Compilación

Esquema de traducción dirigida por la sintaxis

Author:
Xabier Benavides Canta
Unai Lizarralde Imaz
David Sandström Daparte

May 13, 2019

Contents

1 Explicación de los símbolos y sus atributos

1.1 Símbolos no terminales

Los símbolos no terminales empleados para la resolución de la práctica son los que a continuación se detallan, así como los atributos de cada uno de ellos:

1. **lista_de_ident**: no terminal que almacena una lista de identificadores. Emplea el siguiente atributo:
   (a) .lnom: almacena todos los identificadores que puedan aparecer en ciertas situaciones.
2. **tipo**: no terminal que almacena los tipos de las variables declaradas. Emplea el siguiente atributo:
   (a) .clase: almacena el tipo de las variables, empleado en este caso para diferenciar las variables de tipo "real" y "entero".
3. **clase_par**: no terminal que almacena el tipo de cada uno de cada uno de los parámetros que se le pasan a una subrutina. Emplea el siguiente atributo:
   (a) .tipo: almacena el tipo de cada parámetro. En caso de que el parámetro sea de tipo "in" tomará el valor "val" y en caso de que el parámetro sea de tipo "out" o "in out" el valor del atributo será "ref".
4. **lista_de_sentencias**: no terminal que almacena un listado de sentencias leídas como parte del programa de entrada. Algunos ejemplos de las sentencias que puede almacenar son: **if**, **while**... Emplea el siguiente atributo:
   (a) .skip: almacena las referencias de todas las instrucciones skip almacenadas en la lista.
5. **sentencia**: no terminal que almacena las sentencias empleadas en el programa a traducir. Emplea el siguiente atributo
   (a) .skip: referencia en la que se encuentra la sentencia goto asociada a la instrucción.
6. **variable**: no terminal que almacena una de las variables del programa. Emplea el siguiente atributo:

   (a) .nombre: nombre de la variable.

7. **expresion**: no terminal que almacena una expresión aritmética o lógica. Emplea los siguientes atributos:

   (a) .true: almacena la referencia en la que se encuentra el goto que habrá de ejecutarse en caso de cumplirse la condición.

   (b) .false: almacena la referencia en la que se encuentra el goto que habrá de ejecutarse en caso de no cumplirse la condición.

   (c) .nombre: valor registrado por la expresión.

8. **M**: no terminal que almacena la referencia de la instrucción actual. Emplea el siguiente atributo:

   (a) .ref: empleado para almacenar la referencia de la instrucción.

1.2 Símbolos terminales

Los símbolos terminales empleados para la resolución de la práctica son los que a continuación se detallan, así como los atributos de cada uno de ellos:

1. **id**: terminal que guarda el identificador de una variable. Emplea el siguiente atributo:

   (a) .nombre: empleado para almacenar el identificador en las situaciones en que sea necesario.

2. **entero**: terminal que guarda un número entero. Emplea el siguiente atributo:

   (a) .nombre: valor del número entero.

3. **real**: terminal que guarda un número real. Emplea el siguiente atributo:

   (a) .nombre: valor del número real.

2 Descripción de las abstracciones funcionales empleadas

Las abstracciones funcionales empleadas son las siguientes:

- **anadir_inst(instruccion)**: recibe como entrada el código de una instrucción. Añade esa instrucción al codigo traducido.
- **nuevo_id()**: crea un nuevo id con la forma T_{k+1}, siendo k la cantidad de identificadores que se habian creado hasta el momento.
- **anadir_declaracion(lista, tipo)**: por cada nombre de la lista de entrada, empezando por el primero y hasta el último añade una instrucción de esta forma:

  $$nombre : tipo$$

- **anadir(lista, nombre)**: añade el nombre al comienzo de la lista de entrada y retorna la lista con dicha modificación aplicada.
- **inilista(nombre)**: crea una lista con el nombre que recibe como entrada y retorna la lista creada.
- **completar(valor, referencia)**: añade el valor indicado como parámetro de la función (valor) al final de la instrucción cuya referencia se indica mediante el parámetro referencia.
- **lista_vacia()**: retorna una lista vacía.
- **unir(**$lista_1$**,** $lista_2$**)**: retorna la unión de ambas listas.
- **escribir()**: una vez completada la lectura y traducción, vuelca todos los resultados en un fichero de texto.
- **vacio()**: función que retorna el valor *null* o lo que sea necesario asignar como vacío.

3 Esquema de traducción dirigido por la sintaxis

$$
\begin{aligned}
programa &\rightarrow program\ id\ \{anadir\_inst(prog||id.nombre||;);\} \\
&\quad bloque\ ;\ \{anadir\_inst(halt;); escribir();\}
\end{aligned}
$$

$$
bloque \rightarrow \{declaraciones\ decl\_de\_subprogs\ lista\_de\_sentencias\}
$$

$$
\begin{aligned}
declaraciones &\rightarrow var\ lista\_de\_ident\ :\ tipo;\ \{anadir\_declaracion(lista\_de\_ident.lnom,\ tipo.clase);\}\ declaraciones \\
&\quad | \ \xi
\end{aligned}
$$

$$
lista\_de\_ident \rightarrow id\ resto\_lista\_id\ \{lista\_de\_ident.lnom = anadir(resto\_lista\_id.lnom,\ id.nombre);\}
$$

$$
\begin{aligned}
resto\_lista\_id &\rightarrow ,id\ resto\_lista\_id\ \{resto\_lista\_id.lnom = anadir(resto\_lista\_id_1.lnom,\ id.nombre);\} \\
&\quad | \ \xi\ \{resto\_lista\_id.lnom = lista\_vacia();\}
\end{aligned}
$$

$$
\begin{aligned}
tipo &\rightarrow integer\ \{tipo.clase = "int";\} \\
&\quad | \ float\ \{tipo.clase = "real";\}
\end{aligned}
$$

$$
\begin{aligned}
decl\_de\_subprogs &\rightarrow decl\_de\_subprograma\ decl\_de\_subprogs \\
&\mid \xi
\end{aligned}
$$

$$
decl\_de\_subprograma \rightarrow procedure\ id\ \{anadir\_inst(proc||id.nombre||;);\}\ argumentos\ bloque;\{anadir\_inst(endproc;);\}
$$

$$
\begin{aligned}
argumentos &\rightarrow (lista\_de\_param) \\
&\mid \xi
\end{aligned}
$$

$$
\begin{aligned}
lista\_de\_param &\rightarrow lista\_de\_ident\ :\ clase\_par\ tipo\ \{anadir\_declaracion(lista\_de\_ident.lnom, clase\_par.tipo||\_||tipo.clase);\} \\
&\quad resto\_lis\_de\_param
\end{aligned}
$$

$$
\begin{aligned}
clase\_par &\rightarrow in\ \{clase\_par.tipo = "val";\} \\
&\mid out\{clase\_par.tipo = "ref";\} \\
&\mid in\ out\ \{clase\_par.tipo = "ref";\}
\end{aligned}
$$

$$
\begin{aligned}
resto\_lis\_de\_param &\rightarrow ;\ lista\_de\_ident\ :\ clase\_par\ tipo\ \{anadir\_declaracion(lista\_de\_ident.lnom, clase\_par.tipo||\_||tipo.clase);\} \\
&\quad resto\_lis\_de\_param \\
&\mid \xi
\end{aligned}
$$

$$
\begin{aligned}
lista\_de\_sentencias &\rightarrow sentencia\ lista\_de\_sentencias\ \{lista\_de\_sentencias.skip = unir(sentencia.skip, lista\_de\_sentencias_1.skip);\} \\
&\mid \xi\ \{lista\_de\_sentencias.skip = lista\_vacia();\}
\end{aligned}
$$

$$sentencia \rightarrow variable = expresion; \left\{ \begin{array}{c} anadir\_inst(variable.nombre|| := ||expresion.nombre;); \\ sentencia.skip = lista\_vacia(); \end{array} \right\}$$

$$| \; if \; expresion \; then \; \{M \; lista\_de\_sentencias\} M; \left\{ \begin{array}{c} completar(expresion.true, M_1.ref); \\ completar(expresion.false, M_2.ref); \\ sentencia.skip = lista\_de\_sentencias.skip \end{array} \right\}$$

$$| \; while \; M \; expresion \{M \; lista\_de\_sentenciasM\}; \left\{ \begin{array}{c} completar(expresion.true, M_2.ref); \\ completar(expresion.false, M_3.ref + 1); \\ anadir\_inst(goto||M_1.ref); \\ completar(lista\_sentencias.skip, M_1.ref); \\ sentencia.skip := lista\_vacia(); \end{array} \right\}$$

$$| \; do \; M \; \{ \; lista\_de\_sentencias\} \; until \; M \; expresion \; else \; M \; \{ \; lista\_de\_sentencias\} \; ; \left\{ \begin{array}{c} completar(expresion.true, M_1.ref); \\ completar(expresion.false, M_3.ref); \\ completar(lista\_sentencias_1.skip, M_2.ref); \\ sentencia.skip := lista\_vacia(); \end{array} \right\}$$

$$| \; skip \; if \; expresion \; M; \left\{ \begin{array}{c} completar(expresion.false, M.ref); \\ sentencia.skip = expresion.true; \end{array} \right\}$$

$$| \; read \; (variable); \left\{ \begin{array}{c} anadir\_inst(read||variable.nombre||;); \\ sentencia.skip = lista\_vacia(); \end{array} \right\}$$

$$| \; println \; (expresion); \left\{ \begin{array}{c} anadir\_inst(write||expresion.nombre||;); \\ anadir\_inst(writeln;); \\ sentencia.skip = lista\_vacia(); \end{array} \right\}$$

$$variable \rightarrow id\ \{variable.nombre = id.nombre;\}$$

$$M \rightarrow \xi\ \{M.ref = obtenref();\}$$

$$expresion \rightarrow expresion == expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| == ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$| \ expresion > \ \ expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| > ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$| \ expresion < \ \ expresion \left\{ \begin{array}{c} expresion.nombre = vacio(); \\ expresion.true = inilista(obtenref()); \\ expresion.false = inilista(obtenref() + 1); \\ anadir\_inst(if||expresion_1.nombre|| < ||expresion_2.nombre||goto); \\ anadir\_inst(goto); \end{array} \right\}$$

$$
\begin{array}{l}
\left.
\begin{array}{l}
| \; expresion \geq \; expresion
\end{array}
\right\{
\begin{array}{c}
expresion.nombre = vacio(); \\
expresion.true = inilista(obtenref()); \\
expresion.false = inilista(obtenref() + 1); \\
anadir\_inst(if||expresion_1.nombre|| \geq ||expresion_2.nombre||goto); \\
anadir\_inst(goto);
\end{array}
\right\} \\
\left.
\begin{array}{l}
| \; expresion \; \leq expresion
\end{array}
\right\{
\begin{array}{c}
expresion.nombre = vacio(); \\
expresion.true = inilista(obtenref()); \\
expresion.false = inilista(obtenref() + 1); \\
anadir\_inst(if||expresion_1.nombre|| \leq ||expresion_2.nombre||goto); \\
anadir\_inst(goto);
\end{array}
\right\} \\
\left.
\begin{array}{l}
| \; expresion \; \neq expresion
\end{array}
\right\{
\begin{array}{c}
expresion.nombre = vacio(); \\
expresion.true = inilista(obtenref()); \\
expresion.false = inilista(obtenref() + 1); \\
anadir\_inst(if||expresion_1.nombre|| \neq ||expresion_2.nombre||goto); \\
anadir\_inst(goto);
\end{array}
\right\} \\
\left.
\begin{array}{l}
| \; expresion + \; expresion
\end{array}
\right\{
\begin{array}{c}
expresion.nombre = nuevo\_id(); \\
expresion.true = vacio(); \\
expresion.false = vacio(); \\
anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| + ||expresion_2.nombre||;);
\end{array}
\right\}
\end{array}
$$

$$| \; expresion - expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| - ||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; expresion * \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre|| * ||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; expresion / \; expresion \left\{ \begin{array}{c} expresion.nombre = nuevo\_id(); \\ expresion.true = vacio(); \\ expresion.false = vacio(); \\ anadir\_inst(expresion.nombre|| := ||expresion_1.nombre||/||expresion_2.nombre||;); \end{array} \right\}$$

$$| \; id \quad \left\{ \begin{array}{c} expresion.nombre = id.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; n\_entero \quad \left\{ \begin{array}{c} expresion.nombre = entero.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; n\_real \quad \left\{ \begin{array}{c} expresion.nombre = real.nombre; \\ expresion.true = vacio(); \\ expresion.false = vacio(); \end{array} \right\}$$

$$| \; (expresion) \quad \left\{ \begin{array}{c} expresion.nombre = expresion_1.nombre; \\ expresion.true = expresion_1.true; \\ expresion.false = expresion_1.false; \end{array} \right\}$$