Generación de código para funciones

Índice

Marina de la Cruz
Alfonso Ortega

Identificación de los puntos en los que hay que ejecutar acciones

```
subroutine : TOK_FUNCTION mode TOK_ID ★ '('fn_prm_train ')'✚
TOK_BEGIN fn_dcl_train ★ stm_train TOK_END ☆
```

- Modificamos la gramática para ejecutar acciones en los puntos necesarios.

Generación de código para el identificador de una función

```
fn_name : TOK_FUNCTION mode TOK_ID ★
```

Generación de código para los parámetros formales de una función

- La marca ✚ representa un caso especial porque en ese punto no hay que realizar acciones, pero sí hay que ejecutarlas para cada uno de los parámetros del conjunto de parámetros de la función.

Generación de código previo a la sección de sentencias de una función

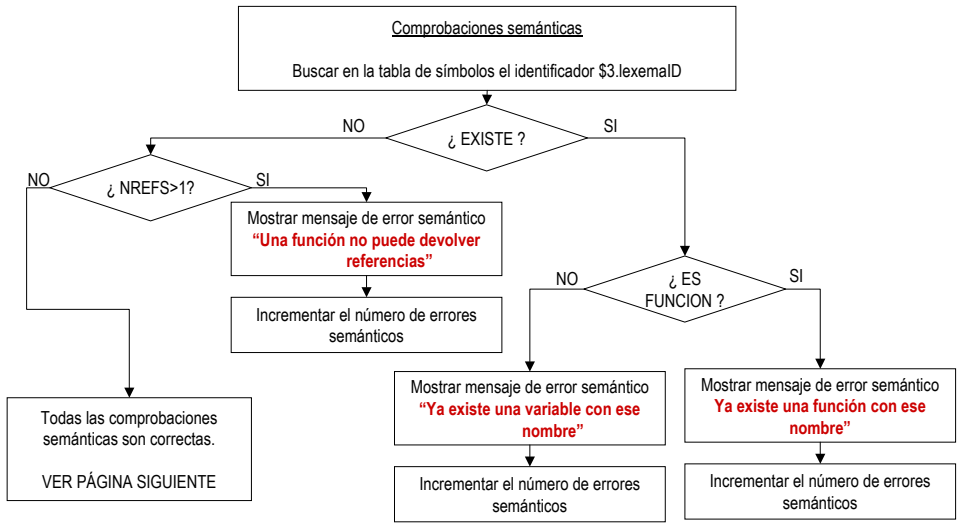
```
fn_declaration:fn_name '('fn_prm_train ')' TOK_BEGIN fn_dcl_train ★
```

Generación de código para el final de una función

```
subroutine : fn_declaration stm_train TOK_END ☆
```

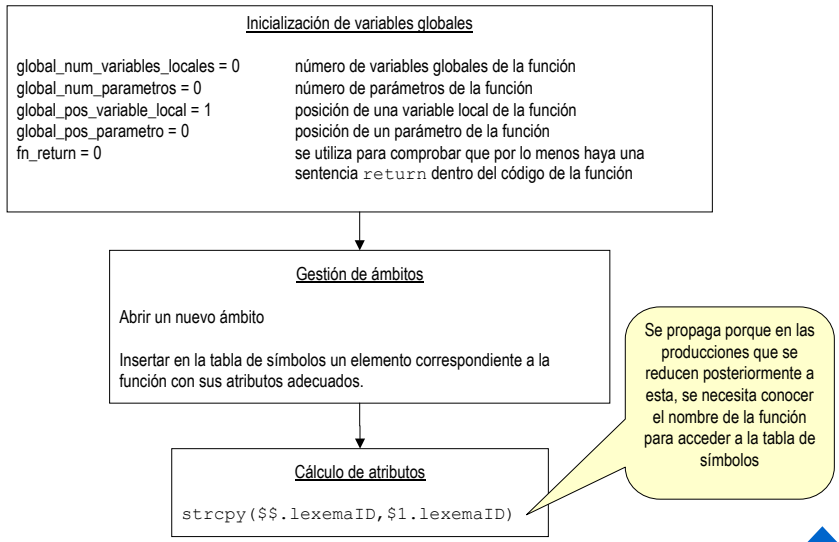

Generación de código para el identificador de una función (I)

```
fn_name : TOK_FUNCTION mode TOK_ID
```



Generación de código para el identificador de una función (II)

```
fn_name : TOK_FUNCTION mode TOK_ID
```



Generación de código para los parámetros formales de una función (I)

- En la lista de parámetros formales de una función, hay que incorporar una acción para gestionar cada parámetro. Para ello modificamos la gramática de la siguiente manera, sustituimos la producción

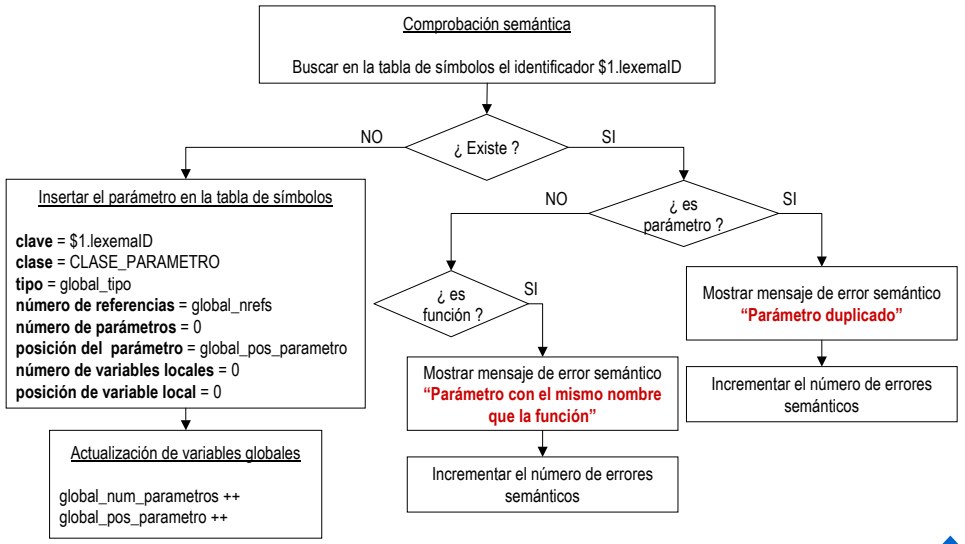
```
fn_parameter: mode TOK_ID
```

por las producciones:

```
fn_parameter: mode idpf
idpf: TOK_ID
```

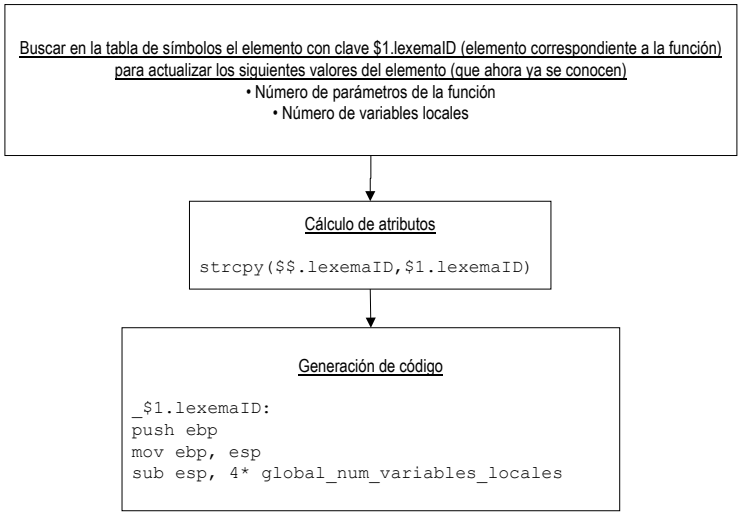

Generación de código para los parámetros formales de una función (II)

`idpf: TOK_ID`



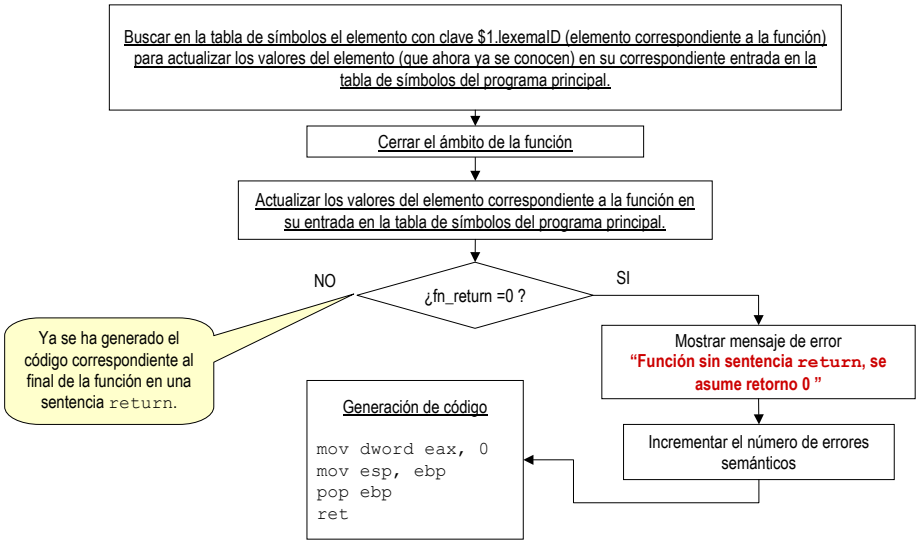
Generación de código previo a la sección de sentencias de una función

`fn_declaration:fn_name '(' fn_prm_train ')' TOK_BEGIN fn_dcl_train`



Generación de código para el final de una función

`subroutine : fn_declaration stm_train TOK_END`



Tratamiento de la sección de declaraciones de las funciones

```
fn_dcl_train : dcl_train
             |
```

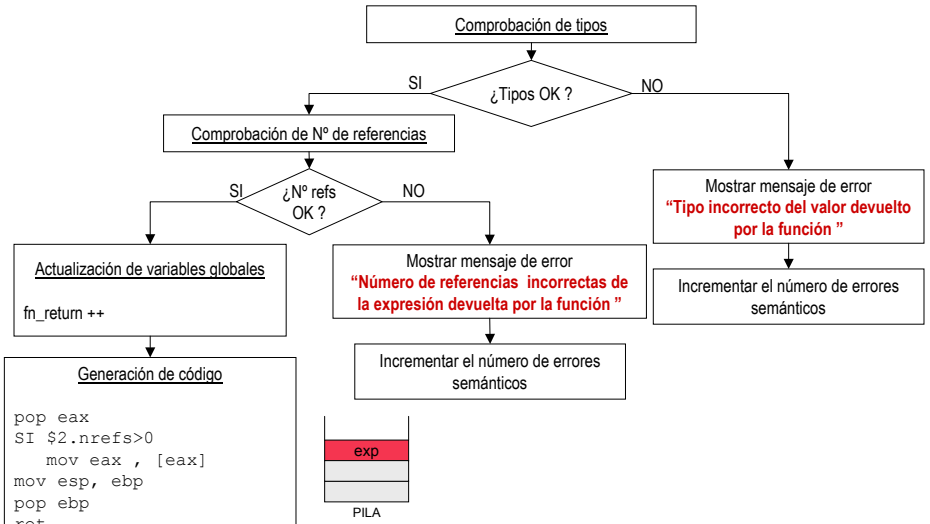
- La sección de declaraciones de una función difiere de la sección de declaraciones del programa principal en que las funciones pueden tener vacía dicha sección.
- El tratamiento de la declaración de una variable local de una función es similar al de una variable global, aunque utilicen tablas de símbolos diferentes y la información que se almacena de cada variable es diferente (por ejemplo, para las variables locales se guarda en la tabla de símbolos la posición de dicha variable).
- Como consecuencia de los dos puntos anteriores, para generar el código correspondiente a la sección de declaraciones de una función, únicamente hay que incorporar el la acción correspondiente a la declaración de una variable global el código necesario para implementar las diferencias entre variables locales y globales.
- Por lo tanto las acciones semánticas de las dos producciones del no terminal `fn_dcl_train` están vacías.

Tratamiento de la sección de sentencias de las funciones

- La sección de sentencias de una función también es “similar” a la sección de sentencias del programa principal. Únicamente hay que incorporar en algunas de las producciones de las sentencias el código necesario para la compilación correcta de la sección de sentencias de las funciones.

Generación de código para la sentencia `return`

```
statement : TOK_RETURN exp
```



Generación de código para la llamada a una función (I)

Modificación de la gramática (i)

- La producción para la llamada a una función en ASPLE es:

```
exp:   TOK_ID ★ '(' explist ')'
```

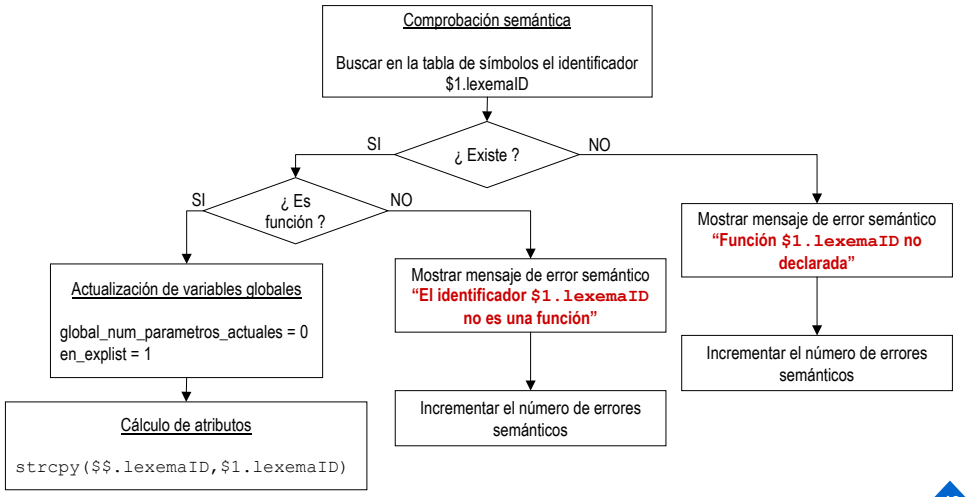
- Cada vez que se llama a una función hay que realizar una acción, en la posición marcada con el símbolo ★
- Como **las acciones semánticas se ejecutan cuando se reduce la regla**, es necesario modificar la gramática de la siguiente manera:

```
exp: idf '(' explist ')'
idf: TOK_ID ★
```

Generación de código para la llamada a una función (II)

Modificación de la gramática (ii)

```
idf: TOK_ID
```



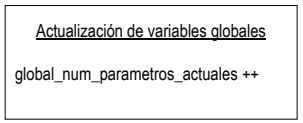
Generación de código para la llamada a una función (III)

Producciones de los parámetros actuales de la función

```
explist: exp rest_explist
```

```
rest_explist: ',' exp rest_explist
```

- Los no terminales `explist` y `rest_explist` tienen asociada la misma acción.



- Las producciones lambda de los no terminales `explist` y `rest_explist` no tienen asociada ninguna acción.

Generación de código para la llamada a una función (IV)

```
exp: idf '('explist')'
```