Lists in Prolog

Representation, Core Predicates, and Patterns

What is a List in Prolog?

- Ordered collection of terms (atoms, numbers, vars, or compound terms)
- Bracket syntax: `[a,b,c]` or empty list: `[]`
- Recursive form: `[H|T]` where `H` is head, `T` is tail
- Internally, Prolog stores a list as linked cells (cons pairs)
- A cons pair (short for constructed pair) is the fundamental building block of lists in logic and functional programming languages.
  - Head (first element)
  - Tail (the rest of the list)

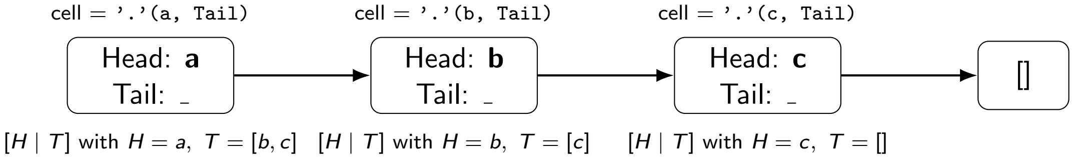
Examples

`[a,b,c]`	finite list of atoms
`[1,2,3]`	list of numbers
`[a,[b,c],d]`	nested lists
`[H|T]`	head–tail pattern

Prolog Lists as Cons Pairs

Syntactic sugar vs. internal term

$$[a, b, c] \equiv [a \mid [b \mid [c \mid []]]] \equiv \texttt{'.'(a, '.'(b, '.'(c, [])))}$$



cell = ’.’(a, Tail)

Head: **a**
Tail: _

$[H \mid T]$ with $H = a,\ T = [b, c]$

cell = ’.’(b, Tail)

Head: **b**
Tail: _

$[H \mid T]$ with $H = b,\ T = [c]$

cell = ’.’(c, Tail)

Head: **c**
Tail: _

$[H \mid T]$ with $H = c,\ T = []$

[]

- Each box is a **cons pair** = ’.’(*Head*, *Tail*).
- The **Tail** points to the **next cons cell** (or to [] to terminate the list).
- Fast head/tail access (O(1)); appending is linear (O(n)).

Head–Tail Decomposition

```
% head_tail(+List, -Head, -Tail).
head_tail([H|T], H, T).

?- head_tail([a,b,c], H, T).
H = a,
T = [b,c].
```

Listing 1: Extract head and tail

Membership (member/2)

```
member(X, [X|_]).          % X is the head
member(X, [_|T]) :-        % otherwise, search in tail
    member(X, T).

?- member(b, [a,b,c]).
true

?- member(d, [a,b,c]).
false
```

Listing 2: member/2 (recursive definition)

Concatenation (append/3)

```
append([], L, L).
append([H|T], L, [H|R]) :-
    append(T, L, R).

?- append([1,2], [3,4,5], X).
X = [1,2,3,4,5]
```

Listing 3: append/3

Reverse of lists

```
% Base case: reversing an empty list gives an empty list.
reverse_list([], []).

% Recursive step:
% Reverse the tail, then append the head to the end.
reverse_list([H|T], R) :-
    reverse_list(T, RT),
    append(RT, [H], R).
reverse([Head|Tail], SoFar, Reversed) :-
  reverse(Tail, [Head|SoFar], revese\_list/2).

?- reverse_list([1,2,3,4], X).
X = [4, 3, 2, 1]
```

Listing 4: Reverse of lists

Summing a List (sum_list/2)

```prolog
sum_list([], 0).
sum_list([H|T], Sum) :-
    sum_list(T, Rest),
    Sum is H + Rest.

?- sum_list([1,2,3,4], S).
S = 10.
```

Listing 5: sum_list/2

Hanoi Tower 1

```
move(1, X, Y,_):-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.
move(N,X,Y,Z):-
    N > 1,
    M is N-1,
    move(M, X, Z, Y),
    move(1, X, Y, _),
    move(M, Z, Y, X).

?- move(2,left, right, center).
Move top disk from left to center
Move top disk from left to right
Move top disk from center to right
```

Hanoi Tower 2

```prolog
move(1, X, Y,_):-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.
move(N,X,Y,Z):-
    N > 1,
    M is N-1,
    move(M, X, Z, Y),
    move(1, X, Y, _),
    move(M, Z, Y, X).

?- move(3,left, right, center). ???
```

Listing 7: Hanoi Tower 2

Factorial

```
% Base case
factorial(0, 1).

% Recursive case
factorial(N, F) :-
    N > 0,
    N1 is N - 1,
    factorial(N1, F1),
    F is N * F1.

?- factorial(6, V).
V = 720
```

Listing 8: factorial

Conditional statement

```
grade(Mark, Result) :-
    (  Mark >= 90 -> Result = ’A’
    ;  Mark >= 80 -> Result = ’B’
    ;  Mark >= 70 -> Result = ’C’
    ;  Result = ’F’
    ).
?- grade(85, R).
V = ’B’
```

Listing 9: Conditional statement

Loop statement

```
countdown(0).
countdown(N) :-
    N > 0,
    writeln(N),
    N1 is N - 1,
    countdown(N1).

?- countdown(5).
5
4
3
2
1
```

Listing 10: Loop statement