Inteligencia Artificial: CLIPS

José Marcos Moreno-Vega (jmmoreno@ull.es)
Israel López-Plata (ilopezpl@ull.es)
Christopher Expósito-Izquierdo (cexposit@ull.es)

Base de Conocimiento

- Contiene el conjunto de reglas que definen el comportamiento del Sistema Experto.
- También puede contener algunas afirmaciones iniciales.
- La Base de Conocimiento es específica del dominio en que el sistema puede considerarse experto.
- El contenido de la Base de Conocimientos se especifica en el fichero **.clp**, y no se modifica en ningún momento durante la ejecución del Sistema Experto.

BC. Reglas

- Una regla en un Sistema Experto representa el comportamiento del mismo cuando se producen determinadas condiciones.

- Consta de 2 partes:
  - **Antecedente.** Contiene las cláusulas que deben cumplirse para que la regla pueda evaluarse o ejecutarse (dispararse).
  - **Consecuente.** Indica las conclusiones que se deducen de las premisas (interpretación declarativa) o las acciones que el sistema debe realizar cuando ejecuta la regla (interpretación imperativa).

- **Ejemplos:**

Si bombilla_encendida entonces habitación_iluminada
Si temperatura_interior > temperatura_deseada entonces encender el ventilador

ANTECEDENTE

CONSECUENTE

BC. Elementos de una regla

- Los elementos que pueden intervenir en una regla son los siguientes.
  - **Hecho.** Por ejemplo, hielo_en_la_carretera, averia_electrica, etc. En un momento dado, cada hecho tiene asociado un valor de verdad.
  - **Dato.** Por ejemplo, nivel_de_gasolina, temperatura_interior, etc. Cada dato puede tomar cierto tipo de valores.
  - **Relación de comparación.** Se establece entre dos datos o entre un dato y un valor. Por ejemplo: nivel_de_gasolina = 8; temperatura_interior < 0
  - **Relación de pertenencia.** Se establece entre una instancia y un hecho no ordenado. Por ejemplo, rueda_izquierda es rueda; Pedro_García es paciente
  - **Cláusula.** consiste en una hipótesis o una relación (de comparación o pertenencia), o bien en la negación, conjunción o disyunción de otras cláusulas. Si es necesario pueden usarse paréntesis para indicar el alcance de cada operador.

BC. Regla en CLIPS

- Para definir una regla se utiliza la sentencia **defrule**.

```
(defrule <nombre-regla>
[<comentario>]
    [<declaración>]
    <antecedente>
    =>
    <consecuente>)
```

```
(defrule regla-ejemplo "Ejemplo de regla"
   (frigorífico interruptor encendido)
   (frigorífico puerta abierta)
   =>
   (assert (frigorífico comida estropeada)))
```

- La regla se identifica por su nombre, así que si se introduce en la base de reglas una nueva regla con el mismo nombre que el de una regla existente, la nueva regla reemplazará a la antigua.

- Si una regla no tiene antecedente, entonces el hecho **(initial-fact)** actuará como el elemento condicional para ese tipo de reglas, y la regla se activará cada vez que se ejecute un comando **reset**.

BC. Regla en CLIPS

- Para que una regla en CLIPS se ejecute, se deben producir 2 factores.
  1. Se satisfacen **todos** los elementos del antecedente.
  2. El motor de inferencia **selecciona** la regla.

- Debido a la condición número 1, se presupone que todos los elementos de un antecedente están separados por un **and**.

- A pesar de cumplir todas las condiciones, una regla puede no ser seleccionada. El criterio de selección se explica en mayor detalle en el tema de la Agenda de CLIPS.

BC. Regla en CLIPS. Antecedente

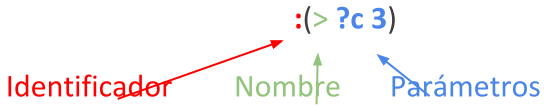
- Una condición básica en el antecedente de una regla se suele mostrar entre paréntesis.

- Se puede hacer uso de variables, las cuales cumplen las siguientes condiciones.
  - Se definen mediante **?<simbolo>.**
  - No se declaran.
  - Su ámbito es la regla donde se utilizan.
  - Se instancia la primera vez que aparece en una regla y mantiene su valor instanciado en sucesivas apariciones (dentro de la misma regla).
  - Se utilizan para relacionar diferentes entidades de las listas de hechos o instancias.
  - **Ejemplo:** (altitud es ?x metros)

- Se puede hacer uso de operadores lógicos. Estos son:
  - Negación: ~
  - Disyunción: |
  - Conjunción: &
  - **Ejemplo:** (habitación (plazas-libres ?p & 1|2))

BC. Regla en CLIPS. Funciones

- En un antecedente de una regla se pueden hacer llamadas a funciones.
- Una función en CLIPS consta de los siguientes elementos.



- Las funciones se pueden utilizar en cualquier parte del programa CLIPS.
- Pueden ser incluidas en el propio CLIPS o definidas por el usuario.
- **Ejemplo:** (habitación (capacidad ?c) (plazas-libres ?p & : (> ?c ?y)))
  (datos $?x & : (> (length ?x) 2))

BC. Regla en CLIPS. Funciones predefinidas

- CLIPS proporciona un gran número de funciones predefinidas que pueden ser utilizadas. Dependiendo del tipo, algunos ejemplos son las siguientes:
  - **Manejo de IU.** read, readline, format, printout.
  - **Entrada/Salida.** open, close, printout, read, readline.
  - **Funciones matemáticas.** abs, max, min, +, -, *, /, exp, log, mod, pi, round, random.
  - **Funciones con String.** lowcase, upcase, str-cat, str-compare, str-length.
  - **Funciones de entorno.** load, save, exit, system, batch.

BC. Regla en CLIPS. Elementos condicionales. test

```
test <llamada_funcion>
```

- Los elementos condicionales son aquellos que nos permiten cambiar las condiciones en el antecedente de una regla. Devuelven siempre **true** o **false**.
- El condicionante test devuelve **true** si la función llamada justo después devuelve cualquier valor distinto a **false**.
- Hacer uso de test nos permite omitir los : en la llamada a la función.
- **Ejemplo:** (test (>= (abs (- ?y ?x)) 3))
  - ¿Se cumplirá la regla si ?y=5 y ?x=2?
  - ¿Se cumplirá la regla si ?y=1 y ?x=2?
  - ¿Se cumplirá la regla si ?y=1 y ?x=6?

BC. Regla en CLIPS. Elementos condicionales. or

```
or <elemento_condicional>+
```

- Elemento condicional que devuelve **true** cuando se satisface al menos una de las condiciones de las que engloba.

- **Ejemplo:** (defrule fallo-del-sistema
  (error-status desconocido)
  (or (temperatura alta)(válvula rota)(bomba (estado apagada)))
  =>
  (printout t "El sistema ha fallado." crlf))

BC. Regla en CLIPS. Elementos condicionales. or

```
and <elemento_condicional>+
```

- Elemento condicional que devuelve **true** cuando se satisfacen todas las condiciones de las que engloba.
- Si en una regla el **and** ya viene implícito. ¿Por qué es necesario definirlo?
- ¿Cual es la diferencia entre las 2 reglas siguientes?

```
(defrule fallo-1
   (error-status confirmado)
   (or ((temperatura alta)(válvula cerrada))
       ((temperatura baja)(válvula abierta)))
   =>
   (printout t "El sistema tiene el fallo 1." crlf))
```

El sistema falla si la temperatura es alta, o la válvula está cerrada, o la temperatura es baja o la válvula está abierta.

```
(defrule fallo-1
   (error-status confirmado)
   (or (and (temperatura alta)(válvula cerrada))
       (and (temperatura baja)(válvula abierta)))
   =>
   (printout t "El sistema tiene el fallo 1." crlf))
```

El sistema falla si la temperatura es alta y la válvula está cerrada o si la temperatura es baja y la válvula está abierta.

- ¿Se puede representar el caso 2 sin un and explícito?

BC. Regla en CLIPS. Elementos condicionales. not

```
not <elemento_condicional>
```

- Elemento condicional que devuelve **true** el elemento condicional que engloba devuelve false.

- **Ejemplo:** (not (habitación (plazas-libres ?p & ~0) (capacidad ?c & : (> ?c ?p))))

BC. Regla en CLIPS. Elementos condicionales. exists

```
exists <elemento_condicional>+
```

- Elemento condicional que devuelve **true** si existe al menos un hecho de entre todos los disponibles que satisface el conjunto de condiciones del exist.
- Con este condicional se dice que existe algún hecho que cumpla las condiciones, que no tiene porque ser el mismo que cumple el resto de condiciones de la regla.
- Equivalente al ∃ de lógica proposicional.

BC. Regla en CLIPS. Elementos condicionales. forall

```
forall <primer_condicional> <resto_condicionales>+
```

- Elemento condicional que devuelve **true** si **todos** los hechos que cumplen con el primer condicional cumplen con los demás.
- Con este condicional se fuerza a que todos los hechos cumplan las condiciones que engloba.
- Equivalente al ∀ de lógica proposicional.

BC. Regla en CLIPS. Elementos condicionales. logical

logical <elemento_condicional>+

- Proporciona un mantenimiento de verdad a todos aquellos hechos incluidos por un consecuente que tenga elementos en el antecedente con el condicional logical.
- El mantenimiento de verdad dice que si un hecho B es generado porque un hecho A es verdadero, en el momento en el que A pase a ser falso, B también.
- Si en el antecedente aparece uno o varios logical, éstos deben aparecer los primeros.

```
(defrule correcta
   (logical (a))
   (logical (b))
   (c)
    =>
   (assert (d)))
```

```
(defrule incorrecta
   (a)
   (logical (b))
   (logical (c))
    =>
   (assert (d)))
```

BC. Regla en CLIPS. Elementos condicionales. logical

- Supongamos las siguientes reglas.

```
(defrule R1
   (logical(G))
    =>
   (assert(F)))
```

```
(defrule R2
   (logical(F))
    =>
   (assert (H)))
```

```
(defrule R2
   (logical(H))
    =>
   (assert (I)))
```

- Supongamos que decimos que F es cierto. Vemos como quedan las MT con estas reglas y con otras similares pero sin logical.

```
CON LOGICAL
INICIAL
F
H
I
```

```
SIN LOGICAL
INICIAL
F
H
I
```

- Supongamos que decimos que H es falso. Las MT quedarían así.

```
CON LOGICAL
INICIAL
F
```

```
SIN LOGICAL
INICIAL
F
I
```

BC. Regla en CLIPS. Obtener hechos utilizando reglas

- CLIPS permite en sus reglas ligar a una variable la dirección de un hecho que cumpla con las condiciones establecidas.
- El fin es poder tener acceso a los hechos de la MT y poder leer o modificar sus datos.
- Esto se hace con el operador <-.
- **Ejemplo.** Obtener una habitación que tenga todas sus plazas libres, mostrar su número y eliminarla de la memoria de trabajo.

```
(defrule hab-vacía
   ?vacía <- (habitación (capacidad ?c)
                         (plazas-libres ?c)
                         (número ?n))
=>
(printout t "Número habitación vacía: " ?n crlf)
(retract ?vacía))
```

Guarda en la variable ?vacía las habitaciones cuya capacidad ?c es la misma que las plazas libres.

Elimina la habitación de la MT.

BC. Consecuente. Modificar la MT

- Existen en CLIPS un conjunto de comandos para modificar la MT de trabajo. Estos son:
  - **(assert <hecho>+).** Añade el hecho o conjunto de hechos a la MT.
  - **(duplicate <dir_hecho>).** Duplica el hecho indexado por la variable <dir_hecho> de la MT.
  - **(retract <dir_hecho>).** Elimina el hecho indexado por la variable <dir_hecho> de la MT.
  - **(modify <dir_hecho> <slot>).** Modifica un slot del hecho indexado por la variable <dir_hecho> de la MT.

- El <dir_hecho> puede ser:
  - Una variable con la dirección del hecho.
  - Un índice del hecho.

BC. Consecuente. Modificar la MT. Ejercicio.

- Partiendo de los siguientes hechos ordenados

```
(deftemplate estudiante
   (slot nombre)
   (slot sexo)
   (slot fuma?)
   (slot alojado))
```

```
(deftemplate habitación
   (slot número)
   (slot capacidad)
   (slot sexos)
   (slot fuman?)
   (slot plazas-libres)
   (multislot
ocupantes))
```

- Escribir una regla que aloje a un estudiante en la mayor habitación libre, si no existe ninguna que esté parcialmente ocupada y que sea compatible (estudiantes del mismo sexo). Esto es:

SI existe un estudiante sin piso asignado
and
no existe una habitación compatible parcialmente ocupada
and
existe una habitación libre
and
es la habitación libre más grande
THEN
Alojar al estudiante
Registrar al estudiante en la habitación

BC. Consecuente. Modificar la MT. Ejercicio.

```
(defrule alojar-mayor-hab-libre
       ?vagabundo <- (estudiante (nombre ?nom)
                                 (sexo ?s)
                                 (fuma? ?f)
                                 (alojado nil))
       (not (habitación (plazas-libres ?p1 & : (> ?p1 0))
                        (capacidad ?c & : (> ?c ?p1))
                        (sexos ?s)
                        (fuman? ?f)))
       ?alojarlo-en <-  (habitación (número ?num)
                        (plazas-libres ?p2)
                        (capacidad ?c1 & ?p2))
       (not (habitación (capacidad ?c2 & : (> ?c2 ?c1))
                        (sexos nil)
                        (fuman? nil)))
     =>
        (modify ?vagabundo (alojado ?num))
        (modify ?alojarlo-en (plazas-libres (- ?p2 1))
                             (ocupantes (create$ ?nom))
                             (sexos ?s)
                             (fuman? ?f)))
```

BC. Acciones procedimentales

- Acciones que permite CLIPS similares a las permitidas por un lenguaje procedimental. Estas son:
  - **(bind <variable> <expresion>).** Asigna el valor de la expresión a la variable indicada.
  - **(if <expresion> then <accion>+ [else <accion>+]).**
  - **(while <expresion> do).**
  - **(loop-for-count (<var> <inicio> <final>) [do] <acción>+).** Bucle for en lenguajes procedimentales.
  - **(return [<expresión>]).** Termina la función actual, devolviendo un valor si se informa.
  - **(break).** Termina el ciclo actual.
  - **(switch <expresion>**
    <sentencia_case>
    <acción_case>+
    [(default <acción_default>)])