CENG 242 – Homework #6

Due: 4.6.2011 23:59

Introduction

Turtle graphics is a concept in computer graphics. It is mostly used for teaching programming to kids. You can learn more about turtle graphics from the wiki page. In this homework, you are going to write an interpreter for a simple turtle language (TL). Your turtle will take a program in TL as input, and produce a list of line segments that will be drawn by the turtle after executing the given program. The prolog code for a TL parser will be given to you so you won't need to worry about parsing.

Tasks

You have to define a predicate called "go", which takes two arguments. The first argument will be the TL code you are asked to execute. It is given as list of words. The second argument will be a variable that you will return from your go predicate. The variable will contain the list of line segments drawn by the turtle. The line segments are represented as:

drawline(x1, y1, x2, y2)

For example, when you are given the following query:

? - go([forward, 3], A).

Your program must return the following list in A.

A = [drawline(0,0,0,3)].

While the turtle is executing TL programs, it must keep track of its current state. The turtle state can be represented by a quintuple (**x**, **y**, **o**, **p**, **m**) where,

- **x** and **y** are the current coordinates of the turtle,
- **o** is the orientation of the turtle, it can take 8 different values: n, nw, w, sw, s, se, e, ne
- **p** is the pen state (up or down, where down means drawing and up means not drawing.
- **m** is the list of macros currently available.

Initially, the turtle will be positioned at (0,0), it will be facing north (+y) and its pen will be down. After that, each command in the program makes some change to the turtle's state. For example, a "turn left" command will update the orientation of the turtle, and a "pen down" command will put the turtle's pen down, so that the turtle will draw lines while moving.

In addition to these simple commands TL supports defining macros. Macros are used for grouping some commands and giving them a name, so that these commands can be executed later by referring to that name. You can think of macros as functions in other programming languages; after the function is defined, it can be called from other parts of the program. In order to call these macros, the turtle must also keep track of the macros defined so far.

Here is the complete list of commands in our turtle language.

- **forward X**
  Moves the turtle X units in the direction that the turtle is facing. If the pen is currently down, also outputs a line segment drawn during moving.
- **left**
  Turns the orientation of the turtle counter-clockwise 45 degrees.

- **right**
  Turns the orientation of the turtle clockwise 45 degrees.
- **penup**
  Holds the pen up. That means the turtle doesn't draw anything while moving.
- **pendown**
  Puts the pen down. That means the turtle draws line segments while moving.
- **push**
  Pushes the current state of the turtle to the stack.
- **pop**
  Pops the last state from the stack. That means, the turtle returns to the state where it last "pushed".
- **begin P end**
  Used to group multiple statements together. Works like { } in C.
- **def V P end**
  Defines a macro with name given in V. The contents of the macro are given in P, the macro ends with an **end** command.
- **call V**
  Executes the macro called V.
- **N command**
  If n is an integer, it repeats the next command N times.

Prolog Code for the Parser

You can use the below prolog code fragment to break TL programs into more meaningful parse trees. The *prog* predicate can be used like *prog(A, Input, []).* It parses the list given in Input and places the corresponding parse tree in A. Feel free to modify and use this to suit your needs.

```
prog([C|P]) --> command(C), prog(P).
prog([C]) --> command(C).
command(repeat(N,C)) --> [N], {integer(N)}, command(C).
command(forward(N)) --> [forward], [N], {integer(N)}.
command(left) --> [left].
command(right) --> [right].
command(penup) --> [penup].
command(pendown) --> [pendown].
command(push(P)) --> [push] , prog(P), [pop].
command(group(P)) --> [begin] , prog(P), [end].
command(def(V,P)) --> [def], variable(V), prog(P), [end].
command(call(V)) --> [call], variable(V).

variable(V) --> [V],{atom(V)}.
```

Sample Input & Output

Consider the following TL program:

```
def drawbox
     def drawandturn
          forward 1
          2 right
     end
     pendown
     4 call drawandturn
     penup
end
left
call drawbox
```

It will be called as:
?- go([def, drawbox, def, drawandturn, forward, 1, 2, right, end, pendown, 4, call, drawandturn, penup, end, left, call, drawbox], X).

And it should return:
X = [drawline(0,0,-1,1),drawline(-1,1,0,2),drawline(0,2,1,1),drawline(1,1,0,0)]

A similar program can be written in the following way, using begin… end.

```
def drawbox
     pendown
     4 begin
          forward 1
          2 right
     end
     penup
end
left
call drawbox
```

It can be called as
?- go([def, drawbox, pendown, 4, begin, forward, 1, 2, right, end, penup, end, left, call, drawbox], P).

The output will be:
P = [drawline(0,0,-1,1),drawline(-1,1,0,2),drawline(0,2,1,1),drawline(1,1,0,0)]

Specifications & Grading

- Your codes will be graded according to the following policy. You can get partial grades even if you can't complete everything.
    - Basic commands (forward, left, right, penup, pendown) : 50 points.
    - Push/Pop: +15 points.
    - Repetition/Begin/End: +15 points.
    - Define/Call: +20 points.
- All the work should be done individually.
- Your codes should be written in GNU Prolog and have the name "hw6.pl"
- In evaluation, black box method will be used. So be careful about the name the predicate.
- You will submit your code through Cow system.
- You should test your codes in inek machines with gprolog before submitting.