Tema 5 (2ª parte): Traductores ascendentes

Procesamiento de Lenguajes

Dept. de Lenguajes y Sistemas Informáticos
Universidad de Alicante

Algoritmo de análisis ascendente

$$E \longrightarrow E\ \textbf{op}\ T$$
$$E \longrightarrow T$$
$$T \longrightarrow \textbf{num}$$

	op	**num**	**$**	*E*	*T*
0		d3		1	2
1	d4		aceptar		
2	r2		r2		
3	r3		r3		
4		d3			5
5	r1		r1		

PILA	ENTRADA	ACCIÓN
0	1+2-3$	d3
0 3	+2-3$	r3
0 2	+2-3$	r2
0 1	+2-3$	d4
0 1 4	2-3$	d3
0 1 4 3	-3$	r3
0 1 4 5	-3$	r1
0 1	-3$	d4
0 1 4	3$	d3
0 1 4 3	$	r3
0 1 4 5	$	r1
0 1	$	aceptar

Algoritmo de análisis ascendente (2)

$E \longrightarrow E$ **op** T
$E \longrightarrow T$
$T \longrightarrow$ **num**

	op	**num**	**$**	E	T
0		d3		1	2
1	d4		aceptar		
2	r2		r2		
3	r3		r3		
4		d3			5
5	r1		r1		

PILA	ENTRADA	ACCIÓN
0	1+2-3**$**	d3
0 **num**	+2-3**$**	r3
0 T	+2-3**$**	r2
0 E	+2-3**$**	d4
0 E **op**	2-3**$**	d3
0 E **op num**	-3**$**	r3
0 E **op** T	-3**$**	r1
0 E	-3**$**	d4
0 E **op**	3**$**	d3
0 E **op num**	**$**	r3
0 E **op** T	**$**	r1
0 E	**$**	aceptar

Algoritmo de análisis ascendente (3)

$E \longrightarrow E$ **op** T
$E \longrightarrow T$
$T \longrightarrow$ **num**

	op	**num**	**$**	E	T
0		d3		1	2
1	d4		aceptar		
2	r2		r2		
3	r3		r3		
4		d3			5
5	r1		r1		

PILA	ENTRADA	ACCIÓN
0	1+2-3**$**	d3
0 **num** [1]	+2-3**$**	r3
0 T	+2-3**$**	r2
0 E	+2-3**$**	d4
0 E **op** [+]	2-3**$**	d3
0 E **op** [+] **num** [2]	-3**$**	r3
0 E **op** [+] T	-3**$**	r1
0 E	-3**$**	d4
0 E **op** [-]	3**$**	d3
0 E **op** [-] **num** [3]	**$**	r3
0 E **op** [-] T	**$**	r1
0 E	**$**	aceptar

Implementación de un ETDS

$$
\begin{array}{lcll}
E & \rightarrow & E\ \textbf{op}\ T & \{E.trad := \textbf{op}.trad || "(" || E_1.trad || "," || T.trad || ")"\} \\
E & \rightarrow & T & \{E.trad := T.trad\} \\
T & \rightarrow & \textbf{num} & \{T.trad := \textbf{num}.lexema\}
\end{array}
$$

Los atributos se almacenan en una pila paralela a la que tiene los estados del analizador, y las acciones se ejecutan al reducir

0				1+2-3$	d3
0	**num**			+2-3$	r3 $T.trad := \textbf{num}.lexema$
	1				
0	T			+2-3$	r2 $E.trad := T.trad$
	1				
0	E			+2-3$	d4
	1				

Implementación de un ETDS (2)

0	E	**op**		2-3$	d3
	1	+			
0	E	**op**	**num**	-3$	r3 $T.trad := \textbf{num}.lexema$
	1	+	2		
0	E	**op**	T	-3$	r1 $E.trad := \textbf{op}.trad \|\| "(" \|\| \ldots$
	1	+	2		
0	E			-3$	d4
	s(1,2)				
0	E	**op**		3$	d3
	s(1,2)	-			
0	E	**op**	**num**	$	r3 $T.trad := \textbf{num}.lexema$
	s(1,2)	-	3		
0	E	**op**	T	$	r1 $E.trad := \textbf{op}.trad \|\| "(" \|\| \ldots$
	s(1,2)	-	3		
0	E			$	aceptar
	r(s(1,2),3)				

Notación de `yacc` y `bison`

El programa `yacc` (y su sucesor `bison`) es un generador de compiladores (*yet another compiler-compiler*). A partir de ETDS en una notación determinada, genera un programa en C que analiza sintácticamente (usando un analizador ascendente LALR(1)) y a la vez traduce un programa fuente.

Ejemplo:

```
e : e OP t   { $$.trad=cat($2.trad,"(",$1.trad,",",$3.trad,")"); }
  | t        { $$.trad=$1.trad; }
  ;

t : NUM      { $$.trad=$1.lexema; }
  ;
```

(`$$` es la parte izquierda, `$1`, `$2`, ... son los símbolos de la parte derecha)

Implementación de ETDS

Resumen:

- Cada estado del analizador en la pila se puede asociar con un símbolo terminal o no terminal de la gramática (el símbolo con el que se llega a ese estado).
- Los atributos se almacenan en una pila paralela a la del analizador. Se implementa con dos vectores (estados y atributos) y un único índice para la cima de la pila
- Las acciones semánticas se ejecutan justo antes de reducir por una regla, cuando los atributos de los símbolos de la parte derecha están disponibles en la pila/vector. Los atributos del no terminal de la parte izquierda se guardan en una variable temporal, y se apilan al apilar el estado asociado a ese no terminal

Problemas:

1. ¿Cómo implementamos las acciones semánticas intermedias?
2. ¿Cómo implementamos los atributos heredados?

Implementación de acciones intermedias

En general, las acciones intermedias se utilizan para asignar valores a atributos heredados, pero en algún caso se usan para otras tareas, como almacenar valores en la tabla de símbolos. Por ejemplo:

$$
\begin{array}{lcl}
D & \rightarrow & T\ \textbf{id}\ \{\texttt{anadirTS}(\textbf{id}.\mathit{lexema}, T.\mathit{tipo})\} L\ \{\ldots\} \\
T & \rightarrow & \textbf{int}\ \{T.\mathit{tipo} := \texttt{ENTERO}\} \\
T & \rightarrow & \textbf{float}\ \{T.\mathit{tipo} := \texttt{REAL}\}
\end{array}
$$

La solución es utilizar por cada acción intermedia un nuevo no terminal, que llamaremos *marcador*, con una única regla que deriva a ϵ y al reducir por esa regla ejecutaremos la acción intermedia:

$$
\begin{array}{lcl}
D & \rightarrow & T\ \textbf{id}\ M\ L\ \{\ldots\} \\
M & \rightarrow & \epsilon\ \{\texttt{anadirTS}(\textbf{id}.\mathit{lexema}, T.\mathit{tipo})\} \\
T & \rightarrow & \textbf{int}\ \{T.\mathit{tipo} := \texttt{ENTERO}\} \\
T & \rightarrow & \textbf{float}\ \{T.\mathit{tipo} := \texttt{REAL}\}
\end{array}
$$

MUY IMPORTANTE: no es necesario modificar el ETDS (no es posible), solamente la tabla de análisis. Los atributos que necesita la acción están en la pila, más abajo.

Implementación de acciones intermedias (2)

$$
\begin{array}{lcl}
D & \rightarrow & T\ \textbf{id}\ M\ L\ \{\ldots\} \\
M & \rightarrow & \epsilon\ \{\texttt{anadirTS}(\textbf{id}.\mathit{lexema}, T.\mathit{tipo})\} \\
T & \rightarrow & \textbf{int}\ \{T.\mathit{tipo} := \texttt{ENTERO}\} \\
T & \rightarrow & \textbf{float}\ \{T.\mathit{tipo} := \texttt{REAL}\}
\end{array}
$$

0				int a$	desplazar
0	**int**			a$	reducir $T \rightarrow$ **int** $T.\mathit{tipo} := \texttt{ENTERO}$
0	T			a$	desplazar
	ENTERO				
0	T	**id**		$	reducir $M \rightarrow \epsilon$ anadirTS(**id**.lexema,T.tipo)
	ENTERO	a			
0	T	**id**	M	$	reducir $L \rightarrow \epsilon$ ...
	ENTERO	a			

Implementación de acciones intermedias (3)

El `yacc` y el `bison` generan automáticamente los marcadores para las acciones intermedias:

```
d : t ID { anadirTS($2.lexema,$1.tipo); } L   { ... }
  ;

t : INT   { $$.tipo = ENTERO; }
  | FLOAT { $$.tipo = REAL; }
  ;
```

Importante: los marcadores de las acciones intermedias ocupan una posición en la regla. En el ejemplo, el marcador es `$3`, y `L` es `$4`

Implementación de atributos heredados

- Aunque se conozca la posición en la pila del no terminal propietario del atributo heredado, no se le puede asignar un valor al atributo puesto que esa posición será ocupada por uno o más símbolos (los primeros de las partes derechas de las reglas del no terminal) antes que por el no terminal
- SOLUCIÓN:
  - Si el atributo heredado es un atributo sintetizado de otro símbolo, *bucear* en la pila hasta el atributo sintetizado, usando `$0`, `$-1`, `$-2`, ... para acceder a los símbolos que hay en la pila debajo de la parte derecha de la regla:

...	$-2	$-1	$0	$1	$2	$3	...

  - Si la posición en la pila del atributo sintetizado asociado al atributo heredado no es siempre la misma, hay que almacenar el atributo en un marcador para conseguir que la posición sea fija
  - Si el atributo heredado se obtiene a partir de otros atributos y/o constantes, se debe almacenar en un atributo sintetizado de un marcador

Implementación de atributos heredados (2)

Ejemplo 1:

$$
\begin{array}{lcl}
D & \rightarrow & T\ \{L.th = T.tipo\}L\ \{\ldots\} \\
L & \rightarrow & \{L_1.th = L.th\}L_1\ \textbf{coma}\ \textbf{id}\ \{\texttt{anadirTS}(\textbf{id}.lexema, L.th)\} \\
L & \rightarrow & \textbf{id}\ \{\texttt{anadirTS}(\textbf{id}.lexema, L.th)\} \\
T & \rightarrow & \textbf{int}\ \{T.tipo := \texttt{ENTERO}\} \\
T & \rightarrow & \textbf{float}\ \{T.tipo := \texttt{REAL}\}
\end{array}
$$

En `yacc`, se escribiría de esta manera, asociando *L*.th con *T*.tipo (que es `$0.tipo`):

```
d : t l  { ... }
  ;

l : l COMA ID { anadirTS($3.lexema,$0.tipo); }
  | ID        { anadirTS($1.lexema,$0.tipo); }
  ;

t : ...
```


Implementación de atributos heredados (3)

Ejemplo 1 (2):

```
d : t l  { ... }
  ;

l : l COMA ID { anadirTS($3.lexema,$0.tipo); }
  | ID        { anadirTS($1.lexema,$0.tipo); }
  ;

t : ...
```

0	*T*	**id**	
	ENTERO	a	
	$0	$1	

red. $L \rightarrow$ **id** `anadirTS`(id.lexema, *T*.tipo)

```
anadirTS($1.lexema,$0.tipo);
```

0	*T*	*L*	**coma**	**id**	
	ENTERO			a	
	$0	$1	$2	$3	

red. $L \rightarrow L$ **coma** **id** `anadirTS`(id.lexema, *T*.tipo)

```
anadirTS($3.lexema,$0.tipo);
```

Implementación de atributos heredados (4)

Ejemplo 2:

$$
\begin{array}{lcl}
D & \rightarrow & T\ \{L.th = T.tipo\}L\ \{\ldots\} \\
D & \rightarrow & T\ \textbf{var}\ \{L.th = T.tipo\}L\ \{\ldots\} \\
L & \rightarrow & \{L_1.th = L.th\}L_1\ \textbf{coma}\ \textbf{id}\ \{\texttt{anadirTS}(\textbf{id}.lexema, L.th)\} \\
L & \rightarrow & \textbf{id}\ \{\texttt{anadirTS}(\textbf{id}.lexema, L.th)\} \\
T & \rightarrow & \ldots
\end{array}
$$

Se puede seguir asociando *L*.th con *T*.tipo, pero están a diferente distancia en las dos reglas. Solución: igualar las distancias.

```
d : t VAR l  { ... }
  | t {}  l  { ... }   // el marcador ocupa el lugar de VAR
  ;

l : l COMA ID { anadirTS($3.lexema,$-1.tipo); }
  | ID        { anadirTS($1.lexema,$-1.tipo); }
  ;
```


Implementación de atributos heredados (5)

Ejemplo 3:

$$
\begin{array}{lcl}
D & \rightarrow & T\ \textbf{id}\ \{L.th = T.tipo; \texttt{anadirTS}(\textbf{id}.lexema, T.tipo)\}L\ \{\ldots\} \\
L & \rightarrow & \textbf{coma}\ \textbf{id}\ \{L_1.th = L.th; \texttt{anadirTS}(\textbf{id}.lexema, L.th)\}L_1\ \{\ldots\} \\
L & \rightarrow & \epsilon \\
T & \rightarrow & \ldots
\end{array}
$$

No se puede asociar *L*.th con *T*.tipo porque la distancia es variable (en este ejemplo depende del número de **id** que aparezcan). Solución: almacenar *L*.th en el marcador de la acción intermedia (usando `$$` nos referimos a los atributos del marcador).

```
d : t ID { anadirTS($2.lexema,$1.tipo); $$.tipo=$1.tipo; } l  { .
  ;

l : COMA ID { anadirTS($2.lexema,$0.tipo); $$.tipo=$0.tipo; } l
  |  /* epsilon */
  ;
```

Implementación de atributos heredados (6)

Ejemplo 4: ejercicio 3

S	$\longrightarrow$	P
P	$\longrightarrow$	**procedure id ;** L **endproc ;**
P	$\longrightarrow$	**function id :** T **;** L **endfunc ;**
L	$\longrightarrow$	P L
L	$\longrightarrow$	**codigo**
T	$\longrightarrow$	**integer**
T	$\longrightarrow$	**real**

```
procedure p1;
  procedure p2;
    codigo
  endproc;
  function f1:real;
    codigo
  endfunc;
  procedure p3;
    procedure p4;
      codigo
    endproc;
    codigo
  endproc;
  codigo
endproc;
```

```
void p1();
void p1_p2();
float p1_f1();
void p1_p3();
void p1_p3_p4();
```

- Se usa un atributo heredado para pasar un prefijo hacia abajo en el árbol
- El atributo heredado se calcula a partir de otro atributo y una constante "_"

Implementación de atributos heredados (7)

Ejemplo 4 (2): ejercicio 3, solución con `yacc/bison`

```
s : { $$.p=""; }  p
  ;

p : PROCEDURE ID PYC          { $$.p = $0.p=="" ? $2.lexema : $0.p+"_"+$2.lexema; }
    l ENDPROC PYC             { $$.trad = "void "+ $4.p + "();\n" + $5.trad; }
  | FUNCTION ID DOSP t PYC    { $$.p = $0.p=="" ? $2.lexema : $0.p+"_"+$2.lexema; }
    l ENDFUNC PYC             { $$.trad = $4.trad+" "+ $6.p + "();\n" + $7.trad; }
  ;

l : p             { $$.p = $0.p; }
    l             { $$.trad = $1.trad+$3.trad; }
  | CODIGO        { $$.trad = ""; }
  ;

t : INTEGER       { $$.trad = "int"; }
  | REAL          { $$.trad = "float"; }
  ;
```

- Inicialmente, en la regla de S se asigna "" al atributo `p` y se guarda en el marcador de la acción `$$.p="";`
- En las reglas de P, después del **pyc**, se calcula el nuevo valor del atributo heredado y se almacena en el marcador de la acción. En la acción situada al final de las reglas se utiliza esa atributo ya calculado (*trampa* de implementación) para construir la traducción
- En la regla recursiva de L, se guarda en el marcador de la acción el atributo heredado para que siempre esté en `$0` (y para evitar que se *aleje* porque la regla es recursiva por la derecha)

Implementación de atributos heredados (8)

Resumen:

- Es necesario asociar un atributo sintetizado de otro símbolo (que podría ser un marcador) con el atributo heredado
- Se debe asegurar que el atributo sintetizado asociado está siempre a la misma distancia en la pila en todas las reglas en que se use el atributo heredado
- Se debe intentar evitar el uso de atributos heredados, rediseñando la gramática si fuese necesario. Si hay atributos heredados y/o acciones intermedias, es siempre preferible la recursividad por la izquierda a la recursividad por la derecha
- Más información: páginas 130-150 del libro *Diseño de compiladores*, de A. Garrido *et al.*

Ejercicio 4 con `yacc/bison`

$$
\begin{array}{lcl}
S & \longrightarrow & C \\
C & \longrightarrow & \textbf{class id \{ } B\ V \textbf{ \}} \\
B & \longrightarrow & \textbf{public : } P \\
B & \longrightarrow & \epsilon \\
V & \longrightarrow & \textbf{private : } P \\
V & \longrightarrow & \epsilon \\
P & \longrightarrow & D\ P \\
P & \longrightarrow & \epsilon \\
D & \longrightarrow & T\ \textbf{id (}\ T\ \textbf{id}\ L\ \textbf{)} \\
D & \longrightarrow & C \\
L & \longrightarrow & \textbf{,}\ T\ \textbf{id}\ L \\
L & \longrightarrow & \epsilon \\
T & \longrightarrow & \textbf{int} \\
T & \longrightarrow & \textbf{float}
\end{array}
$$

```
class A {
  public:
    int f1(int n,float s)
  private:
    class B {
      private:
        float f2 (float r,float s,float t)
        class C {}
    }
}
```

```
clase A {
  público:
    A::f1 (entero x real -> entero)
  privado:
    clase A::B {
      privado:
        A::B::f2 (real x real x real -> real)
        clase A::B::C {}
    }
}
```

Ejercicio 4 con `yacc/bison` (solución)

```
S    : { $$.ph = ""; }   C                  { cout << $2.trad << endl; }
     ;

C    : tkclass id lbra   { $$.ph=$0.ph+$2.lexema+"::"; }
       B
       V rbra            { $$.trad="clase "+$0.ph+$2.lexema+"{\n"+$5.trad+$6.trad+"}\n"; }
     ;

B    : tkpublic dosp {$$.ph = $0.ph;} P       { $$.trad="público:\n"+$4.trad; }
     |                                        { $$.trad=""; }
     ;

V    : tkprivate dosp {$$.ph = $-1.ph;} P     { $$.trad="privado:\n"+$4.trad; }
     |                                        { $$.trad=""; }
     ;

P    : D { $$.ph = $0.ph; } P  { $$.trad=$1.trad+$3.trad; }
     |                         { $$.trad=""; }
     ;

D    : T id lpar T id L rpar { $$.trad=$0.ph+$2.lexema+"("+$4.trad+$6.trad+" -> "+$1.trad+")\n"; }
     | C                     //  $$ = $1
     ;

L    : coma T id L      { $$.trad = " x "+$2.trad+$4.trad; }
     |                  { $$.trad = ""; }
     ;

T    : tkint            { $$.trad = "entero"; }
     | tkfloat          { $$.trad = "real"; }
     ;
```