Introduction to Prolog
Reading and writing

CS181: Programming Languages

Topics:

- Reading and writing
- Prolog in action: Searching a maze
- Prolog in action: Searching directed graphs

Write predicate

- write( ) predicate writes a single term to the terminal. For example:

  write(a).

- The term can be a list (as long as it is *one* list, and not more):

  write([a, b]).

- Or something in the lines of:

  writemyname(X):- write([my, name, is, X]), nl.

- nl is (not surprisingly) the new line predicate.

Write predicate

- And from this point on, you can use write() in the same way as you would have used any other Prolog predicate:

```
writelist([ ]):- nl.
writelist([H | T]):- write(H), tab(1), writelist(T).
```

- tab( ) writes a number of spaces to the terminal.

Read predicate

- read(X) predicate reads a term from the keyboard and instantiates variable X to the value of the read term.
- This term has to be followed by a dot "." and a white space character (such as an enter or space).
- For example:

```
hello :- writelist([what, is, your, name, '?'],
    read(X),
    writelist([hello, X]).
```

Put predicate

- put( ) predicate prints its argument as a character on the terminal:

  ?- put(104), put(101), put(108), put(108), put(111).

- writes:

  hello

- Of course, we can do more sophisticated things:

  printstring([ ]).

  printstring([H|T]):-put(H), printstring(T).

  ?- printstring(“Vladimir Vacic”).

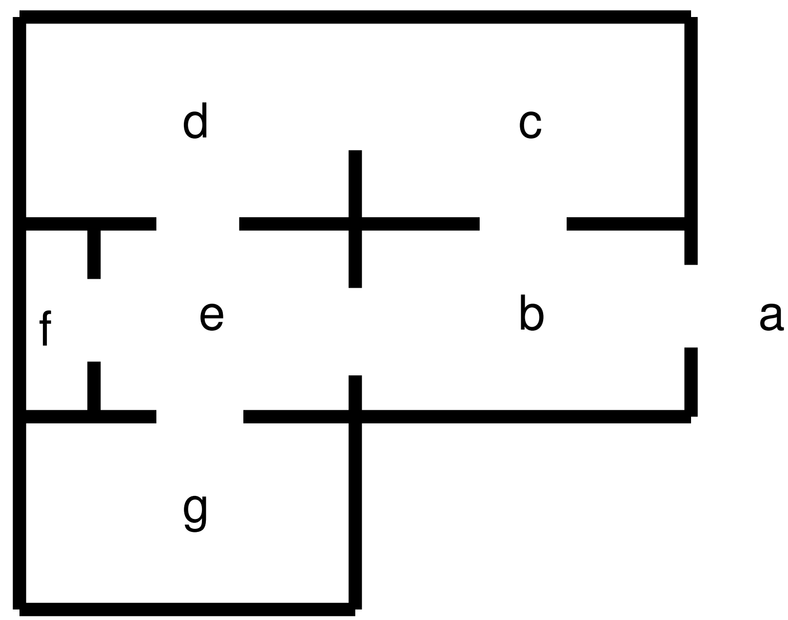
- Note the quotation marks “ ” – list of character codes is a string

Prolog in action: Searching

Searching a maze

- Let's say we have a simple maze like the one below:

```
door(a, b).
door(b, e).
door(b, c).
door(d, e).
door(c, d).
door(e, f).
door(g, e).
```



- Goal is to find a path from a room to another room.

Searching a maze

- Since door(a, b) is the same as door(b ,a), the maze can be modeled with an undirected graph.
- Using the standard approach: we are either in the goal room (base case), or we have to pass through a door to get closer to the goal room (recursive case).
- To avoid going in circles (b-c-d-e or a-b-a-b), we need to remember where “we have been so far” (does this phrase sound familiar?)

Searching a maze

- So, the solution would be something in the lines of:

```
go(X, X, T).
go(X, Y, T):- door(X, Z),
    not(member(Z, T)), go(Z, Y, [Z|T]).
go(X, Y, T):- door(Z, X),
    not(member(Z, T)), go(Z, Y, [Z|T]).
```

- Or, using the semicolon (logical or):

```
go(X, Y, T):- (door(X, Z) ; door(Z, X)),
    not(member(Z, T)), go(Z, Y, [Z|T]).
    go(Z, Y, [Z|T]).
```

Searching a maze

- Let’s add a twist to the story: in one of the rooms (we do not know exactly which one), there is phone which is ringing. We need to get to the room in which the phone is, and pick it up (this actually sound like an AI problem :-).
- So, we ask the question:

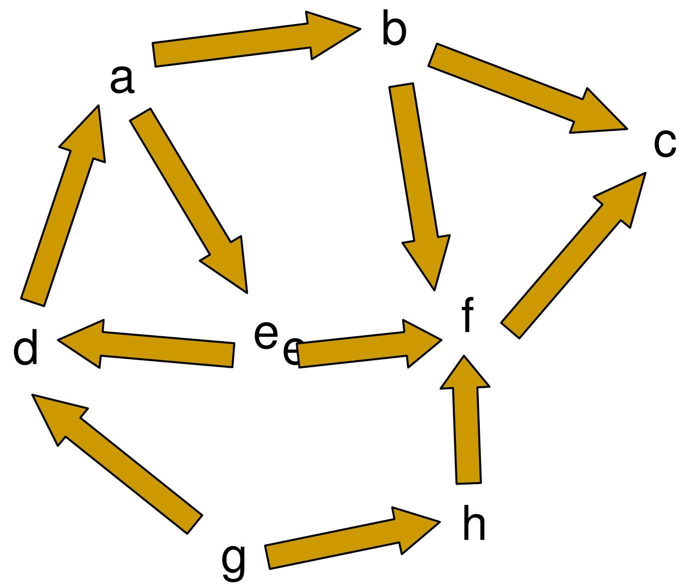
  ?- go(a, X, [ ]), phone(X).

- This follows the “generate and test” paradigm: we first generate a solution to the problem of how to get the room, then check if the phone is in the room.

Searching a graph

- Let's say we have a directed graph as the one below:

  edge(g, h).
  edge(g, d).
  edge(e, d).
  edge(h, f).
  edge(e, f).
  edge(a, e).
  edge(a, b).
  edge(b, f).
  edge(b, c).
  edge(f, c).
  edge(d, a).



- Again, goal is to find a path from a node to another node.

Searching a graph

- The simplest solution is:

```
cango(X, X).
cango(X, Y):- edge(X, Z), cango(Z, Y).
```

or

```
cango(X, Y):- edge(X, Y).
cango(X, Y):- cango(Z, Y), edge(X, Z).
```

- However, the graph has loops (a-e-d), so Prolog might never be able to resolve this (remember the airplane routing problem from assignment 1?)

Searching a graph

- We can use a solution similar to the maze search. However, let's say we are also interested in the route from X to Y:

```
route(Start, Dest, Route):-
    go(Start, Dest, [ ], R),
    rev(R, [ ], Route).

go(X, X, T, [X|T]).
go(Place, Dest, T, R):- edge(Place, Next),
    not(member(Next, T)),
    go(Next, Dest, [Place|T], R).
```

Searching a graph

- This algorithm performs breadth-first search.
- You can of course make this more complex by adding weights to the edges.
- Searching is actually a big topic in AI:
  - breadth-first search
  - depth-first search
  - best-first search (uses a heuristic do decide what is the “best” way to go)
  - A* search (uses the sum of the path so far and the heuristic to estimate the path left)

Reference

- Clocksin, W.F., and Mellish C.S. *Programming in Prolog*. 4th edition. New York: Springer-Verlag. 1994.