IreneCR2
www.wuolah.com/student/IreneCR2

7834

Ejercicios Hechos y Reglas.pdf

EJERCICIOS CLIPS

2º Sistemas Inteligentes

Grado en Ingeniería Informática

Escuela Politécnica Superior de Córdoba
UCO - Universidad de Córdoba

```
;====== RELACIÓN HECHOS ======

;=== Ejercicio 1 ===
;Define una plantilla 'persona' con cinco casillas simples (nombre
(string), apellidos (string), color-ojos (symbol), altura (float) y
edad (integer)) y valor por defecto para todos excepto nombre y
apellidos.

(deftemplate persona
      (slot nombre (type STRING))
      (slot apellidos (type STRING))
      (slot color-ojos (type SYMBOL) (default verde))
      (slot altura (type FLOAT) (default 1.65))
      (slot edad (type INTEGER) (default 20))
)


;=== Ejercicio 2 ===
;Representa con hechos no ordenados los datos de varios paciente
(nombre, apellidos, dni y seguro médico) y las visitas que realizan
(fecha, síntomas, pruebas y medicación).

(deftemplate datosPaciente
      (slot nombre (type STRING))
      (slot apellidos (type STRING))
      (slot dni)
      (slot seguroMedico)
      (slot fechaVisita)
      (multislot sintomas)
      (multislot pruebas)
      (multislot medicacion)
)


;=== Ejercicio 3 ===
;Describe la información de la figura acerca de trayectos aéreos
mediante un conjunto de hechos no ordenados. Usa una única plantilla.
(deftemplate trayecto
      (slot nombreCiudad)
      (multislot ciudadesLlegada);Desde las que llega a esta ciudad
      (multislot ciudadesSalida);Hacia las que sale desde esta ciudad
)
```

```
(deffacts ciudades
      (trayecto (nombreCiudad Lisboa) (ciudadesLlegada Roma)
(ciudadesSalida Paris Madrid))
      (trayecto (nombreCiudad Paris) (ciudadesLlegada Lisboa
Estocolmo) (ciudadesSalida Roma))
      (trayecto (nombreCiudad Estocolmo) (ciudadesLlegada Frankfurt)
(ciudadesSalida Paris))
      (trayecto (nombreCiudad Madrid) (ciudadesLlegada Lisboa Roma))
      (trayecto (nombreCiudad Roma) (ciudadesLlegada Paris Frankfurt)
(ciudadesSalida Lisboa Madrid))
)


;=== Ejercicio 4 ===
;Representa mediante hechos no ordenados los datos de los miembros de
una familia y sus relaciones familiares.

;Primero una plantilla para representar personas
(deftemplate persona
      (slot nombre)
      (slot apellidos)
      (slot dni)
)

;Plantilla para representar relaciones familiares
(deftemplate relFam
      (slot tipo)
      (slot dni-1)
      (slot dni-2)
)

;Declaramos las personas
(deffacts personas
      (persona (nombre Juan) (apellidos Fernandez) (dni 1))
      (persona (nombre Pedro) (apellidos Fernandez) (dni 2))
)

;Declaramos las relaciones familiares
(deffacts relaciones
      (relFam (tipo hijo_de) (dni-1 1) (dni-2 2))
)

;Regla para presentar los datos de las personas por pantalla
(defrule presentaDatos
```

```
      (declare (salience 10));Prioridad 10 para que se ejecute antes
      (persona (nombre ?n) (apellidos ?ap))
=>
      (printout t ?n " " ?ap crlf)
)

;Regla para presentar las relaciones familiares por pantalla
(defrule presentaRelacion
      (relFam (tipo ?rel) (dni-1 ?dn1) (dni-2 ?dn2))
      (persona (nombre ?n1) (apellidos ?ap1) (dni ?dn1))
      (persona (nombre ?n2) (apellidos ?ap2) (dni ?dn2))
      (test (neq ?dn1 ?dn2))
=>
      (printout t ?n1 " " ?ap1 " es " ?rel " " ?n2 " " ?ap2 crlf)
)

;Inferimos una relación familiar a partir de los datos. Si Juan es
hijo de Pedro, Pedro es padre de Juan
(defrule relPadreHijo
      (relFam (tipo hijo_de) (dni-1 ?dn1) (dni-2 ?dn2))
      (persona (nombre ?n1) (apellidos ?ap1) (dni ?dn1))
      (persona (nombre ?n2) (apellidos ?ap2) (dni ?dn2))
      (test (neq ?dn1 ?dn2))
=>
      (assert (relFam (tipo padre_de) (dni-1 ?dn2) (dni-2 ?dn1)))
)



;=== Ejercicio 4b ===
;Programa para representar los datos de las personas ordenados
alfabéticamente
(deftemplate persona
      (slot nombre)
      (slot apellidos)
      (slot ordenado (type INTEGER) (default 0))
)

(deffacts personas
      (persona (nombre Juan) (apellidos Fernández))
      (persona (nombre Pedro) (apellidos Rodríguez))
      (persona (nombre Miguel) (apellidos Borrego))
      (persona (nombre Ana) (apellidos Aguilera))
)
```

```
(defrule presentaOrden
      (declare (salience 5))
      ?p<-(persona (nombre ?n) (apellidos ?ap) (ordenado 0))
      (not (persona (apellidos ?y&:(eq (str-compare ?y ?ap) -1))
(ordenado 0))) ;Que no haya otro apellido que vaya antes
      ;También (eq (str-compare ?ap ?y) 1)
=>
      (printout t ?n " " ?ap crlf)
      (modify ?p (ordenado 1))
)



;====== RELACIÓN REGLAS ======

;=== Ejercicios 1 y 2 ===
;Programa que en base a un hecho que indique el sonido que hace un
animal identifique qué animal es.

(deftemplate animal
      (slot nombre)
      (slot sonido)
)

(deffacts sonidoAnimal
      (animal (nombre perro) (sonido guau))
      (animal (nombre gato) (sonido miau))
      (animal (nombre pato) (sonido cuack))
      (animal (nombre vaca) (sonido mu))
      (animal (nombre oveja) (sonido bee))
      (animal (nombre gallo) (sonido kikiriki))
)

(defrule reconoceAnimal
      (sonidoAnimal ?x)
      (animal (nombre ?n) (sonido ?s)) ;(sonido ?x)
      (test (eq ?x ?s))
=>
      (printout t "Se trata de un " ?n crlf)
)
```

```
;=== Ejercicio 3 ===
;Programa que dado un conjunto de hechos de tipo 'datos' con dos
valores numéricos, detecte qué hechos cumplen que el segundo valor sea
mayor que el primero y los imprima.

(deffacts hechos
      (datos 2 8)
      (datos 6 3)
      (datos 5 5)
)

(defrule segMayor
      (datos ?p ?s)
      (test (> ?s ?p))
=>
      (printout t ?s " es mayor que " ?p crlf)
)



;=== Ejercicio 4 ===
;Programa que dado un conjunto de hechos tipo 'datos' con un número
indefinido de valores, detecte e imprima aquellos tal que el primer
valor sea par y menor o igual al último. (Hemos utilizado una
plantilla)

(deftemplate datos
      (slot nombre)
      (multislot valores)
)

(deffacts Datos
      (datos (nombre A) (valores 3 6 28 56 2)) ;(datos A 3 6 28 56 2)
      (datos (nombre B) (valores 12 5 12))
      (datos (nombre C) (valores 8 14 6 9 34 23 10))
)

(defrule imprimeDatos
      (datos (nombre ?n) (valores ?x $? ?y)) ;(datos ?n ?x $? ?y)
      (test (evenp ?x)) ;Que el número sea par (evenp)
      (test (<= ?x ?y))
=>
      (printout t "Los datos " ?n " cumplen la condición" crlf)
)
```

```
;=== Ejercicio 5 ===
;Programa que dado un hecho 'datos' con un número indefinido de
valores, elimine todas las apariciones del valor 1.

(deffacts datos
      (datos hola 1 3 nuevo 1 adios)
      (datos 1 2 1 4 1 6 1 8)
)

(defrule borra
      ?f<-(datos $?antes ?x&:(integerp ?x)&:(= ?x 1) $?despues)
=>
      (retract ?f)
      (assert (datos $?antes $?despues))
)



;=== Ejercicio 6 ===
;Programa que detecte e imprima los hechos que contengan los valores 3
y 4 en alguna posición, y que entre éstos haya un número impar de
valores. Utilizar la función 'length'.

(deffacts vectores
      (vector A 2 5 3 7 6 4 9)
      (vector B 1 3 8 19 3 4)
      (vector C 4 8 9 2 3 4)
)

(defrule vec
      (or   (vector ?nombre $?antes ?a&:(integerp ?a)&:(= ?a 3)
$?medio ?b&:(integerp ?b)&:(= ?b 4) $?despues)
            (vector ?nombre $?antes ?a&:(integerp ?a)&:(= ?a 4)
$?medio ?b&:(integerp ?b)&:(= ?b 3) $?despues)
      )
      (test (oddp (length $?medio))) ;Que la cantidad de números
(longitud) sea impar (oddp)
=>
      (printout t "El vector " ?nombre " cumple la condición" crlf)
)
```

```
;=== Ejercicio 9 ===
;Programa que dado un conjunto de hechos (dato 1) (dato verde)...,
cree un único hecho (todos-los-datos ...) con todos los valores de los
hechos anteriores.

(deffacts hechos
      (dato 1)
      (dato 5.0)
      (dato verde)
      (dato "hola")
)

(defrule assertTodos
      (not (todos-los-datos))
=>
      (assert (todos-los-datos))
)

(defrule regla
      ?f1<-(dato ?x)
      ?f2<-(todos-los-datos $?a)
=>
      (retract ?f1)
      (retract ?f2)
      (assert (todos-los-datos $?a ?x))
)


;=== Ejercicio 10 ===
;Programa que dado un conjunto de hechos (vector <nombreVector> <val1>
... <valn>) con valores numéricos, ordene sus valores de menor a
mayor.

(deffacts datos
      (vector A 3 35 7 23 4 1 13)
)

(defrule ordenMenorAMayor
      ?f<-(vector ?nombre $?antes ?x ?y $?despues)
      (test (< ?y ?x))
=>
      (retract ?f)
      (assert (vector ?nombre $?antes ?y ?x $?despues))
)
```