# *Técnicas de Inteligencia Artificial – Laboratorio 6*

Requisitos previos:

El alumno ha de saber manejar CLIPS y crear funciones simples.

Objetivos:

Este laboratorio se centrará en seguir adquiriendo conocimientos sobre CLIPS. En este caso, en vez de utilizar la programación funcional, se introducirá la programación heurística.

Al finalizar este laboratorio, el alumno deberá ser capaz de:

- Desarrollar programas utilizando reglas en CLIPS.

Motivación:

A lo largo del curso se utilizará CLIPS para ir desarrollando sistemas expertos, por lo que es imprescindible adquirir el conocimiento del uso de CLIPS y todas las herramientas que proporciona.

# ***Tarea 1: Hechos y Reglas***

En esta primera y única tarea deberéis realizar los siguientes puntos y responder a las preguntas. Una vez realizada la tarea, es imprescindible entregar el laboratorio. Para ello, copiar todos los ficheros CLP dentro de una carpeta, comprimirlo y subirlo a egela.

**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 9))
=>
        (halt))

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

**Cuestiones:**

**1. Sin ejecutar el programa en CLIPS, describe en papel qué hace este programa.**

- Como el elemento es 9 o menor aplicara la primera regla incrementando el valor del elemento hasta que sea 10. En ese instante, ejecutara la regla parar debido a que su prioridad es mayor y cumple la condición parando así la ejecución del programa.

- Incrementa en uno el valor del elemento hasta que este sea mayor que nueve.

**2. 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.**

- Coincide con la descripción dada anteriormente en el apartado 1.

**3. ¿Cuántas veces se activa la regla regla-sumar1 con el mismo elemento?. ¿Por qué?**

- La regla se activa 9 veces, aunque únicamente 1 vez por el mismo elemento, dado que cada vez se introduce un nuevo “fact” como elemento.

## 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 9))
=>
        (halt))

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

**Cuestiones:**

**1. Sin ejecutar el programa en CLIPS, describe en papel qué hace este programa.**

- Como el elemento es 9 o menor aplicara la primera regla duplicando el valor del elemento hasta que sea 16, mayor que nueve. En ese instante, ejecutara la regla parar debido a que su prioridad es mayor y cumple la condición parando así la ejecución del programa. En otras palabras incrementa el número de la base de hechos hasta que este sea mayor que nueve.

- Duplica el valor del número hasta que este sea mayor que 9.

**2. 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.**

- Coincide con la descripción dada anteriormente en el apartado 1.

**3. Modifica el programa para que produzca la misma salida por pantalla, pero en cada iteración sólo haya una única activación en la agenda.**

- Basta con cambiar la regla de sumar elementos

```
(defrule regla-sumar-elementos
        (declare (salience 10))
        ?f1<-(elemento ?x)
        ?f2<-(elemento ?y)
=>
        (retract ?f1 ?f2)
        (assert (elemento (+ ?x ?y)))
        (printout t (+ ?x ?y) crlf))
```

## 3. 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)))
```

### 1. Explica brevemente qué calcula el programa. Sin ejecutar el programa en CLIPS, describe en papel qué hace este programa.

- Lee la plantilla, sacando los valores añadiéndolo a la base de hechos, para posteriormente

compararlos y eliminar los menores y quedarse con el valor del elemento mayor para mostrarlo por pantalla.

- Muestra por pantalla el elemento mayor de los introducidos.

**2. 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.**

- Coincide con la descripción dada anteriormente en el apartado 1.

**3. Ejecuta los siguientes comandos y escribe lo que sale por pantalla tras la ejecución de cada uno.**

CLIPS> (reset) -> nada
CLIPS> (watch rules) -> nada
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)

## 3. Explica qué hace el comando matches.

Indica cuales son los elementos que cumplen la regla y señalando las agrupaciones de los mismo.

## 4. 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))
```

## 1. Explica brevemente para qué sirve este programa.

- Devuelve el valor que tiene asignada la posición del número introducido en la serie de Fibonacci. Donde comenzando con 0 y 1, se van consiguiendo los números siguientes al realizar la suma de los dos elementos inmediatamente anteriores a él.

**2. 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.**

**3. Escribe a la derecha qué saldría por pantalla si hiciéramos:**
**CLIPS> (reset)** No saldría nada;
**CLIPS> (assert (total 6))** Saldria que se ha añadido el fact;
**CLIPS> (run)** Ok 5

**¿Cuantos ciclos son necesarios para que el programa se detenga?**

5 ciclos

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

Con assert 7 son necesarios 8 ciclos y con assert 8 son necesarios 9 ciclos.

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

1 ciclo ya que nos daría ERROR porque se ejecutaria R5 ya que es la única que cuadra con ese hehco, las otras o necesitan un hecho tal que (numero x y) o el total ha de ser mayor que 0.

**5. Ejercicio**

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

```
(deffacts personas
    (persona (nombre Juan) (ciudad Paris))
    (persona (nombre Ana) (ciudad Edimburgo)))

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

**Después de la ejecución tendríamos que 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

Nuestro código:

```
(deftemplate persona
	(slot nombre (type SYMBOL))
	(slot ciudad(type SYMBOL))
	(slot DuracionTotAct (type INTEGER))
	(slot ActRealizadas (type INTEGER))

)

(deftemplate actividad
	(slot nombre (type SYMBOL))
	(slot ciudad(type SYMBOL))
	(slot duracion (type INTEGER))
)

(deffacts personas
	(persona (nombre Juan) (ciudad Paris))
	(persona (nombre Ana) (ciudad Edimburgo))
)

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

(defrule añadirACT
	(declare (salience 50))
	?f1 <- (persona(nombre ?a) (ciudad ?b) (DuracionTotAct ?d) (ActRealizadas ?e))
	?f2 <- (actividad (nombre ?) (ciudad ?b) (duracion ?c))
=>
	(retract ?f1)
	(retract ?f2)
	(assert (persona(nombre ?a)(ciudad ?b)(DuracionTotAct (+ ?d ?c))(ActRealizadas (+ ?e 1)) ))
)

(defrule print
	(declare (salience 5))
	?f1 <- (persona(nombre ?a) (ciudad ?) (DuracionTotAct ?d) (ActRealizadas ?e))
=>
	(printout t "La media de las actividades de " ?a " fue " (/ ?d ?e) crlf)
)

(defrule parar
	(declare (salience 1))
=>
	(halt)
)
```

**6. Ejercicio**

**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)
        (retract ?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))
```

**1. Explica brevemente para qué sirve este programa.**

Sirve para calcular el factorial de un número.

**2. 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.**

**3. Escribe qué saldría por pantalla si ejecutamos cada uno de los comandos:**

**CLIPS>(reset)** no ocurre nada
**CLIPS>(assert (elemento (valor 3)) )** se añade el <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
None
Partial matches for Ces 1-2
none
Activations
None
(2 0 0)
```

## CLIPS>(matches R6)

Matches for Pattern 1
*
Activations
*
(1 0 1)

## Explica el significado de cada uno de las ejecuciones anteriores del comando matches.

El comando matches indica cuales son los elementos que cumplen la regla y señalando las agrupaciones de los mismo.

Matches R1

Nos dice que el fact 1 encaja con el patrón 1, con el patrón dos no encaja ningún fact. La activación parcial de las condiciones se da con fact 1 , y *.
Se activa la regla con f1 ya que al no haber ningún fact de la condición 2 se cumple esta condición.

Matches R3

Nos dice que el fact 1 encaja con el patrón 1, con el patrón dos no encaja ningún fact. La activación parcial de las condiciones no se puede dar en ningún caso y es por esto que no se activara la regla.

Matches R6

Nos dice que el patrón uno se cumple con cualquier caso y por tanto se activa la regla.