Introducción a CLIPS

CLIPS es un lenguaje que permite construir sistemas basados en reglas con encadenamiento hacia adelante usando el algoritmo RETE para la activación de reglas. Cualquier componente en CLIPS se expresa con una lista de símbolos y distingue entre mayúsculas y minúsculas.

Los comentarios de una línea en CLIPS comienzan con ; y los de varias líneas: /* comentario */.

1.1. Hechos

Los hechos en CLIPS pueden tener uno o varios campos (lo más habitual es que tengan varios) y se pueden expresar de distintas formas:

- <valor>
- <atributo><valor> → `(especie perro)`
- <objeto><atributo><valor> → `(Lassie especie perro)`
- <relación><atributo><valor> → `(tratado-con Juan antibiótico)`

Los hechos vistos hasta ahora se denominan **hechos sin etiqueta**, porque los campos que integran el hecho no llevan ninguna etiqueta identificativa. Este tipo de hechos son sensibles al orden. Es decir, estos dos hechos `(tratado-por Juan Dr.Lopez)` `(tratado-por Dr.Lopez Juan)` son dos hechos diferentes.

El tipo al que pertenecen los campos de un hecho sin etiqueta se asignan de forma automática cuando éste se almacena.

En CLIPS además existen los **hechos con etiqueta** (también llamados **plantillas**). Este tipo de hechos son insensibles al orden, ya que los datos se introducen indicando de forma explícita el campo al que pertenecen. La forma general de definir un hecho con nombre o plantilla es mediante la instrucción (*deftemplate*), cuya sintaxis general es la siguiente:

(deftemplate <nombre de plantilla> "Comentario opcional entre comillas"
<definición campo-1>
<definición campo-2>
...
<definición campo-n>
)
<definición campo> = (Slot <nombre > [(type <tipo >) [default <valor >)] … |
Multislot <nombre > [(type <tipo >) [default <valor >)]...)
Los valores posibles para type son: SYMBOL, FLOAT, INTEGER, STRING y NUMBER.

Por ejemplo, para definir una plantilla para almacenar estudiantes:

```
(deftemplate estudiante
  (slot nombre
      (type string))
```

```
(multislot apellidos
     (type SYMBOL))
(slot edad
     (type INTEGER)
     (default 30))
(slot sexo
      (type SYMBOL)
      (allowed-symbols V v M m))
)
```

- Las primitivas `(allowed-symbols)`, `(allowed-strings)`, `(allowed-numbers)`, `(allowed-integers)` `(allowed-floats)` `(allowed-values)` se utilizan para enumerar los valores permitidos para un campo concreto.

Para incluir un hecho en la Base de Hechos hay que asertarlo mediante la instrucción *assert*. Por ejemplo: `(assert (tipo-alienígena marciano)).`

Para eliminar un hecho se usa la instrucción (*retract n*) donde n es el número del hecho a eliminar. En la instrucción *retract* puede utilizarse el símbolo * para eliminar todos los hechos introducidos hasta el momento.

Todos los hechos que se han introducido tienen un número de identificador, relativo al orden de creación de los hechos, que identifica de forma univoca cada hecho. f-0 se corresponde con el hecho inicial creado automáticamente por CLIPS. Si se elimina un hecho su índice no se reasigna a ningún otro hecho. Ejemplo de identificadores de hechos:

```
f-0 (initial-fact)
f-1 (Lassie especie perro)
f-2 (Lassie domestico si)
f-3 (tratado-con Lassie antibiotico)
f-4 (tratado-por Lassie DeLaTorre)
```

Con la instrucción (*deffacts*) lo hechos no se cargan en memoria directamente, sino que se cargan cuando hayamos reiniciado el sistema. El formato de la instrucción es el siguiente:

```
(deffacts   (<hecho-1>)
(<hecho-2>)
        ...
(<hecho-n>)
)
```

1.2. Variables

CLIPS ofrece la posibilidad de utilizar variables para almacenar valores. Estas variables se diferencian de los hechos en que son dinámicas y que los valores que se les asignan pueden cambiar.

El identificador de una variable siempre se escribe como un signo de interrogación seguido de un símbolo que es el nombre de la variable.

Las variables en CLIPS no son tipadas aunque los valores sí lo sean.

Para asignar un valor a una variables se usa la función *bind*, por ejemplo: `(bind ?edad 18)`.

Las variables multivaluadas empiezan por $?, por ejemplo: `$?apellido`.

Las variables globales se definen con la función *defglobal*, por ejemplo: `(defglobal ?*variable* = valor-por-defecto)`.

CLIPS también permite crear listas. Una lista es una secuencia ordenada de valores y se crean con la instrucción create$. Por ejemplo:

```
(bind ?ejemplolista (create$ a b c d e))
```

Funciones para el manejo de listas:

- (nth$) → Devuelve el enésimo elemento.
- (first$) → Devuelve el primer elemento de la lista.
- (rest$) → Devuelve la lista sin su primer elemento.

1.3. Reglas

La instrucción (*defrule*) se utiliza para definir reglas en CLIPS. Su sintaxis es la siguiente

```
(defrule nombreRegla ["Descripción"]
(<patrón-1>)      ; Parte izquierda de la regla
(<patrón-2>)
...
(<patrón-n>)
=>
(<acción-1>)      ;Parte derecha de la regla
(<acción-2>)
...
(<acción-m>)
)
```

Ejemplos:

```
(defrule semaforo-rojo
   (luz roja)
   =>
   (printout t "Detengase" crlf))

(defrule trata-infeccion "Tratamiento empírico"
   (Perez riesgo-infeccion si)
```

```
   (Perez infeccion-antes si)
    =>
   (assert (Perez dar penicilina)
)
```

Las condiciones de una regla están implícitamente conectadas con *and*. Si necesitamos un *or* entonces hay que dividir la regla en dos.

Existen algunas situaciones en las que sólo se especifica una parte de un hecho, y el resto de éste puede contener cualquier valor o estar vacío. Para definir este tipo de reglas se emplean los comodines. Existen dos tipos de comodines en CLIPS:

- Monocampo: El símbolo ? sustituye exactamente a un campo. Por ejemplo:

```
CLIPS> (defrule busca-apellido
(nombre Juan ?)
=>
(assert (encontrado-Juan-apellido si)))

CLIPS> (assert (Juan Gil Martín))
CLIPS> (assert (Juan))
CLIPS> (assert (Juan Gil))
CLIPS>
```

  De los tres hechos, sólo el segundo activa la regla, ya que es el único que posee el símbolo Pedro seguido exactamente de un símbolo.

- Multicampo: El símbolo $? Sustituye cero, uno o más campos en un hecho. Por ejemplo:

```
CLIPS> (defrule busca-apellido
(nombre Juan $?)
=>
(assert (encontrado-Juan-apellido si)))

CLIPS> (assert (Juan Gil Martín))
CLIPS> (assert (Juan))
CLIPS> (assert (Juan Gil))
CLIPS>
```

  En este caso los 3 hechos activan la regla, ya que todos contienen Juan seguido de 0, uno y dos campos.

Se pueden usar variables en las reglas para almacenar un valor en el antecedente y luego utilizarlo en el consecuente. Por ejemplo:

```
(defrule quienesquien
  (cazador ?cazador ?cazado)
   =>
  (printout t ?cazador "dispara al" ?cazado crlf))
```

La parte izquierda también puede incluir un test de expresiones lógicas: (test <expresión lógica>). Por ejemplo: `(test (< ?edad 18)).` También puede incluir expresiones con variables y conectivas lógicas:

- not: ~
- and: &
- or: |

En CLIPS pueden realizarse operaciones aritméticas sobre datos de tipo numérico. El lenguaje proporciona las operaciones y funciones aritméticas básicas: `+, -, *, /, div, max, min, abs, float e integer`. Su significado es análogo al que tienen en cualquier otro lenguaje de programación. Las expresiones numéricas se representan en CLIPS en notación prefija. Por ejemplo, para representar la suma de los enteros 2 + 3, deberíamos escribir `(+ 2 3)`.

Además de las restricciones conectivas, existen en CLIPS las restricciones de predicado, que se usan para evaluar condiciones con campos más complejos. El propósito de este tipo de restricción es admitir o rechazar un campo dependiendo del valor de una expresión Booleana. Si el resultado de la expresión es FALSE, la restricción no se satisfará y la condición no se cumplirá. Este tipo de restricción es muy útil con patrones de tipo numérico. Una función predicado es aquella que devuelve un valor FALSE o no-FALSE. Los dos puntos ":" seguidos de una función predicado constituyen una restricción predicado. Los dos puntos pueden ir precedidos de "&", "|" o de "~" o pueden ir solos como en el patrón `(hecho :(> 2 1))`.

En la parte derecha de las reglas aparecerán distintas acciones implícitamente conectadas con and. Las acciones posibles son:

- Crear un hecho (*assert*).
- Eliminar un hecho (*retract*).
- Modificar el campo de un hecho con nombre (*modify*).
- Asignar un valor a una variable (*bind*).
- Para la ejecución (*halt*)
- Acciones de entrada / salida (*printout*, *read*, *readline*).
  - La función *(read)* sólo lee un campo. Si queremos introducir varias palabras separadas por espacio, debemos encerrarlas entre comillas y CLIPS interpretará la cadena como un solo hecho. Una vez hecho esto, existen funciones que permiten convertir estas cadenas en los correspondientes hechos multicampo.
  - La función (*readline*) se usa para leer valores múltiples hasta que finalicemos introduciendo un retorno de carro. Esta función lee los datos como una cadena. Para insertar hechos introducidos mediante una función (readline), se utiliza la

función (assert-string), que convierte una cadena de caracteres en un hecho de tipo no cadena.

Ejemplo de regla que modifica un hecho:

```
defrule poner-apellidos
   ?persona <-(persona (nombre ?nombre) (apellidos))
      ;; solo equiparan personas sin apellidos
   =>
   (printout t crlf "Introduce apellidos para " ?nombre ": " )
   (modify ?persona (apellidos (read))))
```

Ejemplo de regla que permite al usuario introducir un hecho:

```
(defrule inserta-hecho
      ; no tiene LHS: se dispara si (reset) + (run)
      =>
      ; para escribir un texto al disparar regla
      (printout t "Escribe un hecho como cadena" crlf)
      (assert-string (read)))  ; para leer un hecho
      (reset)
      (run)
Al activarse esta regla ocurriría lo siguiente:
   1. El ordenador escribe esto por pantalla:
            Escribe un hecho como cadena
   2. El usuario escribe un hecho (entre comillas)
         "(persona (nombre NEO) ) "
   3. El sistema añade el siguiente hecho:
         (persona (nombre NEO) (apellidos ) (edad ) (estado ))
```

Si hubiera alguna regla sin parte izquierda (LHS) solo se ejecutará al ejecutar (reset).

1.4. Orden de ejecución de las reglas en CLIPS

CLIPS utiliza una estructura para almacenar y operar con las reglas que se han activado en un instante dado. El mecanismo de resolución de conflictos de reglas de CLIPS por defecto es LIFO: se comprueba qué reglas están activadas y se ejecuta la última que se ha activado.

Si queremos cambiar la estrategia LIFO por una estrategia FIFO podemos hacerlo con el comando (*set-strategy <estrategia>*), los valores posibles para estrategia serán:

- depth: estrategia tipo LIFO.
- breadth: estrategia tipo FIFO.

Además, existe un mecanismo de prioridades manipulable por el usuario. Así, si dos reglas se activan simultáneamente, el orden en el que se ejecutarán dependerá de su prioridad (la más prioritaria irá primero). La propiedad (*salience*) define la prioridad de ejecución de una regla. Para establecer la prioridad de una regla habrá que definir la propiedad *salience* dentro de la regla: *(declare salience <num>).*

1.5. Instrucciones de control

CLIPS proporciona las instrucciones de control típicas: if, if-else, if-else-if, for, while… La sintáxis básica es la siguiente:

```
(if <exp> then <accion>*
 [elif <exp> then <accion>*]*
 [else <accion>*])

(while <exp> [do] <accion>*)

(foreach <var> <lista> <accion>*)
```

1.6. Funciones

Para definir una función en CLIPS utilizamos la construcción (*deffunction*). Los elementos de una función son:

- Nombre.
- Comentario (opcional).
- Lista de cero o más argumentos obligatorios.
- Un argumento para recibir el resto de argumentos recibidos (opcional).
- La secuencia de acciones y expresiones que se ejecutarán.

La sintaxis es:

```
(deffunction <nombre> [<comentario>]
 (<regular-parameter>* [<wildcard-parameter>])
 <acción>*)
<regular-parameter> ::= <single-field-variable>
<wildcard-parameter> ::= <multifield-variable>
```

Ejemplo de función en CLIPS:

```
(deffunction calculaCalorias (?peso ?altura ?edad ?sexo
?actividadFisica)
(if (= ?sexo h) then
(bind ?resultado (* (- (+ 66 (* 13.7 ?peso) (* 5 ?altura)) (*
6.8 ?edad)) ?actividadFisica))
else
```

```
(bind ?resultado (* (- (+ 665 (* 9.6 ?peso) (* 1.8 ?altura)) (*
4.7 ?edad)) ?actividadFisica)))
(return ?resultado))
```

La llamada a funciones sigue una sintaxis con notación prefija: (*id-fun arg1 arg2 …).*

En CLIPS existen funciones predefinidas para comprobar el tipo de una expresión o valor: *(numberp <exp>) (stringp <exp>) (integerp <exp>)*.

Mediante la construcción (*defgeneric*) podemos sobrecargar funciones y definir nuevos comportamientos en función del número y tipo de los argumentos.

1.7. Comandos de interacción con el sistema

- (facts) → Lista todos los hechos presentes en la memoria de trabajo.
- (rules) → Lista todas las reglas que haya en la base de reglas.
- (clear) → Elimina todos los hechos y reglas de la Memoria de Trabajo.
- (reset):
    - Elimina todos los hechos de la Memoria de Trabajo.
    - Elimina las activaciones de las reglas.
    - Inserta el hecho inicial.
    - Inserta los hechos definidos con deffacts.
    - Añade las variables globales con su valor inicial.
- (run) → Ejecución.
- (run 1) → Ejecución de un solo paso.
- (step) → Ejecución paso por paso.
- (help) → Ayuda del sistema.
- (ppdefrule nombreRegla) → Muestra el contenido de la regla <nombreRegla>.
- (watch) → Activa los mecanismos de depuración de CLIPS.
- (watch facts) → Traza lo que va ocurriendo con los hechos durante la ejecución.
- (watch activations) → Traza lo que va ocurriendo con las activaciones durante la ejecución.
- (watch rules) → Traza lo que va ocurriendo con las reglas durante la ejecución.
- (unwatch) → Desactiva los mecanismos de depuración de CLIPS.

Bibliografía complementaria

http://clipsrules.sourceforge.net/OnlineDocs.html

http://www.uco.es/users/sventura/misc/TutorialCLIPS

https://www.csee.umbc.edu/portal/clips/tutorial/