CLIPS Definición de funciones

Por función consideramos tanto las funciones predefinidas en CLIPS como las definidas por el usuario en este lenguaje, o las definidas en un lenguaje externo (C, FORTRAN o ADA) y enlazados con el entorno de CLIPS. En versiones previas de CLIPS este último tipo de funciones era la única forma de añadir funciones a CLIPS (teniendo que recompilar el propio CLIPS con cada nueva función)
Para definir una función utilizamos la construcción (deffunction)
El valor retornado por una función es el resultado de evaluar la última sentencia

Las funciones siguen una sintaxis tipo LISP con notación prefija: (id-fun arg1 arg2 …)
Polimorfismo: Mediante la construcción (defgeneric) podemos sobrecargar funciones y definir nuevos comportamientos en función del número y tipo de los argumentos.

Los elementos de una función son 5:

1. Nombre
2. Comentario (opcional)
3. Lista de cero o más argumentos obligatorios
4. Un argumentos para recibir el resto de argumentos recibidos (opcional)
5. La secuencia de acciones y expresiones que se ejecutarán

Sintaxis:

```
(deffunction <name> [<comment>]
  (<regular-parameter>* [<wildcard-parameter>])
  <action>*)

<regular-parameter> ::= <single-field-variable>
<wildcard-parameter> ::= <multifield-variable>
```

Se exige que una función esté definida antes de poder utilizarla en el cuerpo de otras. La única excepción es la autorecursión.

```
(deffunction print-args (?a ?b $?c)
  (printout t ?a " " ?b " and " (length ?c) " extras: " ?c
            crlf))
```

Esta función ejemplifica el uso de un número variable de argumentos, concretamente “al menos” de 2 argumentos.

```
CLIPS> (print-args 1 2)
1 2 and 0 extras: ()
CLIPS> (print-args a b c d)
a b and 2 extras: (c d)
CLIPS> (print-args 1)
[ARGACCES4] Function print-args expected at least 2 argument(s)
```

Forward Declaration: Si necesitamos recursión cruzada necesitamos declarar el prototipo de una de las funciones. Para ello la definimos como una función sin acciones.
Ejemplo:

```
 (deffunction foo ())
(deffunction bar ()
  (foo))
(deffunction foo ()
  (bar))
```

Funciones genéricas

(Consultar Section 8)

Funciones, Acciones y Ordenes predefinidas

En terminología de CLIPS se denomina acciones a aquellas funciones cuyo valor de retorno no es relevantes, sino que dan lugar a algún efecto lateral interesante (por ejemplo mostrar información por la consola). De forma similar se consideran ordenes (commands) a aquellas funciones que habitualmente se ejecutan directamente desde el la consola principal de CLIPS.

Funciones predicado: (Section 12.1)

Los predicados (i.e., funciones que devuelven un valor booleano) por convenio la última de su identificador es la letra *p*.
Dentro de ellos podemos destacar los que comprueban el tipo de un dato. Los que nos permiten comparar datos.
Comparando igualdad: `(eq <expression> <expression>+)`
Comparando desigualdad: `(neq <expression> <expression>+)`
Comparando igualdad valores numéricos: `(= <expression> <expression>+)`
Comparando desigualdad numérica: `(<> <expression> <expression>+)`
Otras comparaciones numéricas: `(<operador> <expression> <expression>+)`

```
<operador> ::= > | >= | < | <=
```

Operadores lógicos: `(and <expression>+)(or <expression>+)(not <expression>)`

Funciones sobre Listas (expresiones multivaluadas): Section 12.2

En este caso el convenio de nomenclatura es que sus funciones acaban en $ (por similitud con las variables multivalor que empiezan por $ también)

Para crear una lista utilizamos la siguiente sintaxis:

```
(create$ <expression>*)
```

Acceso aleatorio: (la posición inicial es la 1)

```
(nth$ <integer-expression> <multifield-expression>)
```

Buscar un elemento:

```
(member$ <expression> <multifield-expression>)
```

Borrar un subrango:

```
(delete$ <multifield-expression>
      <begin-integer-expression>
      <end-integer-expression>)
```

Reemplazar un subrango con una serie de valores:

```
 (replace$ <multifield-expression>
      <begin-integer-expression>
      <end-integer-expression>
      <single-or-multi-field-expression>+)
```

Insertar elementos:

```
(insert$ <multifield-expression>
       <integer-expression>
       <single-or-multi-field-expression>+)
```

CAR y CDR:

```
(first$ (create$ a b c))
(rest$ <multifield-expression>)
```

Longitud de una expresión:

```
(length$ <multifield-expression>)
```

Las siguientes son dos funciones de control, pero como están directamente relacionadas con las listas se comentan en este punto.

Ejecutar una acción por cada elemento de una lista:

```
(foreach <list-variable> <multifield-expression> <expression>*)
Example
CLIPS> (foreach ?field (create$ abc def ghi)
          (printout t "--> " ?field " " ?field-index " <--" crlf))
--> abc 1
--> def 2
--> ghi 3
CLIPS>
```

Ejecutar una acción por cada elemento de una lista y devolver el resultado evaluar el último elemento: se puede detener la secuencia con break o return.

```
(progn$ <list-spec> <expression>*)
<list-spec> ::= <multifield-expression> |
(<list-variable> <multifield-expression>)
Example
CLIPS> (progn$ (?field (create$ abc def ghi))
(printout t "--> " ?field " " ?field-index " <--" crlf))
--> abc 1 <--
--> def 2 <--
--> ghi 3 <--
CLIPS>
```

(otras funciones consultar manual en su sección 12.2)

Funciones sobre Strings

Evaluar una expresión contenida en un string:

```
(eval <string-or-symbol-expression>)
Ej:
CLIPS> (eval "(+ 3 4)")
7
```

(otras funciones consultar manual en su sección 12.3)

Funciones de Entrada/Salida

(Section 2.3.1: delimitadores a la hora de leer)
(otras funciones consultar manual en su sección 12.4)

Funciones matemáticas

(otras funciones consultar manual en su sección 12.5)

Funciones de control

Proporcionadas para cubrir necesidades propias de lenguajes imperativos.

Definición de variables globales: si no se indica valor se libera la variable.

```
(bind <variable> <expression>*)
```

Sentencia condicional:

```
(if <expression>
       then
         <action>*
       [else
         <action>*])
```

Bucle while:

```
(while <expression> [do]
       <action>*)
```

Bucle for: el valor por defecto de la primera iteración es 1.

```
(loop-for-count <range-spec> [do] <action>*)
<range-spec> ::= <end-index> |
                 (<loop-variable> <start-index> <end-index>) |
                 (<loop-variable> <end-index>)
<start-index> ::= <integer-expression>
<end-index> ::= <integer-expression>
```

Bloque de código: evalua todas las sentencias y devuelve el resultado de evaluar la última de ellas.

```
(progn <expression>*)
```

Interrumpir el bucle actual: while, loop-for-count, progn, progn$, foreach, o determinadas funciones de consulta sobre objetos (do-for-instance, do-for-all-instances and delayed-do-for-all-instances)

```
(break)
```

Instrucción de selección:

```
(switch <test-expression>
  <case-statement>*
  [<default-statement>])

<case-statement> ::= (case <comparison-expression> then <action>*)
<default-statement> ::= (default <action>*)
```

Otras

Generación de números aleatorios:

```
(seed <integer-expression>)
(random [<start-integer-expression> <end-integer-expression>])
```

(Consultar documentación de CLIPS: Basic Programming Guide para conocer el resto de funciones o buscar una función concreta)

- 2.3.2 – Functions (pág 36)
- 2.5.2 Procedural Knowledge (pág 43)
- 2.5.2.1 Deffunctions (pág 43)
- 2.5.2.2 Generic Functions (pág 44)
- Section 7 - Deffunction Construct (pág 99)

- Section 8 - Generic Functions (pág. 101)
- 12 - Actions And Functions (pág. 177)
- Appendix I: Lista completa de “los nombres” de las funciones definidas en el sistema