Tema 8: Módulos

El constructor *defmodule*

Introducción

- CLIPS permite estructurar la base de conocimientos y al mismo tiempo facilitar el control del razonamiento de forma modular.
- *defmodule* permite agrupar constructos en módulos.

Sintaxis

```
(defmodule <nombre-módulo> [<comentario>]
   <especificación-portabilidad>*)

<especificación-portabilidad> ::=
            (export <port-item>) |
            (import <nombre-módulo> <port-item>)

<port-item>  ::= ?ALL |
                 ?NONE |
        <constructor-portable> ?ALL |
        <constructor-portable> ?NONE |
        <constructor-portable> <nombre-constructo>+

<constructor-portable>   ::= deftemplate | defglobal
      | deffunction | defgeneric
```

Definición de módulos

- Un módulo no puede redefinirse ni borrarse, salvo el módulo MAIN, que puede redefinirse una vez.
- La única manera de borrar los módulos es con *clear*.
- Siempre existe un módulo MAIN.

Especificación de módulo

- Cuando se define un constructo, éste se asocia siempre a un módulo.
- Para indicar a qué módulo se asocia un constructo hay dos métodos:
  - De manera explícita.
  - De manera implícita.

Especificación de módulo

- De manera explícita:
  - Para *deffacts*, *deftemplate*, *defrule*, *deffunction* y *defgeneric* se indica incluyéndolo como parte del nombre usando el separador de módulo '::'.
  - Para *defglobal* se indica después de la palabra reservada *defglobal*.
  - El módulo de un *defmethod* es el del *defgeneric* al que se asocia.
- De manera implícita:
  - El constructo se asocia al módulo actual.

Especificación de módulo

- Para hacer referencia a un constructo, es necesario especificar el módulo en el que se encuentra, lo cual puede hacerse de dos maneras:
  - Explícitamente: mediante el nombre del módulo seguido de ‘::’ (para cualquier tipo de constructo).
  - Implícitamente: siempre hay un módulo actual.

Módulo actual

- El módulo MAIN es definido automáticamente por CLIPS y es el módulo actual cada vez que se inicia el intérprete o se ejecuta el comando *clear*.
- El módulo actual se cambia
  - Con la función *set-current-module*.
  - Cuando se define un módulo.
  - Cuando se define un constructor asociado a un módulo.

Ejemplo

```
CLIPS> (clear) ;Módulo actual MAIN
CLIPS> (defmodule A) ;Módulo actual A
CLIPS> (defmodule B) ;Módulo actual B
CLIPS> (defrule r1 =>)
CLIPS> (defrule A::r2 =>) ;Módulo actual A
CLIPS> (list-defrules)
r2
For a total of 1 defrule.
CLIPS> (set-current-module B) ;Módulo actual B
A
CLIPS> (list-defrules)
r1
For a total of 1 defrule.
```

Importación y exportación

- A no ser que se exporten y se importen explícitamente, los constructos de un módulo no pueden ser usados en otros módulos.
- Se dice que un constructo es visible en un módulo si puede usarse en ese módulo.
- Sólo pueden importarse y exportarse plantillas, variables globales y funciones, tanto convencionales como genéricas.
- Los métodos se importan/exportan implícitamente al importar/exportar las correspondientes funciones genéricas.

Importación y exportación

- Un módulo puede exportar cualquier constructo que sea visible en él (no sólo los definidos en él).
- Para importar constructos de un módulo, éstos deben haber sido previamente definidos.
- Los hechos pertenecen al módulo en el que sea visible su correspondiente plantilla, no al módulo en que han sido creados.
- La plantilla *initial-fact* debe ser importada de manera explícita.

Plantillas implícitas e inicialización de CLIPS

- Cuando se afirma o se hace referencia a un hecho ordenado se crea una plantilla implícita con una única casilla multicampo.
- El nombre de la casilla multicampo no se imprime cuando se imprime el hecho.
- Cuando se inicializa CLIPS, se definen automáticamente los siguientes constructos:

```
(deftemplate initial-fact)
(deffacts initial-fact
   (initial-fact))
```

Ejemplo

```
CLIPS> (clear)
CLIPS> (defmodule A)
CLIPS> (deftemplate A::datos (slot x))
CLIPS> (defmodule B)
CLIPS> (defrule B::r1 (datos (x 3)) =>)

[PRNTUTIL2] Syntax Error:  Check appropriate syntax for
   defrule.

ERROR:
(defrule B::r1
   (datos (
CLIPS> (clear)
CLIPS> (defmodule A (export deftemplate datos))
CLIPS> (deftemplate A::datos (slot x))
CLIPS> (defmodule B (import A deftemplate datos))
CLIPS> (defrule B::r1 (datos (x 3)) =>)
```

Ejemplo

```
CLIPS> (clear)
CLIPS> (defmodule A (export deftemplate datos data))
CLIPS> (deftemplate datos (slot x))
CLIPS> (deftemplate data (slot y))
CLIPS> (deffacts info (datos (x 1)) (data (y 2)))
CLIPS> (defmodule B (import A deftemplate datos))
CLIPS> (reset)
CLIPS> (facts A)
f-1     (datos (x 1))
f-2     (data (y 2))
For a total of 2 facts.
CLIPS> (facts B)
f-1     (datos (x 1))
For a total of 1 fact.
```

Ejemplo

```
CLIPS> (clear) ;Cuidado con las plantillas implícitas
CLIPS> (defmodule A (export deftemplate ?ALL))
CLIPS> (assert (datos 1))
<Fact-0>
CLIPS> (facts)
f-0     (datos 1)
For a total of 1 fact.
CLIPS> (defmodule B (import A deftemplate ?ALL))
CLIPS> (assert (datos 2) (data 3))
<Fact-2>
CLIPS> (facts)
f-0     (datos 1)
f-1     (datos 2)
f-2     (data 3)
For a total of 3 facts.
CLIPS> (facts A)
f-0     (datos 1)
f-1     (datos 2)
For a total of 2 facts.
```

Ejecución de Reglas

- Cada módulo tiene su agenda.
- *run* ejecuta la agenda del módulo enfocado.
- *Reset*, *clear* y la inicialización del sistema hacen que el módulo MAIN tenga el foco actual.
- Existe una pila de módulos enfocados cuyas agendas se van ejecutando secuencialmente.
- Una agenda se ejecuta hasta que cambia el foco a otro módulo, se terminan las activaciones o se ejecuta *return* en el consecuente de una regla.

Ejecución de Reglas

- El foco actual se cambia con *focus*. (`focus <nombre-módulo>+`)
- *focus* añade un nuevo foco a la pila de focos, pero no elimina el que había.
- Cuando se cambia el módulo actual no se cambia el foco.
- Cuando se cambia el foco sí se cambia automáticamente el módulo actual.
- *return* termina de manera inmediata la ejecución del consecuente y quita el foco actual de la pila de focos.

Ejemplo

```
CLIPS> (clear)
CLIPS> (defmodule MAIN (export ?ALL))
CLIPS> (defrule ejemplo =>
   (printout t "Foco en MAIN" crlf)
   (focus A B)) ;Añade a la pila de dcha. a izda.
CLIPS> (defmodule A (import MAIN deftemplate initial-
   fact))
CLIPS> (defrule ejemplo =>
   (printout t "Foco en A" crlf))
CLIPS> (defmodule B (import MAIN deftemplate initial-
   fact))
CLIPS> (defrule ejemplo =>
   (printout t "Foco en B" crlf))
CLIPS> (reset)
CLIPS> (run)
Foco en MAIN
Foco en A
Foco en B
```

Ejemplo

```
CLIPS> (clear)
CLIPS> (defmodule MAIN (export deftemplate initial-
   fact))
CLIPS> (defmodule A (import MAIN deftemplate initial-
   fact))
CLIPS> (defrule MAIN::inicio => (focus A))
CLIPS> (defrule A::r1 =>
   (return)
   (printout t "No imprime" crlf))
CLIPS> (defrule A::r2 =>
   (return)
   (printout t "No imprime" crlf))
```

Ejemplo (continúa)

```
CLIPS> (watch rules)
CLIPS> (watch focus)
CLIPS> (reset)
<== Focus MAIN
==> Focus MAIN
CLIPS> (run)
FIRE    1 inicio: f-0
==> Focus A from MAIN
FIRE    2 r1: f-0
<== Focus A to MAIN
<== Focus MAIN
```

Ejemplo

```
CLIPS> (clear)
CLIPS> (defmodule DETECCION)
CLIPS> (defmodule AISLAMIENTO)
CLIPS> (defmodule RECUPERACION)
CLIPS> (deffacts MAIN::informacion-control
   (secuencia-fases DETECCION AISLAMIENTO RECUPERACION))
CLIPS> (defrule MAIN::cambio-fase
   ?l <- (secuencia-fases ?siguiente $?resto)
   =>
   (focus ?siguiente)
   (retract ?l)
   (assert (secuencia-fases $?resto ?siguiente)))
CLIPS> (watch rules)
CLIPS> (watch focus)
```

Ejemplo (continúa)

```
CLIPS> (reset)
<== Focus MAIN
==> Focus MAIN
CLIPS> (run 4)
FIRE    1 cambio-fase: f-1
==> Focus DETECCION from MAIN
<== Focus DETECCION to MAIN
FIRE    2 cambio-fase: f-2
==> Focus AISLAMIENTO from MAIN
<== Focus AISLAMIENTO to MAIN
FIRE    3 cambio-fase: f-3
==> Focus RECUPERACION from MAIN
<== Focus RECUPERACION to MAIN
FIRE    4 cambio-fase: f-4
==> Focus DETECCION from MAIN
<== Focus DETECCION to MAIN
```