CLIPS

**Hecho (Fact)**

Información

* **(facts)**
* **(Assert ())**
* **(retracted #)**

**Regla (Rule)**

Mecanismo que utiliza CLIPS para inferir conocimiento, es decir, obtener nuevo conocimiento (nuevos hechos), a partir del conocimiento actual (hechos actuales).

Por ejemplo, si está lloviendo y no llevo paraguas me mojo. En este caso está lloviendo y no llevo paraguas son los hechos iniciales de los cuales el sistema deduce un nuevo hecho: me mojo.

**Añadir hechos**

(deffacts hechos-iniciales

(llueve)

(no-tiene carlos paraguas)

)

(reset)

Esta orden elimina todos los hechos del sistema (excepto el hecho initial-fact) y carga todos los hechos definidos con deffacts.

**Añadir reglas**

(defrule [Nombre de la regla]

([hecho precedente 1])

([hecho precedente 2])

=>

(assert ([nuevo hecho a insertar]))

)

(defrule si-llueve-te-mojas

(llueve)

(no-tiene carlos paraguas)

=>

(assert (se-moja carlos))

)

(clear) → reinicia el sistema

(run) → hace que el programa empiece a ejecutarse. Si hemos definido una serie de hechos y una serie de reglas, ninguna regla se activará hasta que no hayamos hecho un run.

(facts) → muestra los hechos del sistema junto a su número de hecho (initial-fact tiene asignado el número 0 y a los sucesivos hechos se les va dando un número a partir del 1).

(rules) → muestra las reglas del sistema.

(retract numero) → elimina el hecho cuyo número se especifique. Se puede especificar más de un numero de hecho a eliminar.

(printout t “hola mundo”) → printout es similar al fprintf de C. El primer argumento indica el dispositivo de salida (t indica la salida estándar, es decir, la pantalla) y a continuación pueden ir todos los elementos que se desee, los cuales se imprimirán en pantalla. Si queremos insertar un salto de linea debemos escribir crlf .

(read) (readline) → leen del teclado hasta que el usuario pulsa enter. La diferencia es que read sólo devuelve la primera palabra, es decir, hasta el primer blanco, mientras que readline devuelve hasta el enter.

**Patrón**

Si se quiere agregar la regla cuando juan no tiene paraguas y llueve, es necesario repetir el código anterior cambiando carlos por juan pero esto se puede evitar usando patrones.

(defrule si-llueve-te-mojas

    (llueve)

    (no-tiene ? paraguas)

    =>

    (assert (alguien-se-moja))

)

Como se debe especificar quien se moja en el hecho que se inserta al finalizar la regla, se nombra el comodín “?”.

(defrule si-llueve-te-mojas

    (llueve)

    (no-tiene ?x paraguas)

    =>

    (assert (se-moja ?x))

)

**Listas**

Hecho con varios elementos.

(deffacts hechos

    (animal leon)

    (animales leon cebra loro lobo tiburon serpiente)

)

Hemos declarado dos hechos, el primero (animal) nos dice que el león es un animal, y el segundo (animales) es una lista de animales en la que están el león, la cebra, el loro, etc. Esto es muy útil, pues nos evitamos escribir un hecho para cada uno de los animales.

(deffacts hechos

    (animales leon cebra loro lobo tiburon serpiente)

    (sangre-caliente leon cebra loro lobo)

    (no-pone-huevos leon cebra lobo)

)

(defrule es-mamifero

    (animales $? ?animal $?)

    (sangre-caliente $? ?animal $?)

    (no-pone-huevos $? ?animal $?)

=>

    (printout t "El " ?animal " es un mamífero" crlf)

)

El significado de **$?** es similar al de**?**, con la diferencia que, mientras que **?** se liga con un único elemento, **$?** puede ligarse con 0, 1 o más elementos. **$?** puede ir seguido de un nombre de variable (al igual que **?**) en el que guardar la parte de la regla con la que queda ligado, pero en este caso nos es indiferente.

Dado el conjunto de hechos iniciales, la regla es-mamífero se disparará una vez por cada uno de los mamíferos (para ser mamífero hace falta ser animal, tener sangre caliente y no poner huevos), es decir, su resultado debe ser algo parecido a lo siguiente:

El leon es un mamífero

El cebra es un mamífero

El lobo es un mamífero

Ampliar un poco más la regla de forma que mantengamos una lista con los mamíferos que se vayan encontrando.

(defrule es-mamifero

    (animales $? ?animal $?)

    (sangre-caliente $? ?animal $?)

    (no-pone-huevos $? ?animal $?)

    ?hecho-mamiferos <- (mamiferos $?lista)

=>

    (printout t "El " ?animal " es un mamífero" crlf)

    (retract ?hecho-mamiferos)

    (assert (mamiferos $?lista ?animal))

)

Sólo hemos añadido un hecho más al precedente de la regla (mamiferos $?lista), este hecho es el que almacena la lista de mamíferos. En el consecuente de la regla también se han añadido 2 instrucciones, la primera, **retract**, elimina la lista de mamíferos de la base de hechos, mientras que la segunda lo que hace es añadir la lista de mamíferos de nuevo pero con el nuevo mamífero encontrado.

En cuanto al significado del operador **<-**, lo que hace es asignar el hecho de la derecha a la variable de la izquierda. Bastante intuitivo.

Resumiendo, esto es**lo que hemos aprendido** hoy sobre CLIPS:

* Manejo de hechos como listas.
* Significado del comodín **$?.**
* Significado del operador **<-.**
* Eliminar hechos con **retract**.