

Tema 10: Funciones genéricas

Constructores *defgeneric* y
defmethod

Introducción

- Similares a las funciones *deffunction* pero más potentes.
- Pueden sobrecargarse.
- Una función genérica se compone de varios **métodos**, cada uno de los cuales corresponde a una lista de parámetros diferente.
- Pueden tener funciones del sistema y funciones externas definidas por el usuario como métodos implícitos (ej.: '+' para sumar números y concatenar cadenas).
- CLIPS decide qué método de una función genérica ejecutar mediante el proceso conocido como *ejecución genérica* (generic dispatch).

Definición de funciones genéricas

- Una función genérica se corresponde con una cabecera y cero o más métodos.
- La cabecera se puede declarar
 - explícitamente, con el constructor *defgeneric*
 - implícitamente, con la definición de al menos un método

Definición de funciones genéricas

- Un método se compone de 6 elementos:
 1. Nombre
 2. Índice (opcional)
 3. Comentario (opcional)
 4. Conjunto de parámetros monocampo
 5. Parámetro multicampo (opcional)
 6. Secuencia de acciones
- El valor que devuelve un método es el resultado de evaluar su última acción, y FALSE si no tiene ninguna acción o si ocurre un error durante la ejecución.

Sintaxis

```
(defgeneric <nombre> [<comentario>])
```

```
(defmethod <nombre> [<índice>] [<comentario>]  
  (<restricción-param>*  
   [<restricción-param-multicampo>])  
  <acción>*)
```

```
<restricción-param> ::= <variable-monocampo> |  
  (<variable-monocampo> <tipo>* [<consulta>])
```

```
<restricción-param-multicampo> ::=  
  <variable-multicampo> |  
  (<variable-multicampo> <tipo>* [<consulta>])
```

Sintaxis

```
<tipo> ::=  INTEGER | FLOAT | NUMBER |  
            SYMBOL | STRING | LEXEME
```

```
<consulta> ::=  <variable-global> |  
                <llamada-función>
```

Índices

- Existen dos maneras de identificar un método:
 - Mediante un nombre y una lista de parámetros.
 - Mediante un nombre y un índice.
- A cada método se le asigna un índice único dentro del conjunto de métodos para esa función genérica.
- Si se define un método con el mismo nombre y los mismos parámetros o el mismo índice que otro existente, el nuevo reemplaza al anterior.
- Para reemplazar un método con otro nuevo que tenga una lista de parámetros distinta, se especifica el índice del método a reemplazar.

Parámetros

- Restricciones
 - de tipo: limita los tipos de dato admisibles para un argumento
 - de consulta: comprobación lógica que debe satisfacer un argumento para ser aceptado
- Una restricción de consulta se cumple si se evalúa a un valor distinto de FALSE.
- Las restricciones se comprueban de izquierda a derecha.
- Si una restricción no se cumple, no se evalúan las restantes.
- Un parámetro sin restricciones indica que acepta cualquier argumento.

Ejemplo

```
CLIPS> (defmethod + ((?a STRING) (?b STRING))
        (str-cat ?a ?b))
```

```
CLIPS>
```

```
(list-defmethods)
```

```
+ #SYS1 (NUMBER) (NUMBER) ($? NUMBER)
```

```
+ #2 (STRING) (STRING)
```

For a total of 2 methods.

```
CLIPS>
```

```
(+ 5 3)
```

```
8
```

```
CLIPS>
```

```
(+ "Hola " "mundo")
```

```
"Hola mundo"
```

```
CLIPS>
```

```
(+ "Me " "gusta " "CLIPS")
```

```
[GENRCXEXE1] No applicable methods for +.
```

```
FALSE
```

Ejemplo

```
(defmethod m1 ( ))
```

Ningún parámetro

```
(defmethod m1 (?a))
```

Un parámetro de cualquier tipo

```
(defmethod m1 ((?a INTEGER)))
```

Un parámetro entero

```
(defmethod m1 ((?a INTEGER SYMBOL)))
```

Un parámetro entero o símbolo

```
(defmethod m1 ((?a INTEGER (evenp ?a))))
```

Un parámetro entero par

```
(defmethod m1 ((?a INTEGER SYMBOL) ?b))
```

Un parámetro entero o símbolo y otro parámetro de cualquier tipo

Parámetros

- Si se especifica un parámetro multicampo, el método puede recibir un número de argumentos igual o mayor que un mínimo.
- Los argumentos que no correspondan a un parámetro monocampo se agrupan en un valor multicampo que puede referenciarse mediante el parámetro multicampo en el cuerpo (acciones) del método.
- Las restricciones se aplican a cada uno de los campos del parámetro multicampo.
- No obstante, en la consulta puede hacerse referencia al parámetro multicampo.
- Además, puede usarse la variable especial *?current-argument* para hacer referencia a cada uno de los campos del parámetro multicampo en la consulta. El ámbito de esta variable se limita a la consulta.

Ejemplo

```

CLIPS> (defmethod m1 (($?a INTEGER)))
CLIPS> ;Todos los parámetros deben ser enteros
(m1 1 2 3)
FALSE
CLIPS> (m1 1 2.2 3)
[GENRCEXE1] No applicable methods for m1.
FALSE
    
```

Ejemplo

CLIPS>

```
(defmethod m2 (($?a (printout t $?a crlf))))
```

CLIPS> ;Las restricciones se comprueban para cada
campo del parámetro multicampo

```
(m2 1 2 3)
```

```
(1 2 3)
```

```
(1 2 3)
```

```
(1 2 3)
```

FALSE

Ejemplo

CLIPS>

```
(defmethod m3 (($?a (printout t ?current-argument
    crlf))))
```

CLIPS> (m3 1 2 3)

1

2

3

FALSE

Ejemplo

CLIPS>

```
(defmethod m4 (($?a (> (length$ $?a) 2))))
```

CLIPS> (m4 1 2)

[GENRCEXE1] No applicable methods for m4.

FALSE

CLIPS> (m4 1 2 3)

FALSE

Ejemplo

CLIPS>

```
(defmethod m5 (($?a (evenp ?current-argument))))
```

CLIPS> (m5 1 2 3)

[GENRCEXE1] No applicable methods for m5.

FALSE

CLIPS> (m5 2 4 6)

FALSE

Ejemplo

CLIPS>

```
(defmethod m6 (?a ?b
                  ($?c INTEGER (evenp ?current-argument))))
```

CLIPS> ;Al menos 2 argumentos, de cualquier tipo
 ;Opcionalmente más argumentos, siempre que sean enteros
 pares

```
(m6 Hola)
```

[GENRCEXE1] No applicable methods for m6.

FALSE

```
CLIPS> (m6 Hola mundo)
```

FALSE

```
CLIPS> (m6 Hola mundo 1)
```

[GENRCEXE1] No applicable methods for m6.

FALSE

```
CLIPS> (m6 Hola mundo 2)
```

FALSE

Ejecución genérica

- Cuando se llama a una función genérica, CLIPS selecciona de entre los métodos cuyas restricciones de parámetros son satisfechas por los argumentos, el de mayor precedencia.
- El método seleccionado se ejecuta y el valor que devuelve es el valor devuelto por la función genérica.

Ejecución genérica

- Un método es aplicable a una llamada a función genérica si se cumplen las 3 condiciones siguientes:
 1. Su nombre coincide con el de la función genérica.
 2. Acepta al menos tantos argumentos como se pasan en la llamada.
 3. Cada argumento satisface las restricciones de parámetro correspondientes.
- La precedencia entre dos métodos se determina examinando sus restricciones de parámetro.
- En general, el método con las restricciones más específicas es el que tiene mayor precedencia.

Ejemplo

```
CLIPS> (defmethod m1 () m1-1)
CLIPS> (defmethod m1 (?a) m1-2)
CLIPS> (defmethod m1 ((?a INTEGER)) m1-3)
CLIPS> (defmethod m1 ((?a INTEGER SYMBOL)) m1-4)
CLIPS> (defmethod m1 ((?a INTEGER STRING)) m1-5)
CLIPS> (defmethod m1 ((?a INTEGER (evenp ?a))) m1-6)
CLIPS> (m1)
m1-1
CLIPS> (m1 2.5)
m1-2
CLIPS> (m1 25)
m1-3
CLIPS> (m1 Hola)
m1-4
CLIPS> (m1 22)
m1-6
```

Comandos relacionados

```
(ppdefgeneric <nombre-función-genérica>)
```

```
(ppdefmethod <nombre-función-genérica> <índice>)
```

```
(list-defgenerics [<nombre-módulo> | *])
```

```
(list-defmethods [<nombre-función-genérica>])
```

```
(undefgeneric <nombre-función-genérica> | *)
```

```
(undefmethod <nombre-función-genérica> | *  
  <índice> | *)
```