TEMA 2
Ejercicios propuestos

SOLUCIONES

Ejemplo: Dada una lista de números desordenados, escribir **una única regla** en clips que obtenga una nueva lista con los **números ordenados**

Por ejemplo, dada la lista (lista 9 2 4 6 5 3 7 8), obtendría la lista (lista 2 3 4 5 6 7 8 9).

```
(deffacts lista (lista 9  2  4  6  5  3  7  8))    ;Es un ejemplo de lista

(defrule ordenar
  ?f <- (lista  $?a   ?x    ?y   $?b)
  (test (< ?y ?x))
=>
  (retract ?f)
  (assert (lista   $?a   ?y   ?x   $?b)))
```

Se desea realizar una búsqueda A* en CLIPS. Para ello, las reglas no deben contener la instrucción "retract" en la parte derecha porque:

A. Al borrar los hechos no podemos calcular la g(n) necesaria para una búsqueda A*.
B. No permitiría explorar caminos alternativos al elegido en primer lugar
C. No permitiría encontrar la solución óptima
D. Ninguna de las anteriores

Sea el formato de patrón (lista [nombre^{s} edad^{s}]^{m}) para representar el nombre y la edad de un conjunto de personas. Dada una lista determinada de personas, se quiere contar el número de ellas cuya edad está comprendida entre 18 y 65 años.
Para ello se dispone del hecho que representa la lista de personas, un hecho inicial (contador 0) para contar el número de personas y la regla que se muestra a continuación.

Indica la opción CORRECTA:

```
(defrule contar
    ?f1 <-(lista $?x1 ?num $?x2)
    ?f2 <- (contador ?cont)
    (test (numberp ?num))   ;; numberp devuelve TRUE si ?num es un número
    (test (and (>= ?num 18)(<= ?num 65)))
 =>
    (retract ?f2)
    (assert (contador (+ ?cont 1))))
```

a) El SBR funciona correctamente.
b) Para que el SBR funcione correctamente es necesario añadir solo la instrucción (assert (lista $?x1 $?x2)) en la RHS de la regla.
c) Para que el SBR funcione correctamente es necesario añadir la instrucción (retract ?f1) y la instrucción (assert (lista $?x1 $?x2)) en la RHS de la regla.
d) Ninguna de las anteriores.

Dada la BH={(lista b a a a c a c b b c) (lista1 a c d e f g)} y la siguiente regla, **indica cuál será la BH final.**

```
(defrule R1
  ?f <- (lista  $?x ?a ?a $?y)
                (lista1  $? ?a $?)
  =>
            (retract ?f)
            (assert (lista $?x ?a $?y)))
```

a) {(lista b b b c) (lista1 a c d e f g)}
b) {(lista b b b c)}
c) {(lista b a c a c b b c) (lista1 a c d e f g)}
d) {(lista b a c b b c) (lista1 a c d e f g)}

Dada la base de hechos inicial: BH={(lista 5 7 3 1 6 4) (minimo 9999)} y la siguiente regla para calcular el mínimo de una lista

```
(defrule REGLA
    ?f1 <- (lista $?a ?b $?c)
    ?f2 <- (minimo ?x)
    (test (<  ?b ?x))
=>
     (assert (lista $?a $?c))
    (assert (minimo ?b)))
```

Si nuestro objetivo es obtener una base de hechos final (tras la ejecución sucesiva de la regla) en la cual el hecho (minimo …) solo puede aparecer una vez (conteniendo el valor mínimo de la lista). ¿Cuál de las siguientes afirmaciones es CIERTA para obtener nuestro objetivo?

a) La regla es correcta
b) Sería necesario añadir (retract ?f2)
c) Sería necesario añadir (retract ?f1)
d) Sería necesario añadir (retract ?f1) y (retract ?f2)

Dada la BH={(lista1 b a a a c c a c b b c)(lista2 a c)} y la siguiente regla

```
(defrule r1
  ?f <- (lista1  $?x ?a ?a $?y)
                 (lista2  $? ?a $?)
  =>
            (retract ?f)
            (assert (lista1 $?x ?a $?y)))
```

Indicad cuál será la Base de Hechos final.

a) {(lista1 b b b c) (lista2 a c)}
b) {(lista1 b a c a c b b c) (lista2 a c)}
c) {(lista1 b b b c)}
d) {(lista1 b a c a c b c) (lista2 a c)}

Ejercicio

Escribir una única regla en clips que, dada una lista de números, obtenga una nueva lista en la que se reemplace por un 0 todos aquellos números cuyo valor no coincida con su orden de posición (asumiendo que el primer número a la izquierda ocupa la posición uno).

Por ejemplo, dada la lista (lista 15 2 4 6 5 3 7 8 15), obtendría la lista (lista 0 2 0 0 5 0 7 8 0).

```
(deffacts lista (lista 1 2 4 6 5 3 7 8 9))   ;Es un ejemplo de lista


(defrule uno
      ?f <- (lista $?x ?v $?y)
      (and (test (<> ?v 0))        ;Debe controlarse, para que pueda acabar!!
      (test (<> (length $?x) (- ?v 1))))
      =>
      (retract ?f)
      (assert (lista $?x 0 $?y)))
```

Sea un SBR formado por BHinicial={(lista 2 1 6 2 3) (elemento 5)}, y la regla que se muestra a continuación. ¿Cuál sería el contenido final de la BH?

```
(defrule REGLA
  ?f <- (lista $?x ?z $?w)
        (elemento  ?y)
        (test (< ?z ?y))
  =>
      (assert (lista $?x  $?w))
      (retract ?f))
```

A. {(lista 6) (elemento 5)}
B. {(lista 2 1 2 3) (elemento 5)}
C. {(lista) (elemento 5)}
D. {(lista 2 2) (elemento 5)}

A) Dada la siguiente parte izquierda de una regla:

```
(defrule r2
   ?f <- (lista  $?   ?b   $?x   ?b   $?x)
    =>
   ...
```

y el siguiente hecho: (lista c c d c c d c c d). **¿Cuántas instancias de esta regla se insertarían en la agenda?**

a) 1
b) 2
c) 3
d) 4

B) Y en este caso?

```
(defrule r1
   (lista $?x $?w ?y ?z $?x)
   =>
```

Y el siguiente hecho: (lista a b a b c c),

A.	0
B.	1
C.	3
D.	5

Sea un SBR, con una única regla:

```
(defrule R1
   ?f <- (lista ?x $?y ?x $?z)
=>
(retract ?f)
(assert (lista $?y ?x $?z))
(printout t “La lista se ha modificado “ crlf))
```

, y la BH inicial {(lista a b a b a)}. Tras ejecutar el SBR, ¿cuántas veces se habrá mostrado en pantalla el mensaje “La lista se ha modificado “?

A.1
B.2
C.3
D.4

Dadas dos secuencias de ADN, escribe un SBR (una única regla, si es posible) que cuente el número de mutaciones (se puede utilizar una variable global para almacenar el número de aciertos o bien un hecho para registrar este valor). Por ejemplo, para las dos secuencias de ADN

Por ejemplo, entre (A A C C T C G A A A) y (A G G C T A G A A A) hay tres mutaciones.

```
(deffacts datos
        (ADN 1 A A C C T C G A A A)
        (ADN 2 A G G C T A G A A A)
        (mutations 0))

(defrule R_mutation
  ?f1 <- (ADN ?n1 $?x ?i1 $?y1)
  ?f2 <- (ADN ?n2 $?x ?i2 $?y2)
  ?f3 <- (mutations ?m)
  (test (and (neq ?n1 ?n2)(neq ?i1 ?i2)))
=>
  (retract ?f1 ?f2 ?f3)
  (assert (ADN ?n1 $?y1))
  (assert (ADN ?n2 $?y2))
  (assert (mutations (+ ?m 1))))
```

```
(defrule final
  (declare (salience -10))
  (mutations ?m)
 =>
  (printout t "No. de mutaciones es " ?m crlf))
```

Dada una BH que represente una lista de naturales no ordenados y posiblemente repetidos un número indefinido de veces cada uno de los números, escribir una ***única*** (si es posible) regla de producción que obtenga, como BH-meta, la lista inicial de números, pero en los que no haya ningún número repetido.

Ejemplo:

(lista 1 3 4 5 6 7 4 6 4 4 3 4 5 6 4 5 6 4 8 5 7 3 2 4 4 4 1 5 6 7 1 2 3 2 3 4 6 5 3 4 5 1 4 3)

y se obtendría: (lista 2 7 6 5 4 3 1 8)

```
(deffacts datos
   (Lista 1 3 4 5 6 7 4 6 4 4 3 4 5 6 4 5 6 4 8 5 7 3 2 4 4 4 1 5 6 7 1 2 3  2 3 4 6 5 3 4 5 1 4 3))


(defrule eliminarrepetidos
  ?f1 <-   (Lista $?x ?i1 $?y ?i2 $?z)
    (test (eq ?i1 ?i2))
  =>
     (retract ?f1)
     (assert (Lista $?x ?i1 $?y  $?z)))
```

Diseña un sencillo SBR (**una única regla si es posible**) para determinar el número de aciertos de los boletos sellados de la Lotería Primitiva. En la BH se dispone de un hecho que determina la combinación ganadora y otro hecho que especifica la combinación jugada (se puede utilizar una variable global para almacenar el número de aciertos o bien un hecho para registrar este valor).

Ejemplo de BHinicial: {(boleto-ganador 2 5 8 13 24 35) (combinacion 3 5 15 24 26 37)}

b) Asumiendo que los números de la combinación no se introducen en orden creciente, ¿funcionaría el S.P. que has diseñado?. ¿Porqué ?.

```
(defglobal ?*num-aciertos* = 0)

(deffacts datos
  (boleto-ganador 2 4 8 12 22 34)
  (combinacion 42 13 22 30 2 5))

(defrule cuenta-aciertos
        (boleto-ganador $?x ?y $?z)
        (combinacion $?a ?y $?b)
=>
        (bind ?*num-aciertos* (+ ?*num-aciertos* 1))
        (printout t "El numero de aciertos hasta el momento es " ?*num-aciertos* crlf))
```

Un cartón de bingo se compone de 5 líneas de 5 números cada una. La BH de un SP contiene un hecho que representa una línea ganadora, por ejemplo, (linea-ganadora 12 21 34 56 77). Escribe un hecho que represente un cartón de bingo y una regla de producción que permita determinar si hay alguna línea ganadora en el cartón.

```
(deffacts datos
(linea-ganadora 12 21 34 56 77)
(carton    linea 1 2 3 4 5
           linea 23 44 55 66 77
           linea 12 21 34 56 77
           linea 6 7 8 9 10
           linea 18 28   38 48 58))

 (defrule bingo
 (linea-ganadora $?x)
 (carton $?a linea $?x $?b)
 =>
 (printout t "Linea ganadora " crlf))
```