REPRESENTACIÓN DEL CONOCIMIENTO - CLIPS

Práctica 2

Objetivos:

1. Expresiones y restricciones de campos
2. Evaluación de expresiones lógicas
3. Variables locales y globales
4. Definición de funciones
5. Declaración de la prioridad de una regla

1. EXPRESIONES

Las expresiones aritméticas siguen la notación prefija. Las operaciones se calculan de izquierda a derecha, no hay orden de precedencia. La precedencia se establece por la anidación de paréntesis:

- (+ 2 3 6 7 7)
- (/ ?Y ?X)
- (> (/ 5 2) 3)
- (* 3 4 5)

Calcular área de un rectángulo.

```
(deftemplate rectangulo
              (slot altura)
              (slot base))

(defrule calcular_area
        (rectangulo (altura ?altura) (base ?base))
=>(assert (area (* ?base ?altura))))
```

2. RESTRICCIONES DE CAMPO

Las restricciones de campo permiten establecer un filtro en los atributos de los hechos que se satisfacen en el antecedente de una regla:

Suponiendo la siguiente plantilla:

```
(deftemplate persona
     (slot nombre)
     (slot cabello (allowed-values rubio moreno castagno blanco negro))
     (slot ojos (allowed-values negros verdes azules)))
```

Podemos realizar los siguientes asertos para probar reglas sobre las restricciones de campo:

```
(assert (persona (nombre Claudia) (cabello rubio) (ojos azules)))
(assert (persona (nombre Juan) (cabello moreno) (ojos verdes)))
(assert (persona (nombre Elena) (cabello blanco) (ojos negros)))
(assert (persona (nombre Jorge) (cabello castagno) (ojos negros)))
(assert (persona (nombre Pascual) (cabello moreno) (ojos negros)))
```

Restricción Not : ~ Niega el resultado de la siguiente restricción

```
(defrule persona-no-moreno
      (persona  (nombre ?nombre)
                (cabello ~moreno))   ;;; restricción NOT
      =>
      (printout t ?nombre " no tiene el cabello moreno" crlf))
```

Restricción Or : | permite que varias opciones puedan coincidir como campo de un patrón

```
(defrule persona-morena-o-castagno
      (persona  (nombre ?nombre)
                (cabello castagno| moreno))  ;;; restriccion OR
      =>
      (printout t ?nombre "  tiene el cabello oscuro" crlf))
```

Restricción And : & permite especificar más de una restricción a un atributo

```
(defrule persona-ni-morena-ni-castagna
      (persona  (nombre ?nombre) (cabello ?color&~castagno&~moreno))
      =>
      (printout t ?nombre "  tiene el cabello " ?color crlf))
```

Ejemplo: combinación de las restricciones de campo

```
(defrule comparacion-compleja-ojos-cabello
      (persona (nombre ?nomA)
               (ojos ?ojosA &azules | verdes)
               (cabello ?cabelloA &~negro))
      (persona (nombre ?nomB &~?nomA)
               (ojos ?ojosB &~?ojosA)
               (cabello ?cabelloB&blanco | ?cabelloA))
=>
(printout t ?nomA " tiene ojos " ?ojosA " y pelo " ?cabelloA crlf)
(printout t ?nomB " tiene ojos " ?ojosB " y pelo " ?cabelloB crlf))
```

3. TEST

Se usa en los antecedentes de las reglas para realizar comprobaciones lógicas: **TEST** junto con:

- Operadores de comparación: = , <>, < >, <=, <= eq neq
- Funciones lógicas: **or not and**
- Funciones de predicado:

(**evenp** <arg>) número par
(**oddp** <arg>) número impar
(**floatp** <arg>)
(**integerp** <arg>)
(**lexemep** <arg>) símbolo o string
(**numberp** <arg>) float o entero
(**sequencep** (oddp <arg>)) valor de un campo múltiple
(**stringp** <arg>)
(**symbolp** <arg>)

```
(defrule testeando
    (dato ?x)
    (test ( = ?x  8))
  =>
(printout t ?x " es igual a 8 "
crlf ))

(assert (dato 8))
(assert (dato 18))
(run)
```

```
(defrule comprobando
   (respuesta ?res)
   (lexemep ?res)
=>(printout "la respuesta es un
símbolo o una cadena");
```

4. Otros operadores lógicos

Existen otros operadores para combinar la lista de hechos que una regla ha de satisfacer, tales como OR, AND, NOT, LOGICAL, EXISTS, FORALL.

⇒ De entre ellos **evitaremos el uso del operador OR** que puede ser sustituido por 2 o más reglas independientes de forma que se controlen mejor los efectos laterales.
Por ejemplo:

```
(defrule abuelo
(padre-de (padre ?a) (hijo ?x))
(or (padre-de(padre ?x)(hijo ?n))
    (madre-de(madre ?x)(hijo ?n)))
 =>
(printout t ?a " es el abuelo de "
?n crlf))


;; Esta regla se sustituye por 2
reglas independientes
```

```
(defrule abuelo_paterno
   (padre-de (padre ?a)(hijo ?x))
   (padre-de (padre ?x)(hijo ?n))
 =>
(printout t ?a " es el abuelo
paterno de " ?n crlf))

 (defrule abuelo_materno
   (padre-de (padre ?a)(hijo ?x))
   (madre-de (madre ?x)(hijo ?n))
 =>
(printout t ?a " es el abuelo
materno de " ?n crlf))
```

5. VARIABLES TEMPORALES

Para almacenar resultados temporales se utiliza la función **bind**, que asigna un valor a una determinada variable:

(bind <variable> valor)

```
(defrule areayperime
  (circulo ?radio)
=>(bind ?pi 3.14159)
   (printout t "Area del circulo " (* ?pi ?radio ?radio) crlf)
   (printout t "Perimetro " (* 2 ?pi ?radio) crlf ))

(assert (circulo 3))
```

6. VARIABLES GLOBALES

Las variables globales en CLIPS se denotan comenzando por interrogación y asterisco, y acaban en un asterisco: ?*nomvar*

(defglobal <var_global> **=** <valor>)

```
(defglobal  ?*saludo* = "hola"   ;; cadena de texto
            ?*verbo1*  = (create$ soy eres es somos sois son ser)) ;; valor multicampo
```

⇒ *Ver lista de órdenes para cadenas de texto y valores multicampo*

7. DEFINICIÓN DE FUNCIONES

(**deffunction** <function-name> [comentario opcional]
(?arg1 ?arg2 ... [$?argN] ; lista de argumentos el último puede ser un argumento multicampo
(<accción 1>
<accción 2>
...
<acción K>) ; la única que puede devolver un valor, ninguna de las anteriores puede hacerlo

```
(deffunction hipotenusa  (?a  ?b)              ; lista de argumentos
      (sqrt (+ (* ?a ?a)  (* ?b  ?b)) ) )

(defrule calculahipotenusa
      (dimensiones ?base ?altura)
=>    (printout t "     El valor de la hipotenusa es  "  (hipotenusa
?base ?altura)  crlf) )

(assert (dimensiones 3 4))
```

8. DECLARACIÓN DE LA PRIORIDAD EN REGLAS

Para establecer prioridades entre reglas que permitan resolver conflictos a la hora de determinar qué regla se ejecutará primero, se realiza mediante la orden:

(declare (salience <valor>))

El valor puede oscilar entre -10000 y 10000, por defecto las reglas tienen prioridad 0.
Así en el siguiente ejemplo, ¿Qué regla se ejecutará primero?

(defrule R1
⇨ **(printout t "**R1 ejecutándose**"))**

(defrule R2
(declare (salience 10**))**
⇨ **(printout t "**R2 ejecutándose**"))**

En la sesion de hoy debes entregar la definición de las plantillas y de las primeras 3 reglas de la siguiente práctica.

CAJERO AUTOMÁTICO

Sea un sistema para permitir la obtención de dinero a través de un cajero automático mediante tarjeta de crédito. Se usarán diferentes reglas para validar la tarjeta de un usuario y para concederle el dinero pedido cuando se cumplan una serie de condiciones.
Usaremos tres hechos estructurados:

1. Usuario que representa el usuario que accede al cajero y teclea el pin y la cantidad de dinero que desea obtener:
   Usuario: DNI Titular, Pin, Dinero que desea obtener
2. Tarjeta que contiene la información que tendría la tarjeta:
   Tarjeta: Pin, DNI Titular, Nº intentos, Límite de dinero autorizado a extraer en un mismo día, Año de expiración de la tarjeta (por simplificar no tenemos en cuenta ni mes ni día) , y un slot que indica si la tarjeta está Validada o no (valores permitidos Si o No). Todos los campos menos el pin y el DNI deben tener asociado un valor inicial por defecto.
3. Cuenta representa la información de la cuenta asociada a la tarjeta con el saldo actual:
   Cuenta: DNI Titular, Saldo de la cuenta, y un estado con los posibles valores: enPantalla (mostrado el saldo en pantalla), dineroEntregado, Inicial (valor por defecto), SuperaLimite y SinSaldo.

El proceso consistiría en que el usuario que se acerca a un cajero insertaría hechos de la forma:
(usuario (DNI 123456) (Pin 1212) (Cantidad 300))

Internamente en esta simulación deberíamos tener información sobre la tarjeta y la cuenta asociada a esta persona, para ello tendremos unos hechos iniciales del tipo:
(tarjeta (DNI 1234567) (Pin 1212) (Intentos 3) (Limite 500) (Anno 2018))
(cuenta (DNI 1234567) (Saldo 5000))

El proceso se puede descomponer en dos pasos principales:

- **Validación de la Tarjeta:** donde tendremos 3 reglas principales:
  1. **Regla Supera_Intentos:** Cuando se supera el límite se informa mediante un mensaje. El número de intentos debe verificarse antes que cualquier otro requisito.
  2. **Regla Pin_Invalido:** Cuando no se valida el Pin se informa mediante mensaje**.**
  3. **Regla Valida_Tarjeta:** La validacion consiste en 3 pasos: valida intentos, valida fecha, valida pin.

- **Comprobación de Saldo y Entrega del dinero**:
  1. **Regla Muestra_Saldo:** cuando la tarjeta se ha validado se muestra el saldo en pantalla, y se actualiza el estado de la cuenta (enPantalla).
  2. **Regla Saldo_NoSuficiente:** Si no tiene saldo muestra mensaje en pantalla.
  3. **Regla Comprueba_Limite1:** Si supera el límite establecido por el banco muestra mensaje en pantalla.
  4. **Regla Comprueba_Limite2**: Si supera el límite establecido por la tarjeta muestra mensaje en pantalla.
  5. **Regla Entrega_Dinero:** Muestra mensaje con el nuevo saldo y la cuenta pasa al estado DineroEntregado. Si el cajero da el dinero al usuario se almacenará internamente este nuevo saldo.

1. **Implementa en CLIPS este sistema, creando al menos dos funciones para realizar los cálculos aritméticos: Decrementar un valor en 1 y Calcular la diferencia entre dos valores (aunque sean muy sencillos y puedan realizarse directamente).**
   Los hechos deben ser estructurados, y con las especificaciones del enunciado.
2. **Realiza una traza de ejecución de este sistema contemplando los distintos casos:** superar el número de intentos, pin invalido, acceso correcto y entrega del dinero correctas, acceso correcto y no entrega del dinero**. Indica en cada caso cómo se va actualizando la base de hechos y la agenda.**

** Para comprobar la fecha usaremos una variable global ?*ANNO=2015

Asertos sobre los usuarios que acceden al sistema:

(usuario (DNI 123456) (Pin 1212) (Cantidad 300))
(usuario (DNI 456456) (Pin 1211) (Cantidad 200))
(usuario (DNI 456456) (Pin 4545) (Cantidad 3000))

Para facilitar esta simulación, los datos de las tarjetas y de las cuentas suponemos que se encuentran cargados en el sistema (usa deffacts para definir los siguientes hechos iniciales y algunos más que necesites).

(tarjeta (DNI 123456) (Pin 1212) (Intentos 3) (Limite 500) (Anno 2015))
(tarjeta (DNI 456456) (Pin 4545) (Intentos 3) (Limite 500) (Anno 2015)
(tarjeta (DNI 000111) (Pin 0011) (Intentos 0) (Limite 500) (Anno 2015)

(cuenta (DNI 1234567) (Saldo 5000))
(cuenta (DNI 456456) (Saldo 33))
(cuenta (DNI 000111) (Saldo 30000))