Práctica No. 3b
Mecanismos de búsqueda no informada con sistemas de producción en Common Lisp

Introducción

Uno de los primeros paradigmas de la Inteligencia Artificial fue la representación de problemas como un espacio de estados, lo que permitió definir la solución de los mismos como la búsqueda de un estado que represente una solución aceptable al problema partiendo de un estado que represente la situación inicial del problema. Los algoritmos de búsqueda se clasifican en *No informados* e *Informados*. Dentro de los algoritmos no informados se encuentran: *i*) *Búsqueda Primero en Profundidad* y *ii*) *Búsqueda Primero en Amplitud*. El espacio de estados puede estar representado explícitamente o implícitamente. En la representación implícita normalmente se utiliza un sistema de producción para generar sobre la marcha los estados siguientes de un estado dado.

En esta práctica, se utilizan mecanismos de búsqueda en espacios de estados implícitos, generados por medio de sistemas de producción. Se aplican los métodos no informados primero en profundidad y primero en amplitud al problema de las jarras y se te pide que lo adaptes para el problema de los misioneros y caníbales.

Actividades

1.- El problema de las jarras

1.1- Captura el siguiente código de Common Lisp en el archivo `probjarras.lsp`. Este código implementa las búsquedas primero en amplitud y primero en profundidad con estados implícitos para resolver el problema de las jarras.

```
;;;; ------------------------------------------------------------------------
;;;; Archivo: probjarras.lsp
;;;; Implementa los algoritmos de busqueda Primero en profundidad (bpp) y
;;;; Primero en amplitud (bpa) para resolver el problema de las jarras
;;;; ------------------------------------------------------------------------
(defun extiende (trayectoria)
     (mapcar #'(lambda (nuevo-nodo) (cons nuevo-nodo trayectoria))
             (remove-if #'(lambda (nvo-nodo) (member nvo-nodo trayectoria :test #'equal))
                    (estados-nuevos (first trayectoria)))
      )
)

(defun bpp (inicial &optional (cola (list (list inicial))))
      (cond  ((endp cola) nil) ; ¿esta vacia la cola?
             ((metap (first (first cola))) ; trayectoria completa?
                         (reverse (first cola)) )  ; Devuelve trayectoria.
             (t (bpp inicial (append (extiende (first cola)) (rest cola))))
      )
)

(defun bpa (inicial &optional (cola (list (list inicial))))
      (cond ((endp cola) nil)  ; ¿esta vacia la cola?
            ((metap (first (first cola))) ; trayectoria completa?
                         (reverse (first cola)) )  ; Devuelve trayectoria.
            (t (bpa inicial (append (rest cola)(extiende (first cola)))))
      )
)

;;; -------------------------------------------------------
;;; Define como meta obtener 2 litros en la jarra A.
;;; -------------------------------------------------------
(defun metap (estado)
      (= (first estado) 2)
)
```

```
;;;-----------------------------------------------------------------
;;; Sistema de produccion para el problema de las 2 jarras
;;; Estado: (A,B), donde A = Jarra de 4 litros, B = Jarra de 3 litros
;;;-----------------------------------------------------------------
(defun jarra-a (estado) (first estado))
(defun jarra-b (estado) (second estado))
(defun haz-estado (p s) (list p s))

(defun estados-nuevos (estado)
      (remove-if #'(lambda (x) (endp x))
            (list
                  (llena-a estado)
                  (llena-b estado)
                  (vacia-a-en-b estado)
                  (vacia-b-en-a estado)
                  (vacia-a estado)
                  (vacia-b estado)
            )
      )
)

(defun llena-a (estado)
      (cond ((< (jarra-a estado) 4) (haz-estado 4 (jarra-b estado))))
)

(defun llena-b (estado)
      (cond       ((< (jarra-b estado) 3) (haz-estado (jarra-a estado) 3)))
)

(defun vacia-a-en-b (estado)
      (let ( (p (jarra-a estado))
             (s (jarra-b estado)))
           (cond  ((zerop p) nil)          ; Jarra A vacia no puede vaciarse
                  ((= s 3) nil)                  ; Jarra B llena no puede llenarse
                  ((<= (+ p s) 3)                ; Vacia A en B
                        (haz-estado 0 (+ p s)))
                  (t    (haz-estado (- (+ p s) 3) 3)))
           )
)

(defun vacia-b-en-a (estado)
      (let ((p (jarra-a estado))
            (s (jarra-b estado)))
           (cond  ((zerop s) nil)          ; Jarra B vacia no puede vaciarse
                  ((= p 4) nil)                  ; Jarra A llena no puede llenarse
                  ((<= (+ p s) 4)                ; Vacia B en A
                        (haz-estado (+ p s) 0))
                  (t    (haz-estado 4 (- (+ p s) 4)))
           )
      )
)

(defun vacia-a (estado)
      (cond ((> (jarra-a estado) 0) (haz-estado 0 (jarra-b estado))))
)

(defun vacia-b (estado)
      (cond ((> (jarra-b estado) 0) (haz-estado (jarra-a estado) 0)))
)
```

1.2.- Desde el intérprete de Common Lisp carga el archivo tecleando (`load "probjarras.lsp"`), ejecuta las siguientes búsquedas y reporta el resultado en cada caso:

`(bpa '(0 0))`Resultado: _____
`(bpa '(0 1))`Resultado: _____
`(bpa '(1 0))`Resultado: _____
`(bpp '(0 0))`Resultado: _____
`(bpp '(0 1))`Resultado: _____
`(bpp '(1 0))`Resultado: _____

Modificando la meta
La meta del programa actúal es obtener 2 litros en la jarra A. Modifica el código de la función `metap(estado)` para que la nueva meta sea obtener 2 litros en la Jarra B.

1.3.- Nuevo código de la función `metap(estado)`

1.4.- Recarga el archivo tecleando `(load "probjarras.lsp")`, ejecuta las siguientes búsquedas y reporta el resultado en cada caso:

`(bpa '(0 0))` Resultado: _____
`(bpa '(0 1))` Resultado: _____
`(bpa '(1 0))` Resultado: _____
`(bpp '(0 0))` Resultado: _____
`(bpp '(0 1))` Resultado: _____
`(bpp '(1 0))` Resultado: _____

2.- El problema de los misioneros y caníbales
2.1.- Modifica el archivo `probjarras.lsp` para implementar las búsquedas no informadas para resolver el *Problema de los Misioneros y Caníbales*. Guarda el archivo con tu propuesta de solución con el nombre `probmyc.lsp`. Para generar los nuevos estados utiliza el sistema de producción definido en el archivo `spmyc.lsp`, generado en la práctica 2.

2.2.- Reporta la nueva definición de la función `estados-nuevos (estado)`

2.3.- Reporta la nueva definición de la función `metap (estado)`

2.4.- Desde el intérprete de Common Lisp carga el archivo `probmyc.lsp`, ejecuta las siguientes búsquedas y reporta los resultados obtenidos:

`(bpa '(izq 3 3))` Resultado: _____

`(bpa '(izq 2 2))` Resultado: _____
`(bpa '(der 1 0))` Resultado: _____
`(bpp '(izq 3 3))` Resultado: _____
`(bpp '(izq 2 2))` Resultado: _____
`(bpp '(der 1 0))` Resultado: _____

2.5.- Anexa el contenido completo del archivo `probmyc.lsp`

3.- **Comentarios y conclusiones**

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____