Intelligent Systems: Reasoning and Recognition

James L. Crowley

ENSIMAG 2 / MoSIG M1 Second Semester 2010/2011

Exercise 2 25 Februrary 2011

GRAPHSEARCH with CLIPS

The objective of this exercise is to program Nilsson's Graphsearch algorithm with forward chaining rules using CLIPS. While this is not necessarily the best method of programming such an algorithm, it is a good exercise for CLIPS programming and for use of Graphsearch for planning.

In this exercise, we will use CLIPS rules plan a path for a mobile robot. The robot's knowledge of the world is provided by a "Network of Places". A place is defined by a name, a 2D cartesian position, and a list of adjacent "neighboring" places. Any place can be reached from a neighboring place by a simple straight line motion. Places are defined as templates :

```
(deftemplate place
	(slot name (type SYMBOL) (default NIL))
	(slot x (type NUMBER))
	(slot y (type NUMBER))
	(multislot neighbors (default NIL))
)
```

The Graphsearch algorithm will construct a search tree composed of "nodes". Search tree nodes are are represented by a template defined with the following attributes:

name: The name of the place represented by this node
status: A slot indicating if the node is Open, Closed, Active or New.
father: The Parent node in the search tree. The node by which this node is reached.
f: The estimated cost of a path from start to goal through this place
g: The estimated cost of a path from start to this place.
h: The estimated cost of a path from this place to the goal.
x: The x coordinate of the position of the place
y: The x coordinate of the position of the place

You will use Euclidean distance to compute the estimated cost from the node to the goal. This can be computed using the following function:

```
(deffunction distance (?x1 ?y1 ?x2 ?y2)
 (bind  ?dx (- ?x1 ?x2))
 (bind  ?dy (- ?y1 ?y2))
       (sqrt (+ (* ?dx ?dx) (* ?dy ?dy)))
)
```

The distance function can be executed in the action part of the rules as follows:

```
=>
       (modify ?E (h ( distance ?x1 ?y1 ?x2 ?y2)))
```

The program is initiated with the creation of facts for the start and goal places of the form:

(start <PLACE>)
(goal <PLACE>)

In each cycle, the rule Generate will generate a new set of nodes to explore. This rule is triggered by existence of a node with the status active, and generates new nodes with status new.

```
;;
;;  Rule to create new nodes for the network of places.
;;


(defrule Generate
       "Generate nodes "
       (declare (salience 20))
       ?n <- (node (name ?p) (status active) (g ?g) (x ?x1) (y ?y1))
       (place (name ?p) (neighbors $? ?v $?))
       (place (name ?v) (x ?x2) (y ?y2))
=>
       (assert (node (name ?v) (status new) (father ?p) (x ?x2) (y ?y2) (h 0)
              (g (+ ?g (distance ?x1 ?y1 ?x2 ?y2)))))
)
```

For each rule that you write, give the rule a name of the form Rn, where n is a number from 1 to 6. This will help in correction of the exercise. You should only need 6 rules, labeled R1 through R6.

a) Write a deftemplate command to declare the template for nodes of the search tree, as well as the start and goal nodes. The start and goal nodes have only the names of the start and goal.

b) In each cycle of the algorithm, one node is labeled as "active". Initially this is the node that corresponds to the start place. Assume that the a fact (start <PLACE>) has been created. Write a rule (R1) that creates a node for the start place, using the network of places to determine the position of this place. Give this node the status "active".

c) Write a rule (R2) to detect if the active node is the goal. When the active node is the goal, then print out the message "goal found" and halt. Give this rule a salience of 20.

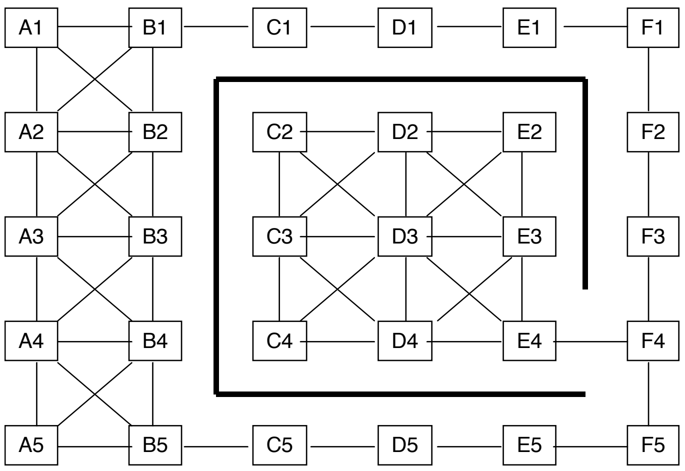
d) When nodes are created by the “Generate” rule, they are given the status "new". Nodes for places that have already been explored need to be relabeled as closed. Write the rule (R3) that detects if a node of status "new" has the same name as a node of status "closed". If so, delete the new node.

e) To explore a new node, we must compute the estimated cost, h, of a path from the current node to the goal. Write the rule (R4) that calculates the cost h for a node of status new using the distance to the goal from the place give by the node.

f) Write a rule, R5 that calculates f, the estimated cost of a path from the start point to the goal passing through the place listed by this node. The cost f is h+g. (R5) should not execute unless h is not zero. R5 should also change the status from new to open.

g) At the conclusion of each cycle, a node with status "open" with the lowest cost f is selected as the next "active" node. Write the rule R6 that selects the open node with a cost f such that no other open node exists with a lower cost. This rule should not fire if there exist any nodes to status "new" or "active".

To test your solution you may use the following network. Your robot should plan a path from A3 to C3.



```
(deffacts network-of-places
      (place(name A1) (x 0) (y 1) (neighbors   A2 B1 B2))
```

```
	(place(name A2) (x 0) (y 2) (neighbors   A1 A3 B1 B2 B3))
	(place(name A3) (x 0) (y 3) (neighbors   A2 A4 B2 B3 B4))
	(place(name A4) (x 0) (y 4) (neighbors   A3 A5 B3 B4 B5))
	(place(name A5) (x 0) (y 5) (neighbors   A4 B4 B5))
	(place(name B1) (x 1) (y 1) (neighbors   A1 A2 B2 C1))
	(place(name B2) (x 1) (y 2) (neighbors   A1 A2 A3 B1 B3))
	(place(name B3) (x 1) (y 3) (neighbors   A2 A3 A4 B2 B4))
	(place(name B4) (x 1) (y 4) (neighbors   A3 A4 A5 B3 B5))
	(place(name B5) (x 1) (y 5) (neighbors   A4 A5 B4 C5))
	(place(name C1) (x 2) (y 1) (neighbors   B1 D1))
	(place(name C2) (x 2) (y 2) (neighbors   C3 D2 D3))
	(place(name C3) (x 2) (y 3) (neighbors   C2 C4 D2 D3 D4))
	(place(name C4) (x 2) (y 4) (neighbors   C3 D3 D4))
	(place(name C5) (x 2) (y 5) (neighbors   B5 D5))
	(place(name D1) (x 3) (y 1) (neighbors   C1 E1))
	(place(name D2) (x 3) (y 2) (neighbors   C2 C3 D3 E2 E3))
	(place(name D3) (x 3) (y 3) (neighbors   C2 C3 C4 D2 D4 E2 E3 E4))
	(place(name D4) (x 3) (y 4) (neighbors   C3 C4 D3 E3 E4))
	(place(name D5) (x 3) (y 5) (neighbors   C5 E5))
	(place(name E1) (x 4) (y 1) (neighbors   D1 F1))
	(place(name E2) (x 4) (y 2) (neighbors   D2 D3 E3))
	(place(name E3) (x 4) (y 3) (neighbors   D2 D3 D4 E2 E4))
	(place(name E4) (x 4) (y 4) (neighbors   D3 D4 E3 F4))
	(place(name E5) (x 4) (y 5) (neighbors   D5 F5))
	(place(name F1) (x 5) (y 1) (neighbors   E1 F2))
	(place(name F2) (x 5) (y 2) (neighbors   F1 F3))
	(place(name F3) (x 5) (y 3) (neighbors   F2 F4))
	(place(name F4) (x 5) (y 4) (neighbors   E4 F3 F5))
	(place(name F5) (x 5) (y 5) (neighbors   E5 F4))
)


;;
;; Rule to inialise the research
;;

(defrule init "initiatlise the goal and start nodes"
	(initial-fact)
	(not (start))
=>
	(printout t "The name of the start node? ")
	(bind ?n (read))
	(assert (start (name ?n) ))
	(printout t "The name of the goal? ")
	(bind ?n (read))
	(assert (goal (name ?n)))
)

;;
;; recover the path
```

```
;;

(defrule recover-path "Compose a path list"
        ?C <- (path ?n $?q)
        (node  (name ?n) (father  ?p&:(neq ?p nil)))
        (goal (name ?b&:(neq ?p ?b)))
=>
        (retract ?C)
        (assert (chemin ?p ?n $?q))
)

;;
;; print the path
;;

(defrule print-path      "Print out the path"
        (path ?d $?q ?b)
        (start (name ?d))
=>
        (printout t "the path from " ?d " to " ?b " goes through " $?q crlf)
        (halt)
)
```