Universidad
Carlos III de Madrid

Grado en Ingeniería Informática

Curso 2021-2022

Ingeniería del Conocimiento

Ejercicios Básicos de CLIPS

Jorge Rodríguez Fraile 100405951 Grupo 83 100405951@alumnos.uc3m.es

1. Dado el siguiente programa en CLIPS:

```
(defrule regla-sumar1
    (declare (salience 10))
    ?a <- (elemento ?x)
    =>
    (assert (elemento (+ 1 ?x))))

(defrule regla-parar
    (declare (salience 20))
    (elemento ?x)
    (test (> ?x 99999))
    =>
    (halt))

(deffacts hechos-iniciales
    (elemento 1))
```

a) Sin ejecutar el programa en CLIPS, describe en papel que hace este programa.

Dado un único hecho, el (elemento 1), aplicará la regla regla-sumar1 que le aumentará el valor al elemento en uno y lo meterá con el nuevo valor en la base de hechos, sin sacar el previo. Repetirá el proceso de sumar uno hasta el que el valor de elemento sea mayor que 99999 es decir cuando el elemento alcance 100000 hará halt y terminará.

b) Ahora carga el programa en CLIPS, ejecuta paso a paso e inspecciona el contenido de la agenda y la base de hechos en cada paso. Asegúrate de que entiendes perfectamente como CLIPS ejecuta el programa antes de pasar a la siguiente cuestión. Si no coincide la ejecución con lo descrito en el apartado anterior, vuelve a describir que hace el programa.

Como se ha descrito antes parte de (elemento 1) y en todo momento hasta que se aplique la regla-parar en la agenda se podrá ejecutar regla-sumar1 sobre el último elemento, solo sobre este porque sobre los previos ya se ha aplicado para dar los superiores. La base de hechos se va llenando cada vez que se suma uno, pero la agenda solo tendrá una en este problema.

c) ¿Cuántas veces se activa la regla regla-sumar1 con el mismo elemento? ¿Por qué?

Solo aplicará una vez, por el principio de refracción, puesto que como no se sacan los elementos de la base de hechos se podría poner en bucle a ejecutar siempre la misma regla.

2. Dado el siguiente programa en CLIPS:

```
(defrule regla-sumar-elementos
    (declare (salience 10))
    (elemento ?x)
    (elemento ?y)
    =>
    (assert (elemento (+ ?x ?y)))
    (printout t (+ ?x ?y) crlf))

(defrule regla-parar
    (declare (salience 20))
    (elemento ?x)
    (test (> ?x 99999))
    =>
    (halt))

(deffacts hechos-iniciales
    (elemento 1))
```

a) Sin ejecutar el programa en CLIPS, describe en papel que hace este programa.

Este programa lo que hará es partiendo de un solo hecho irá generando hechos cuyo valor es la suma de otros elementos, pudiendo sumarse un elemento consigo mismo. Se parará cuando un elemento supere el valor 99999.

b) Ahora carga el programa en CLIPS, ejecuta paso a paso e inspecciona el contenido de la agenda y la base de hechos en cada paso. Asegúrate de que entiendes perfectamente como CLIPS ejecuta el programa antes de pasar a la siguiente cuestión. Si no coincide la ejecución con lo descrito en el apartado anterior, vuelve a describir que hace el programa.

Funciona como lo esperado, al principio suma el elemento consigo mismo, metiendo el nuevo valor en los hechos y sin sacar los utilizados. Por la estrategia realiza la suma de un elemento consigo mismo, el último añadido, por lo que alcanza más rápido el fin de programa, con el elemento 131072 que es el hecho 18.

c) Modifica el programa para que produzca la misma salida por pantalla, pero en cada iteración solo haya una única activación en la agenda.

Para lograr esto lo que podemos hacer es sabiendo que siempre aplica la regla sobre el elemento último elemento generado es que se haga directamente sobre sí mismo la suma, y por el principio de refracción solo habrá una. Quedaría así:

```
(defrule regla-sumar-elementos
    (declare (salience 10))
    (elemento ?x)
    =>
    (assert (elemento (+ ?x ?x)))
    (printout t (+ ?x ?x) crlf))
(defrule regla-parar
    (declare (salience 20))
    (elemento ?x)
    (test (> ?x 99999))
    =>
    (halt))
```

```
13
14  (deffacts hechos-iniciales
15      (elemento 1))
```

d) Cambia la estrategia de resolución del conjunto conflicto haciendo (set-strategy random). ¿Qué efecto tiene hacer esto en la ejecución?

Sobre el código base (sin modificación) lo que pasa es que ya no aplica siempre la suma sobre sí mismo más reciente, sino que al coger una aleatoria entre las posibles realiza más operaciones hasta llegar al final. Con respecto al caso modificado funciona igual, dado que solo hay una posible regla en todo momento, no puede escoger entre otra aleatoriamente.

3. Haz un programa en CLIPS que dada una base de hechos con un único hecho inicial del tipo (elemento <valor>), genere una base de hechos con un único hecho, con la misma estructura que el inicial, y cuyo valor sea el factorial del valor inicial.

```
(defrule inicial
    (elemento ?x)
    (not (resultado ?))
    =>
    (assert (resultado 1)))

(defrule fin
    ?x <- (elemento 1)
    ?z <- (resultado ?y)
    =>
    (assert (elemento ?y))
    (retract ?x)
    (retract ?z)
    (halt))

(defrule paso
    ?w <- (elemento ?x)
    ?z <- (resultado ?y)
    (test (<> ?x 1))
    =>
    (assert (elemento (- ?x 1)))
    (assert (resultado (* ?y ?x)))
    (retract ?w)
    (retract ?z))

(deffacts hechos-iniciales
    (elemento 5))
```

4. Dado el siguiente programa en CLIPS:

```
(deftemplate elemento
    (slot valor (type INTEGER)))

(defrule regla1
    (declare (salience 10))
    (elemento (valor ?x))
    =>
    (assert (valor ?x)))

(defrule regla2
    (declare (salience 5))
    ?a <- (valor ?x)
    (valor ?y)
    (test (< ?x ?y))
    =>
    (retract ?a))

(defrule regla3
    (declare (salience 1))
    ?a <- (valor ?x)
    =>
    (printout t "Resultado: valor " ?x crlf)
    (retract ?a))

(deffacts hechos-iniciales
    (elemento (valor 1))
    (elemento (valor 8))
    (elemento (valor 5)))
```

a) Explica brevemente que calcula el programa. Sin ejecutar el programa en CLIPS, describe en papel que hace este programa.

Este programa lo que hace es buscar el elemento de valor más bajo, para ello lo que hace es extraer los valores de los elementos e ir comparándolos y borrando el más alto, cuando ya no se pueden aplicar las otras reglas que tienen mayor prioridad termina el programa imprimiendo el valor y sacando el valor.

b) Ahora carga el programa en CLIPS, ejecuta paso a paso e inspecciona el contenido de la agenda y la base de hechos en cada paso. Asegúrate de que entiendes perfectamente como CLIPS ejecuta el programa antes de pasar a la siguiente cuestión. Si no coincide la ejecución con lo descrito en el apartado anterior, vuelve a describir que hace el programa.

Funciona como se esperaba, es decir genera los hechos valor para cada elemento y progresivamente va eliminando el mayor de entre ellos dejando solo uno al final, que permite aplicar regla3. La regla1 no se aplica más veces por el principio de refracción y es posible que siga el orden por las prioridades, que al ir eliminando hechos valor hacen que no se termine prematuramente.

c) Ejecuta los siguientes comandos y escribe lo que sale por pantalla tras la ejecución de cada uno

```
CLIPS>(reset)
CLIPS>(watch rules)
CLIPS>(matches regla1)
CLIPS>(run 1)
CLIPS>(matches regla1)
CLIPS>(matches regla2)
CLIPS>(matches regla3)
CLIPS>(run 1)
CLIPS>(matches regla1)
CLIPS>(matches regla2)
CLIPS>(matches regla3)
```

Explica que hace el comando matches.

```
CLIPS> (clear)
CLIPS> (load 4.clp)
%***$
TRUE
CLIPS> (reset)
CLIPS> (watch rules)
CLIPS> (matches regla1)
Matches for Pattern 1
f-1
f-2
f-3
Activations
f-3
f-2
f-1
(3 0 3)
CLIPS> (run 1)
FIRE    1 regla1: f-3
CLIPS> (matches regla1)
Matches for Pattern 1
f-1
f-2
f-3
Activations
f-2
f-1
(3 0 2)
CLIPS> (matches regla2)
Matches for Pattern 1
f-4
Matches for Pattern 2
f-4
Partial matches for CEs 1 - 2
    None
Activations
    None
(2 0 0)
CLIPS> (matches regla3)
```

```
Matches for Pattern 1
f-4
Activations
f-4
(1 0 1)
CLIPS> (run 1)
FIRE    1 regla1: f-2
CLIPS> (matches regla1)
Matches for Pattern 1
f-1
f-2
f-3
Activations
f-1
(3 0 1)
CLIPS> (matches regla2)
Matches for Pattern 1
f-4
f-5
Matches for Pattern 2
f-4
f-5
Partial matches for CEs 1 - 2
f-4,f-5
Activations
f-4,f-5
(4 1 1)
CLIPS> (matches regla3)
Matches for Pattern 1
f-4
f-5
Activations
f-5
f-4
(2 0 2)
```

El comando matches lo que hace es mostrar cuáles son aquello hechos que coinciden con el patrón definido por la precondición de la regla. También muestra si este patrón tiene varias variables que candidatos hay para cada uno y sus combinaciones. Por último las activaciones posibles en el estado actual.

5. Dado el siguiente programa en CLIPS:

```
(defrule R1
    (declare (salience 15))
    ?a <- (numero ?x ?u)
    ?b <- (numero ?y ?v)
    (test (> ?u ?v))
    =>
    (assert (numero (+ ?x ?y) (+ ?u 1)))
    (retract ?b))

(defrule R2
    (declare (salience 5))
    ?b <-(total ?x)
    (test (> ?x 0))
    =>
    (assert (numero 0 1)))

(defrule R3
    (declare (salience 5))
    ?b <-(total ?x)
    (test (> ?x 1))
    =>
    (assert (numero 1 2)))

(defrule R4
    (declare (salience 20))
    (total ?a)
    (numero ?x ?a)
    =>
    (printout t "OK:" ?x crlf)
    (halt))

(defrule R5
    (declare (salience 1))
    =>
    (printout t "ERROR" crlf))
```

a) Explica brevemente para qué sirve este programa.

Este programa tiene como hechos total, que indica cuantas veces se sumará un número con su predecesor, y número en forma de par de valores, uno la suma hasta él y el otro la posición que ocupa ene orden, lo que hace inicialmente es partir de 0 para ir paso a paso sumando su valor con el anterior y aumentar su índice.

b) Ahora carga el programa en CLIPS. Introduce en la base de hechos el hecho (total 5). Ejecuta paso a paso e inspecciona el contenido de la agenda y la base de hechos en cada paso. Asegúrate de que entiendes perfectamente como CLIPS ejecuta el programa antes de pasar a la siguiente cuestión. Si no coincide la ejecución con lo descrito en el apartado anterior, vuelve a describir que hace el programa.

Como se esperaba parte de 0 1 y 1 2 a partir de ahí va con la R1 sumando los primeros, quedándose con el de mayor posición e iterándolo, hasta que llega el punto que el total es el iterador y termina.

c) Escribe que saldría por pantalla si hiciéramos:

```
CLIPS> (reset)
CLIPS> (assert (total 6))
CLIPS> (run)
```

¿Cuántos ciclos son necesarios para que el programa se tenga?

```
CLIPS> (clear)
CLIPS> (load 5.clp)
*****
TRUE
CLIPS> (reset)
CLIPS> (assert (total 6))
<Fact-1>
CLIPS> (run)
OK:5
CLIPS>
```

Necesitará 7 ciclos, 2 para empezar y que tenga el valor 2, otros 4 hasta que alcance el 6 y otro final que termine el programa e imprima.

d) Responder a las mismas preguntas del apartado anterior cambiando el (assert (total 6)), por (assert (total 7)) y por (assert (total 8))

Para el (total 7) necesita 8 y para el (total 8) necesitará 9, solo varía en que entremedias necesitan 1 o 2 ciclos más.

e) Y si en vez de (assert (total 5)) hiciéramos (assert (total -1))

Saltaría ERROR por la regla R5, ya que no se podría aplicar ni R2 que es la inicial.

f) Describe 2 casos de uso del comando (matches) con reglas que se pueden disparar y otros 2 casos con reglas que no cumplen todas las precondiciones

En el caso inicial se pueden lanzar la regla R2, la R3 y la R5, dado que la R5 siempre se puede lanzar y la R2 y R3 al ser el caso inicial y no haberse disparado todavía.

También en el caso inicial no se pueden lanzar la R1 al no haber ningún número todavía y tampoco la R4 porque tenemos total, pero no un número con el valor de total.

6. Escribe un programa CLIPS (reglas y clases necesarias) para calcular la duración media de todas las actividades que una persona realiza durante la visita a una ciudad. Suponiendo que la persona hace todas las actividades de la ciudad en la que esta y que los datos vienen representados de la siguiente forma:

```
(definstances personas
    (of persona (nombre Juan) (ciudad Paris))
    (of persona (nombre Ana) (ciudad Edimburgo)))

(definstances actividades
    (of actividad (nombre Torre_Eiffel) (ciudad Paris) (duracion 2))
    (of actividad (nombre Castillo_de_Edimburgo) (ciudad Edimburgo) (duracion 5))
    (of actividad (nombre Louvre) (ciudad Paris) (duracion 6))
    (of actividad (nombre Montmartre) (ciudad Paris) (duracion 1))
    (of actividad (nombre Royal_Mile) (ciudad Edimburgo) (duracion 3)))
```

Después de la ejecución deberíamos tener la siguiente salida:
La duración media de las actividades de Ana fue 4.0
La duración media de las actividades de Juan fue 3.0

```
(defclass persona (is-a USER)
    (slot nombre
        (type STRING))
    (slot ciudad
        (type STRING)))

(defclass actividad (is-a USER)
    (slot nombre
        (type STRING))
    (slot ciudad
        (type STRING))
    (slot duracion
        (type INTEGER)))

(defrule iniciar-persona
    (declare (salience 30))
    (object (is-a persona) (nombre ?x) (ciudad ?y))
    =>
    (assert (suma ?x 0))
    (assert (contador ?x 0)))

(defrule asig-act-per
    (declare (salience 10))
    (object (is-a persona) (nombre ?x) (ciudad ?y))
    (object (is-a actividad) (nombre ?v) (ciudad ?y) (duracion ?w))
    =>
    (assert (act-per ?x ?v ?y ?w)))

 (defrule sumar-actividad
    (declare (salience 20))
    (object (is-a persona) (nombre ?x) (ciudad ?y))
    ?c <- (act-per ?x ? ?y ?w)
    ?a <- (suma ?x ?z)
    ?b <- (contador ?x ?v)
```

```
35      =>
36      (assert (suma ?x (+ ?z ?w)))
37      (assert (contador ?x (+ ?v 1)))
38      (retract ?a)
39      (retract ?b)
40      (retract ?c))
41
42  (defrule imprimir-media
43      (declare (salience 5))
44      (object (is-a persona) (nombre ?x) (ciudad ?y))
45      ?a <- (suma ?x ?z)
46      ?b <- (contador ?x ?v)
47      =>
48      (printout t "La duracion media de las actividades de " ?x " fue " (/ ?z ?v) crlf)
49      (retract ?a)
50      (retract ?b))
51
52  (defrule fin
53      (declare (salience 1))
54      =>
55      (halt))
56
57  (definstances personas
58      (of persona (nombre Juan) (ciudad Paris))
59      (of persona (nombre Ana) (ciudad Edimburgo)))
60
61  (definstances actividades
62      (of actividad (nombre Torre_Eiffel) (ciudad Paris) (duracion 2))
63      (of actividad (nombre Castillo_de_Edimburgo) (ciudad Edimburgo) (duracion 5))
64      (of actividad (nombre Louvre) (ciudad Paris) (duracion 6))
65      (of actividad (nombre Montmartre) (ciudad Paris) (duracion 1))
66      (of actividad (nombre Royal_Mile) (ciudad Edimburgo) (duracion 3)))
```

7. Dado el siguiente programa en CLIPS:

```
(deftemplate elemento
    (slot valor (type INTEGER)))

(defrule R1
    (declare (salience 40))
    (elemento (valor ?x))
    (not (inicial ?))
    (test (>= ?x 0))
    =>
    (assert (inicial ?x)))

(defrule R2
    (declare (salience 20))
    (elemento (valor ?x))
    (not (elemento (valor 2)))
    (not (borrando))
    (test (> ?x 2))
    =>
    (assert (elemento (valor (- ?x 1)))))

(defrule R3
    (declare (salience 15))
    ?a <- (elemento (valor ?x))
    ?b <- (elemento (valor ?y))
    (test (> ?x ?y))
    =>
    (assert (borrando))
    (assert (elemento (valor (* ?x ?y))))
    (retract ?a ?b))

(defrule R4
    (declare (salience 30))
    ?a <- (elemento (valor 0))
    =>
    (retract ?a)
    (assert (elemento (valor 1))))

(defrule R5
    (declare (salience 10))
    (inicial ?x)
    (elemento (valor ?y))
    (test (>= ?y 0))
    =>
    (printout t "OK:" "(" ?x "," ?y ")" crlf)
    (halt))

(defrule R6
    (declare (salience 1))
    =>
    (printout t "ERROR" crlf))
```

a) Explica brevemente para qué sirve este programa.

Este programa calcula el factorial del elemento que se da inicio, que pasa a ser inicial, después genera elementos de valor descendente hasta llegar a 2 (ya que multiplicar por 1 da lo mismo) y finalmente va multiplicando los elementos por parejas, elimina los utilizados y crea uno nuevo que es el acumulado.

b) Ahora carga el programa en CLIPS. Introduce en la base de hechos el hecho (elemento (valor 3)). Ejecuta paso a paso e inspecciona el contenido de la agenda y la base de hechos en cada paso. Asegúrate de que entiendes perfectamente como CLIPS ejecuta el programa antes de pasar a la siguiente cuestión. Si no coincide la ejecución con lo descrito en el apartado anterior, vuelve a describir que hace el programa.

Es como se esperaba, es un diseño ingenioso para el cálculo de factoriales en el que se utilizan variables que habilitan reglas como es inicial para R5 o limitando como borrando a R2.

c) Escribe que saldría por pantalla si ejecutamos cada uno de los comandos:

```
CLIPS>(reset)
CLIPS>(assert (elemento (valor 3))
CLIPS>(matches R1)
CLIPS>(matches R3)
CLIPS>(matches R6)
```

Explica el significado de cada una de las ejecuciones anteriores del comando matches

```
CLIPS> (clear)
CLIPS> (load 7.clp)
%******
TRUE
CLIPS> (reset)
CLIPS> (assert (elemento (valor 3)))
<Fact-1>
CLIPS> (matches R1)
Matches for Pattern 1
f-1
Matches for Pattern 2
 None
Partial matches for CEs 1 - 2
f-1,*
Activations
f-1,*
(1 1 1)
CLIPS> (matches R3)
Matches for Pattern 1
f-1
Matches for Pattern 2
f-1
Partial matches for CEs 1 - 2
 None
Activations
 None
(2 0 0)
CLIPS> (matches R6)
Matches for Pattern 1
*
```

```
Activations
*
(1 0 1)
```

El (reset) y (assert (assert (elemento (valor 3))) permiten inicial la base de hechos del problema.

El (matches R1) nos muestra que solo se puede emplear el hecho creado y que para la ? puede ser cualquier cosa, ya que no existe inicial.

El (matches R3) nos dice que el hecho es candidato para ocupar cualquiera de las dos posiciones, pero que no hay ninguna tupla de 2 variables que lo cumpla, ya que tiene que ser una mayor que la otra.

El (matches R6) nos dice que siempre es aplicable con el * dado que no tiene precondición más que se llegue a mínima prioridad.