Prolog Lists, Operators and Arithmetic

Outcomes

1. Prolog predicate argument types
2. Directives
3. Prolog Lists
4. Prolog Operators
5. Prolog Arithmetic

Prolog

Prolog = “pure Prolog” + additions

Pure Prolog ~ logic

Additions make Prolog's logical basis to work in practice

Additions:

“Pure” (do not affect logical meaning – just notational cosmetics)

- List notation, operator notation

“Dirty” (do not have a logical meaning, eg. write(X))

- arithmetic, I/O

Argument Mode Indicators

https://www.swi-prolog.org/pldoc/man?section=preddesc

- An *argument mode indicator* (in the documentation) gives information about the intended direction in which information carried by a predicate argument is supposed to flow.

Example: https://www.swi-prolog.org/pldoc/doc_for?object=member/2

member(*?Elem, ?List*):

True if *Elem* is a member of *List*.

- Think of the ?-argument as either providing input or accepting output or being used for both input and output.

```
?- member(X,[a,b,c]).    % first arg is output, second arg is input
```

Predicate Directives

https://www.swi-prolog.org/pldoc/man?section=declare

https://www.swi-prolog.org/pldoc/doc_for?object=(dynamic)/1

- Informs the interpreter that the definition of the predicate(s) may change during execution (using assert/1 and/or retract/1).

https://www.swi-prolog.org/pldoc/doc_for?object=(multifile)/1

- Informs the system that the specified predicate(s) may be defined over more than one file. This stops consult/1 from redefining a predicate when a new definition is found.

https://www.swi-prolog.org/pldoc/doc_for?object=(discontiguous)/1

- Informs the system that the clauses of the specified predicate(s) might not be together in the source file. See also style_check/1.

List Notation

Examples of lists:

```
[ a, b, c, d]
[]
[ ann, tennis, tom, running]
[ link(a,b), link(a,c), link(b,d)]

[ a, [b,c], d, [ ], [a,a,a], f(X,Y) ]
```

Head and Tail

- L = [a, b, c, d]
  - a is head of L
  - [b, c, d] is tail of L
- More notation, vertical bar:
  - L = [Head | Tail]
  - L = [a, b, c] = [a | [b, c]] = [a, b | [c]] = [a, b, c | [ ]]

List notation is Syntactic Sugar

List notation: [Head | Tail]

- Equivalent to standard Prolog notation: '[|]'(Head, Tail)
- Note: '[|]' is a functor (with –traditional option to prolog interpreter, this functor is ".")

Equivalent terms:

- [a, b, c] = '[|]'(a, '[|]'(b, '[|]'(c, [ ])))

The latter expression can be, as usual, shown as a tree (first '[|]' is root of tree)

List Membership

```
% member( X, L) means that X is member of List L

member( X, [ X | _ ]).        % X appears as head of list

member( X, [ _ | L]) :-
    member( X, L).            % X in tail of list
```

TRY VARIOUS USES OF member/2

Concatenation of Lists

```
% conc( L1, L2, L3) means that L3 is concatenation of L1 and L2
conc( [ ], L, L).                    % Base case
conc( [X | L1], L2, [X | L3]) :-   % Recursive case
    conc( L1, L2, L3).
```

Many uses of conc

```
?- conc( [a,b,c], [1,2,3], L).
 L = [a,b,c,1,2,3]
?- conc( [a,[b,c],d], [a,[ ],b], L).
 L = [a, [b,c], d, a, [ ], b]
?- conc( L1, L2, [a,b,c] ).
 ....
```

Generating Lists

Try this:

```
?- conc( L, _, _).
 ....
```

Example of conc

Months before and after May?

```
?- Months = [jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec] ,
    conc( Before, [may | After], Months).
…
```

Another example of conc

Delete everything that follows three consecutive occurrences of 'z'

```
?- L1 = [a,b,z,z,c,z,z,z,d,e],     % Given list
conc( L2, [z,z,z | _ ], L1).        % L2 is L1 up to 3 z‟s
```

List membership with conc

```
% member2( X, L) means that X is member of list L
member2( X, L) :-
    conc( _, [X | _ ], L).
```

List Deletion

```
% del( X, L, NewL) means that NewL is the List L with first X removed
del( X, [X | Tail], Tail).
del( X, [Y | Tail], [Y | Tail1] ) :-
    del( X, Tail, Tail1).

?- del( X, [ a, b, c, d], L1).
```

List insertion

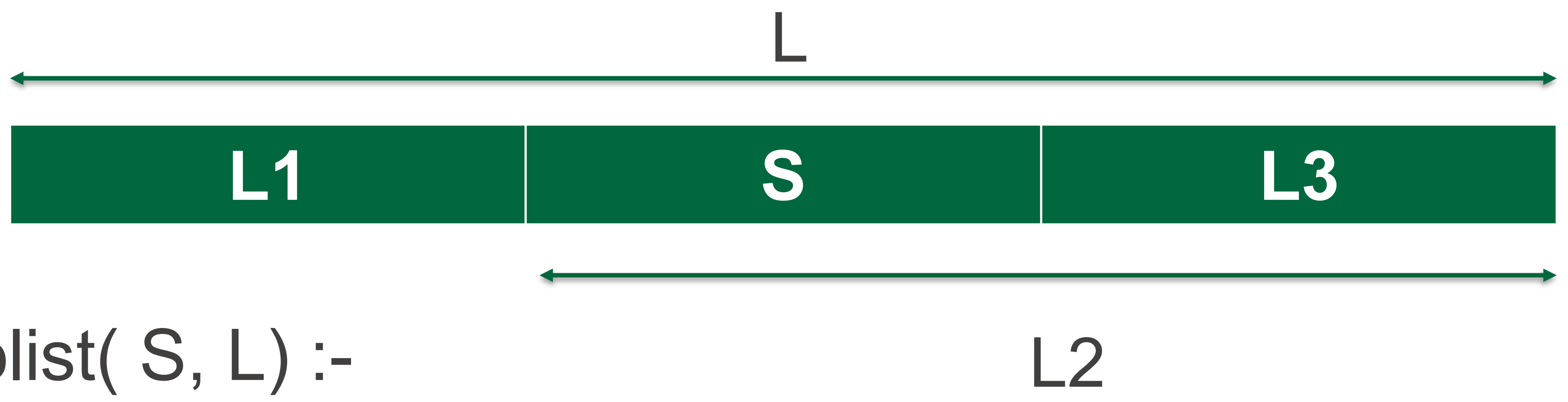
```
% insert( X, L, NewL) means that NewL is List L with X inserted anywhere
%  insert X into L "non-deterministically" at any position,
%  resulting in NewL
insert( X, L, [X | L]).                    % Insert X as head
insert( X, [Y | L], [Y | NewL]) :-
      insert( X, L, NewL).                 % Insert X into tail
```

Insert as opposite of delete

```
?- del( apple, L, [1,2,3] ). % What is L?

 ...

% insert( X, L, LongerL) means that LongerL is List L with X inserted at any position

insert( X, List, LongerList) :-

    del( X, LongerList, List).

% member3( X, L) means that X is an element of List L, alternative implementation

member3( X, L) :-
    del( X, L, _).                    % X can be deleted from L
```

Sublist of a List

```
% sublist( List, Sublist) means that  Sublist appears as a sublist in List
% It's easy!
% Just draw List and Sublist and rewrite the drawing into Prolog
```



```
sublist( S, L) :-
    conc( L1, L2, L),
    conc( S, L3, L2).
```

Time to check your learning!

Let’s see how many key concepts from Prolog Lists you recall by answering the following questions!

What is the difference between '[|]'(a,[]) and [a]?

What is the difference between [a] and [a|[ ]]?

Operator Notation

Operator notation is just a cosmetic, surface notational improvement

Equivalent notations for arithmetic expressions:

+(*(2,a), *(b,c)) = 2*a + b*c

+, * are infix operators built into Prolog

Higher precedence **in Prolog** means the outermost functor, so + has higher precedence than *

Operators with **lower Prolog precedence bind tighter** as a result of this

User-defined operators

https://www.swi-prolog.org/pldoc/man?section=operators

```
has( peter, information).

supports( floor, table).
```

This can be rewritten with infix operators as:

```
    :- op( 600, xfx, has).
    :- op( 600, xfx, supports).

peter has information.

floor supports table.
```

Operator types

(1) infix operators

xfx xfy yfx

(2) prefix operators

fx fy

(3) postfix operators

xf yf

yfx is left associative operator

xfy is right associative operator

‘y’should be interpreted as “on this position a term with precedence lower or equal to the precedence of the functor should occur” . For‘x’ the precedence of the argument must be strictly lower. (remember lower precedence in Prolog is more tightly binding)

Arithmetic operations

- Try to add 1 + 2 with:

```
?- X = 1+2.
X = 1 + 2          % Prolog just keeps expression unevaluated
```

- This is better:

```
?- X is 1 + 2.     % "is": built-in predicate that forces calculation

 X=3
```

- It is an error if the right side contains uninstantiated variables

-Number **is** *+Expr*

Arithmetic operations (cont'd)

+, -, *, /, **	addition, subtraction, …
//, mod	operations on integers
sin, cos, log, ...	standard functions

```
?- X is 2 + sin(3.14/2).
 X = 2.9999996829318345

 ?- A is 11/3.
Y = 3.6666666666666665

 ?- B is 11//3.
C=3
?- C is 11 mod 3.
C=2
```

Comparison Predicates

```
X>Y

X<Y

X >= Y

X =< Y

X=:=Y     X and Y are numerically equal

X=\=Y     X and Y are not numerically equal

?- 315 * 3 >= 250*4.

yes
?- 2+5 = 5+2.
no

?- 2+5 =:= 5+2.

yes
```

List length

```
% length( L, N): N is the length of list L

length( [ ], 0).

length( [ _ | L], N) :-
    length( L, N0),
    N is N0 + 1.
```

In the second clause, can the order of goals be reversed?

Time to check your learning!

Let’s see how many key concepts from Prolog Arithmetic you recall by answering the following questions!

What is the difference between the two following prolog statements:

```
X is 3 + 4
X = 3 + 4
```