a given integer in a list.

  my_member(H, [H|_T]).

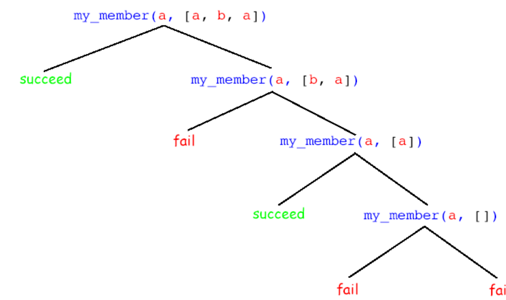
  my_member(X, [H|T]) :- my_member(X,T).
- Testing membership

  | ?- my_member(a, [a,b,a])

  yes

Derivations

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



Example: my_member

- Is a given integer in a list.

  my_member(H, [H|_T]).

  my_member(X, [H|T]) :- my_member(X,T).
- Testing membership

  | ?- my_member(a, [a,b,a])

  yes
- Enumerating members

  | ?- my_member(E, [a,b,a])

Example: my_member

- Is a given integer in a list.

  my_member(H, [H|_T]).

  my_member(X, [H|T]) :- my_member(X,T).
- Testing membership

  | ?- my_member(a, [a,b,a])

  yes
- Enumerating members

  | ?- my_member(E, [a,b,a])

  E = a.

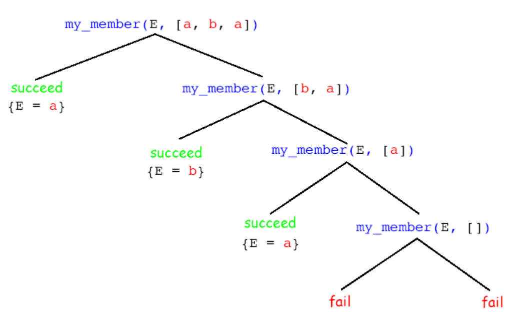
  E = b.

  E = a.

  no

Derivations

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



= vs. ==

- Can we use my_member to check if a variable is in a list? E.g. **my_member(X, [Y,Z]).**

  my_member(H, [H|_T]).

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

= vs. ==

- Can we use my_member to check if a variable is in a list? E.g. my_member(X, [Y,Z]).

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

No.

Revised my_member:

```
my_member(H, [H1|_T]) :- H == H1.
my_member(X, [H|T]) :-
        X \== H, my_member(X,T).
```

13

Example: length

- Finding the length of a list
  E.g. length([1,2,3], X) should return 3.

Haskell:

```
length [] = 0
length (x:xs) = 1 + length xs
```

Prolog:

Cs571 Programming Languages

14

Example: length

- Finding the length of a list
  E.g. length([1,2,3], X) should return 3.

Haskell:

```
length [] = 0
length (x:xs) = 1 + length xs
```

Prolog:

```
length([], 0).
length([H|T], N) :- length(T, N1), N is N1 + 1.
```

15

Example: length

- Finding the length of a list

```
length([], 0).
length([H|T], N) :-
    length(T, N1), N is N1 + 1.
```

- Testing the length

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

- Computing the length

```
| ?- length([a,b,c], X).
```

Cs571 Programming Languages

16

Example: length

- Finding the length of a list

```
length([], 0).
length([H|T], N) :-
    length(T, N1), N is N1 + 1.
```

- Testing the length

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

- Computing the length

```
| ?- length([a,b,c], X).
```

Cs571 Programming Languages

17

Example: length

- Finding the length of a list

```
length([], 0).
length([H|T], N) :-
    length(T, N1), N is N1 + 1.
```

- Testing the length

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

- Computing the length

```
| ?- length([a,b,c], X).
X = 3.
no
```

Cs571 Programming Languages

18

Example: append

- Append two lists:

| ?- append([1,2], [3,4], X)
X = [1,2,3,4].
no

| ?- append(X, [3,4], [1,2,3,4])
X = [1,2].
No

Example: append

- Append two lists:

Haskell:

```
append [] ys = ys
append (x:xs) ys = x: append xs ys
```

Prolog:

Example: append

- Append two lists:

Haskell:

```
append [] ys = ys
append (x:xs) ys = x: append xs ys
```

Prolog:

```
append([], L, L).
append([H1|T1], L, Res) :-
        append(T1, L, T2), Res = [H1|T2].
        or
append([], L, L).
append([H1|T1], L, [H1|T2]) :- append(T1, L, T2).
```


Example: Reverse a List

- rev/2 finds the reverse of a given list

e.g. rev([1,2,3], X) should succeed with X = [3,2,1].
what is the relationship between the reverse of [1,2,3] and the reverse of [2,3]

Example: Reverse a List

- rev/2 finds the reverse of a given list

e.g. rev([1,2,3], X) should succeed with X = [3,2,1].
what is the relationship between the reverse of [1,2,3] and the reverse of [2,3]

```
rev([], []).
rev([X|Xs], Ys) :- rev(Xs, Zs), append(Zs, [X], Ys).
```


Examples: my_last, last_but_one

- Find the last element of a list. Assume that the list contains at least one elements.

?- my_last([a,b,c,d],X)
X = d

- Find the last but one element of a list. Assume that the list contains at least two elements.

?- last_but_one([a,b,c,d],X)
X = c

Examples: my_last, last_but_one

- Find the last element of a list. Assume that the list contains at least one elements.

?- my_last([a,b,c,d],X)
X = d

my_last([X], X).
my_last([X,Y|Ys],Res) :- my_last([Y|Ys],Res).

- Find the last but one element of a list. Assume that the list contains at least two elements.

?- last_but_one([a,b,c,d],X)
X = c

Examples: my_last, last_but_one

- Find the last element of a list. Assume that the list contains at least one elements.

?- my_last([a,b,c,d],X)
X = d

my_last([X], X).
my_last([X,Y|Ys],Res) :- my_last([Y|Ys],Res).

- Find the last but one element of a list. Assume that the list contains at least two elements.

?- last_but_one([a,b,c,d],X)
X = c

last_but_one([X,_], X).
last_but_one([X,Y,Z|Ys],Res) :- last_but_one([Y,Z|Ys],Res).

Example: nth

- Find the nth element of a list (N>=1). Assume that the list contains at least N element.

?- nth([a,b,c,d],3,X)
X = c

Example: nth

- Find the nth element of a list (N>=1). Assume that the list contains at least N element.

?- nth([a,b,c,d],3,X)
X = c

nth([X|_],1,X).
nth([_|L],N,X) :-
    N > 1, N1 is N - 1, nth(L,N1,X).

Example: dupli

- Duplicate the elements of a list

?- dupli([a,b],L)
L = [a,a,b,b]

Example: dupli

- Duplicate the elements of a list

?- dupli([a,b],L)
L = [a,a,b,b]

dupli([],[]).
dupli([X|Xs],Res) :- dupli(Xs,Ys), Res=[X,X|Ys].

Or

dupli([],[]).
dupli([X|Xs],[X,X|Ys]) :- dupli(Xs,Ys).

Example: inc_odd

- Define a prolog predicate inc_odd(L,L1) which increases every element occurring at the odd position of L and stores the result in L1.

E.g. | ?- inc_odd([2,3,5,8],L).
L = [3,3,6,8].
no

Example: inc_odd

- Define a prolog predicate inc_odd(L,L1) which increases every element occurring at the odd position of L and stores the result in L1.

E.g. | ?- inc_odd([2,3,5,8],L).
L = [3,3,6,8].
no

```
inc_odd([],[]).
inc_odd([X],[X1]) :- X1 is X+1.
inc_odd([X,Y|Xs], [X1,Y|Res1]) :-
    inc_odd(Xs, Res1),
    X1 is X+1.
```


Example: remove

- Remove the nth element from a list. If the list contains less than n elements, then return the list.

?- remove([a,b,c,d],2,L)
L = [a,c,d]

Example: remove

- Remove the nth element from a list. If the list contains less than n elements, then return the list.

?- remove([a,b,c,d],2,L)
L = [a,c,d]

```
remove([],_,[]).
remove([_|Xs],1,Xs).
remove([Y|Xs],N,[Y|Ys]) :-
        N > 1,
        N1 is N - 1,
        remove(Xs,N1,Ys).
```


If-then-else

Write a Prolog program add(X,L1,L2) that adds X to list L1 if X is not in L1, and stores the result in list L2.

```
add(X,L1,L2) :-
(member(X,L1) -> L2 = L1
              ;  L2 = [X|L1]
).

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


Example: length

- Finding the length of a list

```
length([], 0).
length([H|T], N) :- length(T, N1), N is N1 + 1.
```

Using if-then-else:

Example: length

- Finding the length of a list

```
length([], 0).
length([H|T], N) :- length(T, N1), N is N1 + 1.
```

Using if-then-else:

```
length(L,N) :-
    (L == [] -> N is 0
     ;
        L = [H|T],
        length(T, N1),
        N is N1 + 1
    ).
```


Example: delete

- delete(X, L1,:L2): Removes all occurrences of X from list L1 and stores the result in L2

```
|?- delete(1,[2,1,3,1,4], L2).
   L2 = [2,3,4]
```


Example: delete

- delete(X, L1,:L2): Removes all occurrences of X from list L1 and stores the result in L2

```
|?- delete(1,[2,1,3,1,4], L2).
   L2 = [2,3,4]

delete(X, [], []).
delete(X, [H|T], Res) :- X == H, delete(X, T, Res).
delete(X, [H|T], [H|Res]) :- X \== H, delete(X, T, Res).
```

Using if-then-else:

```
delete(X, [], []).
delete(X, [H|T], Res) :- (X == H -> delete(X, T, Res)
                          ; delete(X, T, Res1),
                            Res = [H|Res1]
).
```


Example: split

- Write a prolog program split(L,L1,L2) that splits a list L into two lists L1 and L2 such that L1 contains all elements of L occurring at odd positions in L, and L2 contains all elements of L occurring at even positions in L. E.g.

```
| ?- split([1,2,4,5,6],L1,L2).
L1 = [1,4,6]
L2 = [2,5].
no.
```


Example: split

- Write a prolog program split(L,L1,L2) that splits a list L into two lists L1 and L2 such that L1 contains all elements of L occurring at odd positions in L, and L2 contains all elements of L occurring at even positions in L. E.g.

```
| ?- split([1,2,4,5,6],L1,L2).
L1 = [1,4,6]
L2 = [2,5].
no.

split([],[],[]).
split([X],[X],[]).
split([X,Y|T], [X|L1],[Y|L2]) :- split(T, L1, L2).
```


sum_odd

- Define a prolog predicate sum_odd(L,L1) which computes the sum of elements occurring at odd position L and store the result in L1. If the list is empty, then the sum is 0.

```
E.g. | ?- sum_odd([2,3,4,5,6,7,8],L).
L = 20.
```

sum_odd

- Define a prolog predicate sum_odd(L,L1) which computes the sum of elements occurring at odd position L and store the result in L1. If the list is empty, then the sum is 0.

E.g. | ?- sum_odd([2,3,4,5,6,7,8],L).
L = 20.

```
sum_odd([],0).
sum_odd([X],X).
sum_odd([X,Y|Xs], Res) :-
    sum_odd(Xs, Res1),
    Res is X + Res1.
```


splitn

- Define a prolog predicate splitn(L,N,L1,L2) to split a list L into two parts L1 and L2; the length of the first part is N.

?- splitn([a,b,c,d,e,f,g,h,i,k],3,L1,L2).
L1 = [a,b,c]
L2 = [d,e,f,g,h,i,k].

splitn

- Define a prolog predicate splitn(L,N,L1,L2) to split a list L into two parts L1 and L2; the length of the first part is N.

?- splitn([a,b,c,d,e,f,g,h,i,k],3,L1,L2).
L1 = [a,b,c]
L2 = [d,e,f,g,h,i,k].

```
splitn(L,0,[],L).
splitn([X|Xs],N,L1,L2) :-
        N > 0, N1 is N - 1,
        splitn(Xs,N1,L3,L4),
        L1 = [X|L3], L2 = L4.
```