Artificial Intelligence 1

Prof. Dr. Frank Hopfgartner
Dr. Matthias Horbach

Institute for Web Science and Technologies (WeST)
University of Koblenz

Overview

Definition

A list `L` is a sequence of terms. A list is itself a term.

Example:

```
[1,2,3,4,5]
[p(X,Y), 3, [X,Y,Z], q([a,b,c], 5), 7 ]
```

Alternative definition:

Definition

A list L is either

1. the empty list `[]` or
2. a term `T` followed by a list `L'`: `[T|L']`

For `L=[H|T]` we call `H` the head and `T` the tail of the list `L`.

```
?- [1,2,3,4,5] = [H|T].
H = 1,
T = [2, 3, 4, 5].
```

We could define:

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

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

How?

```
head([H|_],H).
tail([_|T],T).
```

`member(E,L)`: true iff `E` is element of the list `L`.

Definition:

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

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

?- member(4, [1,2,3]).
false.

?- member(X, [1,2,3]).
X=1 ;
X=2 ;
X=3 .

?- member(X, Y).
Y = [X|_G280] ;
Y = [_G16, X|_G20] ;
Y = [_G16, _G19, X|_G23] ; ...
```

`append(L1,L2,L)`: true iff `L` is the concatenation of `L1` and `L2`.

Definition:

```
append([],L,L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

```
?- append([1,2], [3,4], [1,2,3,4]).
true.

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

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

Lists 5/8

`reverse(L1,L2)`: true iff `L1` is the reverse list of `L2`.

```
?- reverse([1,2,3],[3,2,1]).
true.

?- reverse([1,2,3], L).
L = [3,2,1].
```

Definition:

```
reverse([],[]).
reverse([X|Y],L) :- reverse(Y,Z), append(Z,[X],L).
```

Lists 6/8

`length(L,X)`: true iff `X` is the number of elements in `L`.

```
?- length([1,2,3,4],4).
true.

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

?- length(X,4).
X = [_G2369, _G2372, _G2375, _G2378].
```

Definition:

```
length([],0).
length([_|X], N) :- length(X,M), N is (M+1).
```

Insertion sort in Prolog:

```
sorted([]).
sorted([X]).
sorted([X|[Y|Z]]) :- X =< Y, sorted([Y|Z]).

insert(X,[],[X]).
insert(X,[Y|Z], [X|[Y|Z]]) :- X =< Y.
insert(X,[Y|Z], [Y|U]) :- Y<X, insert(X,Z,U).

sort(X,X) :- sorted(X).
sort([X|Y], Z) :- sort(Y,U), insert(X,U,Z).
```

Examples:

```
?- sorted([1,2,3]).
true.

?- sorted([2,3,1]).
false.

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

The cut operator 1/4

Prolog always searches for all possible solutions for a query (via backtracking).

```
a(1). a(2). b(3).
p(X) :- a(X).
p(X) :- b(X).
```

```
?- p(X).
X = 1;
X = 2;
X = 3.
```

The cut operator 2/4

However, it can be reasonable to abort backtracking at some time

- for efficiency reasons: sometimes returning a single solution is sufficient
- for programmatic reasons: maybe the use of another clause should be avoided if a usable clause has already been found

The cut operator `!` can be used to abort backtracking

```
a(1). a(2). b(3).
p(X) :- !, a(X).
p(X) :- b(X).
```

```
?- p(X).
X = 1;
X = 2.
```

The cut operator 3/4

- The cut operator `!` is an atom and can be used as such in the body of any Prolog rule
- It has no influence on derivations as the atom `!` is always satisfied.
- Side effect: the derivation of the atom in the head of the current rule is “frozen”
  - All variable assignments are fixed
  - No further clauses with the same head are tried

Compare:

```
a(1). a(2). b(3).
p(X) :- !, a(X).
p(X) :- b(X).
```

```
?- p(X).
X = 1;
X = 2.
```

```
a(1). a(2). b(3).
p(X) :- a(X), !.
p(X) :- b(X).
```

```
?- p(X).
X = 1.
```

The cut operator 4/4

Another example:

```
p :- a, b.
p :- c.
```

$p \Leftarrow (a \wedge b) \vee c.$

```
p :- a, !, b.
p :- c.
```

$p \Leftarrow (a \wedge b) \vee (\neg a \wedge c).$

Note that for data basis {`a.`,`c.`} we can derive `p` in the left program but not in the right program.

Text output

- Prolog allows for text output during execution with the special predicate `write` (and `write_ln`)
- The use of text output has no effect on the derivation, atom with `write` are always satisfied.
- Side effect is text output

Beispiele:

```
?- write(john).
john
true.

?- write('A whole sentence').
A whole sentence
true.

?- write(X).
_G309
true.
```

Example: Path finding in directed graphs

Problem:

- Given a directed graph with nodes $k_1, \ldots, k_n$
- Is there a path between any k_i and k_j (for arbitrary i, j)?

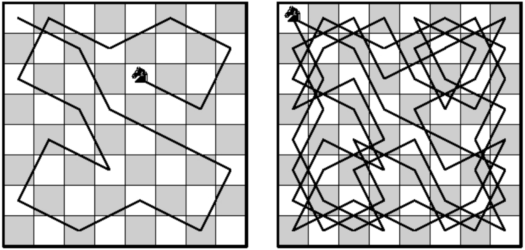
paths.pl:

```
edge(k1,k3).
edge(k2,k5).
edge(k4,k2).
edge(k5,k1).
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).
```

```
?- path(k1,k2).
false.

?- path(k2,k1).
true.
```

Example: knight's tour 1/2



Question: is there a sequence of moves of a knight on a chessboard such that the knight visits every square exactly once (i. e., is there a Hamilton tour for a knight?).

Simplified question here: is there a path from a square (x_1, y_1) to some other square (x_2, y_2)?

Example: knight's tour 2/2

knight.pl:

```
legal(1). ... legal(8).
move(X1,Y1,X2,Y2) :- legal(X1), legal(X2), legal(Y1), legal(Y2),
       X2 is X1+1, Y2 is Y1+2.
move(X1,Y1,X2,Y2) :- legal(X1), legal(X2), legal(Y1), legal(Y2),
       X2 is X1+1, Y2 is Y1-2.
move(X1,Y1,X2,Y2) :- legal(X1), legal(X2), legal(Y1), legal(Y2),
       X2 is X1-1, Y2 is Y1+2.
move(X1,Y1,X2,Y2) :- legal(X1), legal(X2), legal(Y1), legal(Y2),
       X2 is X1-1, Y2 is Y1-2.
move(X1,Y1,X2,Y2) :- legal(X1), legal(X2), legal(Y1), legal(Y2),
       X2 is X1+2, Y2 is Y1+1.
move(X1,Y1,X2,Y2) :- legal(X1), legal(X2), legal(Y1), legal(Y2),
       X2 is X1+2, Y2 is Y1-1.
move(X1,Y1,X2,Y2) :- legal(X1), legal(X2), legal(Y1), legal(Y2),
       X2 is X1-2, Y2 is Y1+1.
move(X1,Y1,X2,Y2) :- legal(X1), legal(X2), legal(Y1), legal(Y2),
       X2 is X1-2, Y2 is Y1-1.

path(X1,Y1,X1,Y1,L).
path(X1,Y1,X2,Y2,L) :- legal(X3), legal(Y3), move(X1,Y1,X3,Y3),
       not(member([X3,Y3],L)), path(X3,Y3, X2,Y2, [[X3,Y3]|L]).
path(X1,Y1,X2,Y2) :- path(X1,Y1,X2,Y2,[[X1,Y1]]).
```

Chapter 2.2: Prolog

Summary

Chapter 2.2: Summary

- First-order logic and Horn logic
- Prolog programs
  - data base D
  - rule base R
- Prolog interpreter SWI-Prolog
- Arithmetic operators `+`,`-`,`*`,`/`, comparison operators `=`, `=:=`, `=\=`,`<`,`>`,`>=`,`=<`, assignment `is`
- Lists and list operations
- Cut operator, text output