PRÁCTICA NÚMERO 4
INTRODUCCIÓN A CLIPS

Control de la ejecución de programas CLIPS

Cuando la base de reglas es muy grande, se produce un problema para el mantenimiento, la depuración, la comprensión y la traza del programa. Para reducir estos problemas se usa el diseño modular.

Mediante el diseño modular se organiza mejor la base de reglas y la forma de ejecución del programa. La organización de los hechos no se hace a nivel del hecho individual, sino a nivel de los constructores sobre los que están basados. El acceso a las construcciones entre diferentes módulos depende de la declaración del módulo y sus construcciones.

Para definir un módulo se utiliza la sentencia **defmodule**:

(defmodule <nombre-modulo> [<comentario>] <especificacion-de-acceso>*)

Donde <especificacion-de-acceso> serán líneas de la forma:
(**export** <elemento>)
ó
(**import** <nombre-modulo> <elemento>)

A su vez, “elemento” puede se una de las siguientes expresiones:
?ALL
?NONE
<construccion> ?ALL
<construccion> ?NONE
<construccion> <nombre-construccion>+

Las construcciones que se pueden exportar son: deftemplate, defclass, defglobal, deffunction o defgeneric, no se exportan ni los defrule ni los deffacts. Un módulo puede

exportar todas las construcciones que le son visibles, no sólo las que se definen dentro del propio módulo.

La única forma de borrar un módulo es mediante la sentencia clear. Después de un comando clear, se crea automáticamente el módulo MAIN. Este módulo es el único que puede ser redefinido una sóla vez después del inicio de CLIPS.

Para crear una construcción del tipo defrule o deffacts dentro de un determinado módulo se pueden usar dos formas: explícita o implícita.

1. La forma explícita se da cuando se hace referencia directa al nombre del módulo donde estará la construcción que se define, para realizar una referencia explícita se escribe el nombre del módulo seguido de "::" y por último el nombre de la construcción:

   DETECCION::saltar-alarma

2. La forma implícita se da cuando el módulo donde se define la construcción está implícito en el sistema, ya que siempre existe un módulo "actual". El módulo actual cambia cuando:
   - Se define una nueva construcción defmodule.
   - Se especifica el nombre de un módulo de forma explícita (con "::").
   - Se especifica el módulo actual con la instrucción **set-current-module**.

Exportación e importación de construcciones

La especificación export se usa para indicar qué construcciones son visibles a otro módulo externo a aquél en que se usa.

La especificación import sirve para indicar que la construcción utilizada en el módulo que se está definiendo pertenece a otro módulo.

Formas de hacer la importación o exportación:

1. Para exportar todas las construcciones que son visibles a ese módulo se usa la palabra clave ?ALL:

   (defmodule A(export ?ALL))

2. Para exportar todas las construcciones de un determinado tipo que le son visibles, se usa la instrucción:

   (export <tipo> ?ALL)

   Por ejemplo:

   (defmodule B (export deftemplate ?ALL))

3. Para exportar unas construcciones de un determinado tipo y con unos determinados nombres se utiliza la expresión:

   (export <tipo> <nombre_construccion>+)

Para importar, las formas de las instrucciones son las mismas, cambiando “export” por “import” y teniendo en cuenta que hay que indicar el nombre del módulo de donde se está importando.

Para indicar que no se exporta/importa ninguna construcción del módulo (o ninguna construcción de determinado tipo) las instrucciones son como las de los casos 1 y 2 anteriores, cambiando ?ALL por ?NONE.

Para poder importar construcciones de otros módulos, tanto los módulos de los que se importa como las construcciones que se importan deberán estar previamente definidas.

Las construcciones deftemplate y todos los hechos que las utilicen pueden ser compartidas con otros módulos. Los hechos correspondientes a una declaración de plantilla son “propiedad” del módulo donde se definió el deftemplate y no del módulo donde se crea el hecho.

Los hechos e instancias son visibles sólo para aquellos módulos que importan la correspondiente deftemplate o defclass al que pertenecen estos objetos. El hecho initial-

fact se importa siempre de forma explícita en todos los módulos, ya que si no, no se podrían ejecutar reglas que tuviesen este hecho como condición.

Una base de conocimiento puede dividirse de forma que las reglas y otros constructores puedan ver solamente aquellos hechos o instancias que sean de su interés.

Control de la ejecución

Además del control sobre la visibilidad de los diferentes componenetes del sistema, los módulos se usan también para controlar la ejecución del programa (la ejecución de las reglas).

Cada módulo tiene su propia agenda, con sus propias reglas activas. El control del programa se hace "poniendo el foco" en uno de los módulos. Cuando se ejecuta el comando run, se ejecutarán las reglas activas de la agenda correspondiente al módulo que tenga el foco en ese momento.

Para cambiar el foco a un módulo, se utiliza el comando **focus**:
(focus <nombre-módulo>+)

CLIPS guarda la información sobre el módulo que tiene el foco en cada momento en una variable denominada "foco actual". Además, CLIPS guarda información sobre los módulos que han tenido el foco en etapas anteriores en una estructura en forma de pila, de forma que el valor del último foco se coloca sobre todos los anteriores.

La ejecución de reglas de una agenda continúa hasta que:

- Cambia el foco a otro módulo.
- No hay más reglas en esa agenda.
- Se ejecuta el comando **return** en la parte derecha de alguna regla del módulo que tiene el foco.

Cuando se vacía la agenda de un módulo (foco actual), el foco actual se elimina de la pila de focos, y el siguiente módulo en la pila de focos se convierte en el foco actual. Si en una

misma sentencia focus se especifica más de un foco, los focos entran a la pila de derecha a izquierda según se han especificado en la instrucción.

Un mismo módulo puede estar más de una vez en la pila de focos, pero ejecutar un comando focus sobre un módulo que ya es foco actual no tiene efecto alguno.

Comandos de la pila de focos:

(list-focus-stack)	Lista el contenido de la pila.
(clear-focus-stack)	Elimina todos los focos de la pila.
(get-focus-stack)	Devuelve un valor multicampo con los nombres de los módulos en la pila.
(get-focus)	Devuelve el nombre del módulo que tiene el foco actual, o el valor FALSE si la pila está vacía.
(pop-focus)	Elimina el foco actual de la pila de focos y devuelve el nombre del módulo, o FALSE si la pila está vacía.

El comando return

Para detener la ejecución de las reglas de una agenda antes de que ésta esté vacía se usa el comando **return**.

Si se usa en la parte derecha de una regla, no se deben indicar argumentos. Además de quitar el foco actual de la pila, el comando return hace que se interrumpa la ejecución de las acciones de la regla. Si sólo se quiere quitar el foco actual pero terminar de ejecutar las acciones de la regla se usaría el comando pop-focus.

Propiedad auto-focus

La propiedad **auto-focus** supone la ejecución de un comando focus cuando la regla en la que está declarada se activa, convirtiendo en foco actual al módulo en el que está esta regla. La declaración del auto-focus se hace en la sentencia declare de una regla:

```
(defrule regla-1
        (declare (auto-focus TRUE))
....
....
```

VARIABLES GLOBALES

La construcción *defglobal* permite definir variables que sean visibles globalmente en todo el entorno de CLIPS. Es decir, una variable global puede ser accedida desde cualquier punto en CLIPS y retiene su valor independientemente de otras construcciones.
Las variables globales en CLIPS no están restringidas a contener un valor de un tipo de datos determinado.

Ejemplo:
CLIPS>(defglobal ?*variableA* = 12)
CLIPS>(defglobal ?* variableB* = 1 ?* variableC* = 2 ?* variableD* = 3)

Sintaxis: (defglobal [<nombre-módulo>] <asignación>*)
<asignación> ::= ?*<nombre-variable-global>* = <expresión>
donde <nombre-módulo> indica el módulo en el que se define(n) la(s) variable(s) global(es). Si no se especifica, las definiciones globales se colocarán en el módulo actual. Si se usa una variable ya definida anteriormente con una construcción *defglobal*, su valor se reemplazará con el valor especificado en la nueva construcción *defglobal*.

Para que un módulo *A* pueda acceder a una variable global de otro módulo B, este último debe exportarla, y *A* debe importarla.

Al igual que con las variables locales mediante la función bind se puede asignar el valor de la variable global.

Las variables globales recuperan su valor inicial cuando:

- se utiliza la función *bind* con la variable global sin especificar un valor, ó
- se ejecuta un comando *reset*.

Los siguientes comandos permiten eliminar las variables globales del entorno:

- *clear*
- *undefglobal*

CLIPS>(undefglobal variableA)

Con el comando undefglobal la variable global se especifica sin el '?' y sin los asteriscos. Además la variable no debe estar actualmente en uso. En caso contrario, se produce un error. Se puede usar el símbolo '*' para eliminar todas las variables globales, a menos que exista una variable global cuyo nombre sea *, en tal caso se elimina ésta, y no todas. Esta función no devuelve ningún valor.

Se puede visualizar el valor de una variable global introduciendo su nombre en el *prompt* directamente y sin paréntesis, o mediante el comando show-defglobals.

Sintaxis: (show-defglobals [<nombre-módulo>])

CLIPS>(show-defglobals)

Este comando muestra el valor de todas las variables globales del módulo especificado, o del actual, si no se especifica. Si se da como nombre de módulo el símbolo '*', presenta todas las variables (y sus valores) de todos los módulos.

EJERCICIOS

Adivinar un número

Dado un número entero entre 1 y 100, se pide realizar un programa en CLIPS que permita al usuario introducir números enteros entre 1 y 100 hasta que adivine el número que el sistema tiene almacenado. La entrada de números se hará mediante la sentencia read, después de cada entrada, el sistema debe indicar por pantalla si el número buscado es mayor o menor que el introducido. La condición de parada se produce cuando el usuario acierte el número en cuestión.
El número a adivinar debe almacenarse en una variable global.

Control de temperatura

Realizar un programa en CLIPS que indique si la temperatura de unas válvulas es normal o peligrosa. La temperatura será peligrosa cuando pase de 95 ºC y no es peligrosa cuando sea igual o menor a este valor. Los datos de la temperatura se recogen de un fichero "temperaturas.dat" que contiene estos valores

valvula1 110
valvula2 56
valvula3 98
valvula4 90
valvula5 43
valvula6 95
valvula7 114

La salida, indicando la situación de la temperatura deberá guardarse en un archivo de nombre "alarmas.out".

Se utilizarán módulos para controlar la ejecución del programa.