Analizadores sintácticos
Construcción con Yacc I

M. Luisa González Díaz

La fase de análisis sintáctico debe construir de algún modo un árbol de derivación, partiendo de la cadena de componentes léxicos de entrada y sobre la base de una descripción sintáctica (*gramatical*) del lenguaje. Tal descripción suele ser un gramática *independiente del contexto*.

Un árbol de derivación tiene como etiqueta de su raíz al *símbolo inicial* de la gramática, que se desarrolla en sucesivos niveles de nodos, etiquetados con símbolos *auxiliares*, siguiendo las reglas, hasta las hojas, que estarán etiquetadas con símbolos *terminales* de la gramática, ahora componentes léxicos.

La construcción de tal árbol puede hacerse partiendo de la raíz hacia las hojas o partiendo de las hojas hasta conseguir la raíz. En el primer caso, se dirá que el análisis se hace de forma *descendente* y en el segundo *ascendente*; en ambos, se intenta construir progresivamente el árbol leyendo la cadena de izquierda a derecha y empleando el mínimo número de símbolos necesario para tomar las decisiones adecuadas.

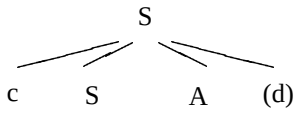
1. Analizadores descendentes deterministas

Tomemos como ejemplo la gramática siguiente:

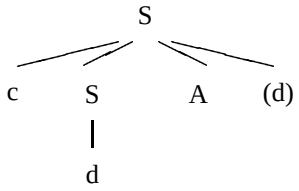
$$\begin{array}{lcl} S & \rightarrow & cSAd \mid d \\ A & \rightarrow & aB \mid a \\ B & \rightarrow & a \mid b \end{array}$$

frente a la cadena de entrada *cdad*. Un análisis descendente partiría de la hipótesis de que la cadena admite un árbol de derivación de raíz S, y pediría al analizador léxico el primer símbolo de la entrada, en este caso *c*. Este primer símbolo guía al analizador a elegir la regla $S \rightarrow cSAd$ y construir el siguiente nivel del árbol. Si el símbolo de entrada hubiese sido *d* se habría elegido la segunda regla, y cualquier otro símbolo de entrada habría bastado para marcar un error sintáctico y detener el análisis (aunque en realidad, y para que el proceso sea más útil, el análisis no se detendrá, sino que continuará después de hacer alguna suposición).

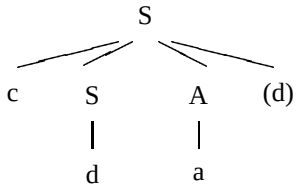
Se tiene entonces parte del árbol:



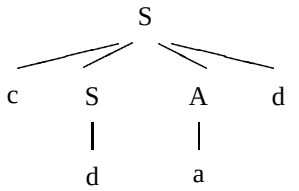
A continuación, se consideraría el siguiente nodo a desarrollar, en un orden de recorrido en profundidad del árbol: S. Hay dos posibilidades: $S \rightarrow cSAd$ ó $S \rightarrow d$. Se pide un nuevo símbolo al analizador léxico, que proporciona *d*, de modo que se elige la segunda opción, para tener



En el siguiente paso se considera el desarrollo del auxiliar A frente a la cadena de entrada que resta (ad). El siguiente símbolo que proporciona el analizador léxico es a, que no permite decidir entre las dos reglas posibles ($A \to aB$ ó $A \to a$). Lo que ocurre es que esta gramática requiere dos símbolos para tomar las decisiones, de modo que se solicitan los dos símbolos siguientes al analizador léxico, que proporciona a y d. Como lo que se derive de B no puede empezar por d, se elige la regla $A \to a$ para tener el árbol



y, finalmente, se termina de recorrer el árbol para comprobar que el símbolo siguiente de entrada (d) es el esperado, y que no hay ningún otro símbolo pendiente de ser leído (el analizador léxico proporciona la marca de fin de cadena, lo que indicaremos con el símbolo \$). El árbol que se ha recorrido es



Cuando una gramática permite realizar este tipo de análisis sin requerir más que un símbolo de entrada cuando se le solicita al analizador léxico se dice que la gramática es $LL(1)$. No toda gramática es de este tipo. Por ejemplo, la gramática del ejemplo no es $LL(1)$, pero sí lo es su equivalente:

$$\begin{cases} S & \to & cSAd \mid d \\ A & \to & aR \\ R & \to & B \mid \lambda \\ B & \to & a \mid b \end{cases}$$

y las dos siguientes son también equivalentes entre sí, la primera $LL(1)$ y la segunda no.

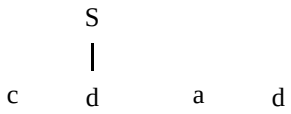
$$\begin{cases} S & \to & cSAd \mid d \\ A & \to & bA' \\ A' & \to & aA' \mid \lambda \end{cases} \qquad \begin{cases} S & \to & cSAd \mid d \\ A & \to & Aa \mid b \end{cases}$$

2. Analizadores ascendentes deterministas

Tomando como base la misma gramática anterior y la misma cadena, el proceso ahora consiste en considerar inicialmente la cadena y construir el árbol a partir de las hojas a medida que se obtienen los símbolos que la constituyen.

El árbol de derivación para la cadena, que se obtuvo en el apartado anterior, servirá de guía para mostrar cómo se realiza el análisis, aunque de momento no determinaremos exactamente en qué se basan las decisiones.

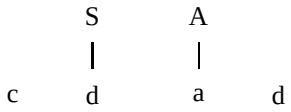
El analizador sintáctico solicita símbolos al analizador sintáctico hasta que consigue una subcadena que sea un consecuente de regla *adecuado*, en este caso la primera *d*, para construir una pequeña parte del árbol:



Con esto se ha obtenido el último paso de una derivación:

$$cS\,ad \Rightarrow c\,sad$$

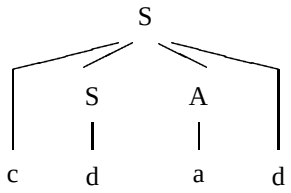
El problema ahora se reduce a construir un árbol para la *forma sentencial* $cS\,ad$, en la que aún no se ha leído *ad*. El siguiente símbolo que proporciona el analizador léxico, *a*, permite construir otro fragmento:



y se tiene el final de la derivación:

$$cS\,Ad \Rightarrow cS\,ad \Rightarrow c\,sad$$

Ahora la forma sentencial es $cS\,Ad$, que es un consecuente para el símbolo inicial de la primera regla, de forma que se ha conseguido el símbolo inicial cuando no hay más símbolos de entrada:



2.1. Análisis por desplazamiento-reducción (shift-reduce)

El análisis ascendente requiere, como se ha visto, la localización, en las formas sentenciales que se van obteniendo, de un consecuente de regla apropiado. En el ejemplo, las formas sentenciales son $c\underline{d}ad$, $cS\,\underline{a}d$, $\underline{cS\,Ad}$ y *s* en las que se ha subrayado los consecuentes de reglas elegidos.

En el primer paso, para la forma sentencial *cdad* hay tres consecuentes posibles: la subcadena *d* de la posición 2, la *d* de la posición 4 (en ambos casos consecuentes de la segunda regla $S \rightarrow d$), y la subcadena *a* (consecuente de las reglas 4 y 5).

En el árbol de derivación que debe obtenerse finalmente se observa que la segunda *d* no va a utilizarse como consecuente en la derivación (mientras que la primera *d* y la *a* sí), de forma que elegir la segunda *d* llevaría a un camino sin salida. Es decir, hay dos subcadenas *candidatas*, con posibilidades de éxito. De ellas, como la lectura se hace de izquierda a derecha, se elegirá la primera: la *d* de la posición 2 y la regla $S \rightarrow d$, para obtener la forma sentencial $cS.ad$, donde el punto indica en qué posición de lectura se encuentra el analizador. Como, dada una lista de candidatas en una forma sentencial de las que se van obteniendo, lo natural es elegir la que primero aparece, a su derecha sólo aparecerán símbolos terminales, que no es necesario haber leído aún. Este hecho permite trabajar apilando los símbolos que se han leído, y *desplazando* consecutivamente los que queden en la entrada, hasta el momento justo en que se haya conseguido la subcadena candidata elegida.

Cada vez que se sustituye el consecuente de una regla por el antecedente en este proceso, se dice que se ha realizado una *reducción*, y cada reducción produce una nueva forma sentencial, que sería la anterior en una derivación que reflejara el árbol. Concretamente, la derivación que se deduce del ejemplo es:

$$S \Rightarrow \underline{cSAd} \Rightarrow cS\underline{ad} \Rightarrow c\underline{d}ad$$

En el siguiente paso, cuando se realiza otra lectura (petición al analizador léxico), la situación es $cSa.d$, y la subcadena candidata está en la cima de la pila (de nuevo, aunque también *d* sea consecuente de una regla, su elección sería equivocada). Pero además, ahora hay dos posibles reglas para el consecuente *a*, y también la elección de $B \rightarrow a$ sería incorrecta. Debe elegirse $A \rightarrow a$ para obtener $cSA.d$

El tercer paso realiza otra lectura o desplazamiento, que consigue poner en la pila $cSAd$ para realizar la última reducción a S y terminar el análisis.

El proceso muestra una sucesión de formas sentenciales de una derivación en orden inverso (desde la cadena de terminales hasta el símbolo inicial). La derivación obtenida es del tipo "más a la derecha" y se resume completamente como se ve a continuación:

Pila	Entrada	Acción
\$	cdad\$	desplazar
\$c	dad\$	desplazar
\$cd	ad\$	reducir $S \rightarrow d$
\$cS	ad\$	desplazar
\$cSa	d\$	reducir $A \rightarrow a$
\$cSA	d\$	desplazar
\$cSAd	\$	reducir $S \rightarrow cSAd$
\$S	\$	aceptar

La dificultad está en encontrar el consecuente adecuado β en cada forma sentencial derecha $\alpha\beta x$, y decidir cuál es la regla $A \rightarrow \beta$ que debe aplicarse, para obtener la forma sentencial anterior de una derivación más a la derecha:

$$\alpha Ax \Rightarrow \alpha\beta x$$

A esto es a lo que se denomina **pivote** de la forma sentencial ("*handle*"). Un pivote es por lo tanto

- una subcadena β de la forma sentencial (como subcadena, es decir, localizada en la forma sentencial por su posición)
- una regla cuyo consecuente es β

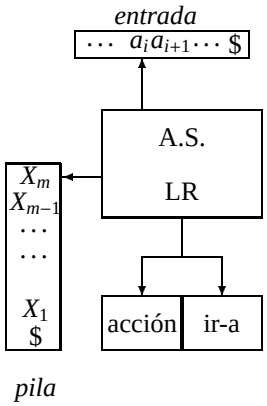
Los pivotes son en el ejemplo, respectivamente:

forma sentencial	*pivote*	
	subcadena	regla
$cdad$	*d* de la posición 2	$S \rightarrow d$
$cSad$	*a*	$A \rightarrow a$
$cSAd$	$cSAd$	$S \rightarrow cSAd$

La localización del pivote se traduce en decidir cuándo hay que realizar un desplazamiento. Si no hay que realizar desplazamiento, es porque hay que realizar una reducción, estando el consecuente en la cima de la pila. Entonces, la dificultad está en elegir la regla por la que hay que reducir. Se puede realizar por lo tanto este análisis con comodidad si se dispone de criterios que permitan, dada una situación de la pila y un símbolo de entrada, decidir

- si hay que desplazar o reducir
- caso de que haya que reducir, por cuál de las reglas posibles

Un análisis por desplazamiento-reducción responde al siguiente esquema:



El analizador sintáctico contiene una pila y dispone de unas tablas que guían sus decisiones. La *configuración* del analizador está determinada por

- el contenido de la pila
- la cadena de entrada que resta por leer

que puede resumirse, en un momento dado, por el par

$$(X_1 X_2 \ldots X_m, a_i a_{i+1} \ldots \$)$$

Conocidos el contenido de la pila ($X_0 \ldots X_m$) y el síguiente símbolo a leer (a_i), el analizador sintáctico accede a la tabla "acción", que especificará una de estas cuatro posibilidades:

- desplazar
- reducir por una determinada regla
- aceptar
- marcar error

Si la acción prescrita es desplazar, apilará el símbolo de entrada en la pila, pasando a la configuración

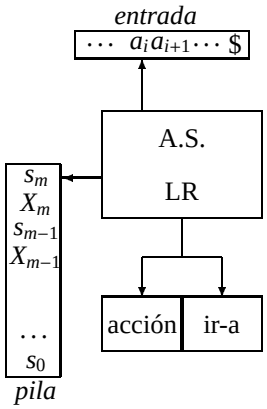
$$(X_1 X_2 \ldots X_m a_i, a_{i+1} \ldots \$)$$

Si es reducir, la regla indicada será de la forma $A \to X_{m-r} \ldots X_{m-1} X_m$, y se quitará el consecuente de la pila para apilar el antecedente, pasando a la configuración

$$(X_1 X_2 \ldots X_{m-r-1} A, a_i a_{i+1} \ldots \$)$$

Si la acción es "aceptar", terminará el análisis; si es "marcar error", lo marcará y pasará a realizar alguna rutina de recuperación.

Como examinar el contenido completo de la pila sería complejo, se resume el estado en el que se encuentra, codificándolo mediante *estados* (numéricos: s_j), que se intercalarán entre los símbolos. Cada estado resume la información del contenido de la pila que está por debajo.



La tabla *acción* tiene como coordenadas los estados y los símbolos de entrada, además del símbolo de fin de cadena. En cada casilla $[s_j, a_i]$ se especifica una de las cuatro acciones siguientes:

desplazar *s* que significa que se debe desplazar el símbolo de entrada a la pila, y colocar encima el número de estado *s*

reducir *r* que significa que se debe reducir por la regla especificada, es decir, eliminar el consecuente de la regla *r* de la pila, con los números de estado que están intercalados, y apilar en su sutitución el antecedente de *r*. Es necesario ahora apilar un número de estado que resuma lo sucedido, que se encontrará en la tabla *ir-a*.

aceptar que significa que el análisis ha terminado con éxito

error que significa que la cadena de entrada no se ajusta a la gramática especificada

La tabla *ir-a* tiene como coordenadas los estados y los símbolos auxiliares de la gramática, y se usa en las reducciones. Cuando se elimina un consecuente de la pila, queda al descubierto un estado s_p. Después de apilar el antecedente *A*, el número de estado que hay que colocar encima se encontrará en la casilla $[s_p, A]$ de esta tabla.

Para que el análisis pueda hacerse como se está indicando, es necesario que en cada casilla de las tablas mencionadas haya un sólo valor. Las gramáticas que cumplen este requisito son las llamadas de tipo *LR* en general.

Yacc realiza el cálculo de las tablas para un conjunto restringido de gramáticas *LR*, llamadas *LALR*(1). Para el ejemplo que se está siguiendo, las tablas calculadas por Yacc son

		ACCIÓN					**IR-A**		
estado y significado		a	b	c	d	$end	S	A	B
0	$accept:.S$end (0)			s1	s2		3		
1	S : c.SAd (1)			s1	s2		4		
2	S: d. (2)	r2	r2	r2	r2	r2			
3	$accept: S.$end					accept			
4	S : cS.Ad (1)	s5						6	
5	A:a.B (3); A:a. (4)	s7	s8		r4				9
6	S : cSA.d (1)				s10				
7	B : a. (5)	r5	r5	r5	r5	r5			
8	B : b. (6)	r6	r6	r6	r6	r6			
9	A : aB. (3)	r3	r3	r3	r3	r3			
10	S : cSAd. (1)	r1	r1	r1	r1	r1			

"s" significa "shift"(desplazar), y "r" reducir. El número que sigue a cada "s" es el estado que hay que apilar, y el que sigue a cada "r", el número de regla que hay que aplicar. Los números entre paréntesis son los números de regla. Cada estado representa un momento de recorrido de una regla,

marcado con un punto en la misma (o varios, como en el estado 5). Se añade a la gramática un nuevo símbolo $*accept*, inicial, y una regla (0) que siempre es $*accept* $\rightarrow$ *S* $*end*, para asegurar que una cadena de entrada que se ajuste a *S* no va seguida de nada más.

Ahora la simulación se transforma en:

Pila	Entrada	Acción	
$ 0	cdad$	s1(desplazar)	
$ 0 c 1	dad$	s2	
$ 0 c 1 d 2	ad$	r2 (reducir $S \rightarrow d$);	ir-a[1,S]=4
$ 0 c 1 S 4	ad$	s5	
$ 0 c 1 S 4 a 5	d$	r4 ($A \rightarrow a$);	ir-a[4,A]=6
$ 0 c 1 S 4 A 6	d$	s10	
$ 0 c 1 S 4 A 6 d 10	$	r1 ($S \rightarrow cSAd$);	ir-a[0,S]=3
$ 0 S 3	$	accept	

Obsérvese que cada reducción sencillamente cuenta el número de símbolos del consecuente y retira de la pila el doble de este número de elementos (puesto que están intercalados los números de estado), y que no es siquiera necesario almacenar en la pila los símbolos, de modo que el análisis podría hacerse sin ellos (ahora cada reducción elimina de la pila tantos elementos como longitud tenga el consecuente):

Pila	Entrada	Acción	
0	cdad$	s1 (desplazar)	
0 1	dad$	s2	
0 1 2	ad$	r2 (reducir a *S*, 1 símbolo);	ir-a[1,S]=4
0 1 4	ad$	s5	
0 1 4 5	d$	r4 (*A*, 1 símbolo);	ir-a[4,A]=6
0 1 4 6	d$	s10	
0 1 4 6 10	$	r1 (*S*, 4 símbolos);	ir-a[0,S]=3
0 3	$	accept	

Obsérvese también que la lista de reducciones realizadas muestra una derivación más a la derecha en orden inverso.

2.2. Yacc

Yacc[1] es una herramienta que toma una especificación de gramática y escribe un analizador sintáctico en *C* que reconoce sentencias válidas para dicha gramática.

2.2.1. Especificación de gramáticas para Yacc

Un programa fuenta para Yacc consta de tres secciones, como Lex, separadas por líneas `%%`:

... sección de definiciones ...
`%%`
... sección de reglas ...
`%%`
... sección de rutinas ...

La segunda sección es obligatoria.

Los símbolos auxiliares son cadenas de letras, dígitos, caracteres de subrayado y puntos, que no comiencen por un dígito, excluyendo la palabra reservada *error*, además de los *literales*: símbolos entre comillas simples. Se consideran auxiliares a los que aparecen como antecedentes de reglas.

La sección de definiciones puede contener

[1] "Yet Another Compiler Compiler": desarrollado por S.C. Johnson en los años 70. Estándar en las veriones de Unix. Variantes: AT&T, Berkeley, *bison* (GNU), MKS (para DOS y OS/2), Abraxas, ...

- un *bloque literal*, entre líneas `%{` y `%}`, que será copiado al principio del programa C generado y que contendrá normalmente líneas de declaración y líneas `#include`
- líneas de declaración **%union**, **%start**, **%token**, **%type**, **%left**, **%right** y **%nonassoc**
- comentarios en C (entre `/*` y `*/`)
- líneas de código C, precedidas por un espacio en blanco

La sección de reglas contiene la especificación de la gramática. Cada regla está constituída por un símbolo auxiliar, dos puntos (:) y una cadena (posiblemente vacía) de símbolos, componentes léxicos literales y acciones y un punto y coma (;) para finalizar. Cada regla comienza en una nueva línea.

Si varias reglas tienen el mismo antecedente, pueden resumirse sustituyendo el antecedente y los dos puntos por una barra vertical (|), pudiéndose entonces suprimir el punto y coma que finaliza la regla anterior.

Una acción es una sentencia compuesta en C que se ejecutará cuando el analizador alcance el punto de la gramática en al que está anotada la acción. Por ejemplo:

```
S  :  'a' S 'b' {printf ("Regla S->aSb\n");}
   |            {printf ("Regla S->lambda\n");
;
```

La de rutinas, código C que será también trascrito literalmente al programa C obtenido.

2.2.2. Uso básico

La orden para "compilar" con yacc es[2]

`yacc` [*opciones*] *fuente-yacc*

El programa fuente para yacc deberá tener la extensión `.y`. Se generará el programa en C `y.tab.c`, que contiene la función que realiza el análisis sintáctico, llamada `yyparse()`

Para compilar éste, hay que proporcionar las rutinas **main** y **yyerror**, bien usando la biblioteca `liby.a`, mediante

```
cc y.tab.c -ly
```

o escribiéndolas directamente en la fuente para yacc, por ejemplo:

```
%{
#include <stdio.h>
yyerror (char *s) {
    fprintf (stderr, "%s\n", s);
}
%}
%%
S   : 'a' S 'b'
    |
    ;
%%
main() {
   yyparse();
}
```

Si la compilación se hace con la opción **-v**, yacc generará, además, un fichero `y.output` que contiene una descripción textual de las tablas, así como un resumen del proceso, en cuanto a número de estados, terminales, auxiliares y reglas procesados.

[2] Para distinguir entre "AT&T yacc" y "Berkeley yacc", emitir la orden **yacc** sin argumentos. Si muestra un mensaje de error, es At&T. Si muestra un resumen de la sintaxis, es Berkeley