El generador de analizadores sintácticos *Yacc*
II

Teoría de autómatas y lenguajes formales
Alma María Pisabarro Marrón (alma@infor.uva.es)
Universidad de Valladolid

1. Interacción con un analizador léxico

Los componentes léxicos ("token") de un programa fuente en yacc pueden ser

- caracteres directamente ('h', '+' ...)
- especificados en una línea %token

En cualquier caso deben estar proporcionados por una función de nombre yylex() que devuelve un tipo entero.

Ejemplo:

```
%{
#include <stdio.h>
yyerror (char *s) {
      fprintf (stderr, "%s\n", s) ;
}
%}

%%
S     : 'H' 'o' 'l'  'a' '\n' ;
%%
yylex() { /* código de yylex proporcionado directamente */
      return (getchar()) ; /* devuelve el propio código (ASCII) del
carácter */
}

main() {
      yyparse();
}
```

Al ejecutar el código obtenido de la compilación de esta especificación Yacc:

- con una entrada como 'Hola' seguido de "newline" y "EOF" no habrá salida (no se ha especificado acción)
- con cualquier otra entrada dará un mensaje de error (*syntax error*)

En este caso, no se han definido explícitamente los componentes léxicos, de forma que su valor es el propio código del carácter en cuestión.

2. Compilación

El analizador léxico debe conocer los tokens, este conocimiento puede establecerse de formas diferente.

1. Incluyendo en el fuente Yacc, en la sección de definiciones, %{#include "lex.yy.c"%}
2. Generando un fichero y.tab.h que se incluye en el fuente Lex en la sección de definiciones, %{#include "y.tab.h"%}

Dependiendo de cual se elija la compilación de los programas fuente Yacc debe hacerse de manera distinta.

2.1. *Primer método de compilación*

El código de yylex() puede obtenerse usando Lex, en cuyo caso, una posibilidad es incluirlo mediante la directiva #include:

```
%token H, O, L, A
%{
#include <stdio.h>
#define H 257
#define O 258
#define L 259
#define A 260
#include "lex.yy.c"

yyerror (char *s) {
      fprintf (stderr, "%s\n", s) ;
}
%}

%%
S     : H O L A '\n' ;
%%
main() {
      yyparse();
}
```

Siendo lex.yy.c el resultado de la compilación, por ejemplo, de

```
%{
#define H 257
#define O 258
#define L 259
#define A 260
%}
%%
[ \t]+          ;
[Hh]  return H;
[Oo]  return O;
[Ll]  return L;
[Aa]  return A;
.     |
\n    return yytext[0];
```

(Obsérvese que ahora tanto lex como yacc acceden a la misma definición de las constantes H, O, L y A).

En este caso, una vez obtenidos lex.yy.c y y.tab.c hace una compilación

cc y.tab.c -ly -ll

(NOTA: es importante el orden -ly -ll en el caso de necesitarse ambas cosas)

2.2. *Segundo método de compilación*

La línea %token proporciona una manera de evitar las líneas de definición de constantes ("defines"). Si yacc encuentra esta línea, genera las líneas de definición correspondientes numerando a partir de 256, y si en la compilación yacc se usa la opción -d, lo hace en un fichero de nombre y.tab.c Por ejemplo, con el código yacc:

```
%{
#include <stdio.h>
yyerror (char *s) {
      fprintf (stderr, "%s\n", s) ;
}
%}
%token H, O, L, A
%%
S     : H O L A '\n' ;
%%
main() {
      yyparse();
}
```

la compilación

yacc -d fichero.y

generará un y.tab.c y un y.tab.h (compruébese y obsérvese el contenido). Si ahora se tiene un fichero lex :

```
%{
#include "y.tab.h"
%}
%%
[ \t]+          ;
[Hh]  return H;
[Oo]  return O;
[Ll]  return L;
[Aa]  return A;
.     |
\n    return yytext[0];
```

La definición de constantes estará ya incluida, y se compilará con

lex fichero.l

Finalmente se combinan los resultados con

cc y.tab.c lex.yy.c -ly -ll

3. Tablas de análisis

Yacc puede obtener explícitamente las tablas de análisis por desplazamiento-reducción si se usa la opción -v En este caso se genera (además del y.tab.c) un fichero textual y.output con dichas tablas.

En el caso del ejemplo final del documento anterior (Yacc I), el fichero que se obtiene es

```
          0  $accept : hola $end
          1  hola    : H letra_o L A
          2  letra_o : O
          3          | letra_o O

state 0
          $accept : . hola $end  (0)
          H  shift 1
          .  error
          hola  goto 2
state 1
          hola : H . letra_o L A  (1)
          O  shift 3
          .  error
          letra_o  goto 4
```

```
state 2
          $accept : hola . $end  (0)
          $end  accept
state 3
          letra_o : O .  (2)
          .  reduce 2
state 4
          hola : H letra_o . L A  (1)
          letra_o : letra_o . O  (3)
          O  shift 5
          L  shift 6
          .  error
state 5
          letra_o : letra_o O .  (3)
          .  reduce 3
state 6
          hola : H letra_o L . A  (1)
          A  shift 7
          .  error
state 7
          hola : H letra_o L A .  (1)
          .  reduce 1

6 terminals, 3 nonterminals
4 grammar rules, 8 states
```

Corresponde a la tabla siguiente:

	ACCIÓN					IR A	
ESTADO	H	O	L	A	$end	hola	letra_o
0 $accept : . hola $end	s1					2	
1 hola : H . letra_o L A		s3					4
2 $accept : hola . $end					Accept		
3 letra_o : O.	r2	r2	r2	r2	r2		
4 hola : H letra_o . L A letra_o : letra_o . O		s4	s6				
5 letra_o : letra_o O.	r3	r3	r3	r3	r3		
6 hola : H letra_o L . A				s7			
7 hola : H letra_o L A.	r1	r1	r1	r1	r1		