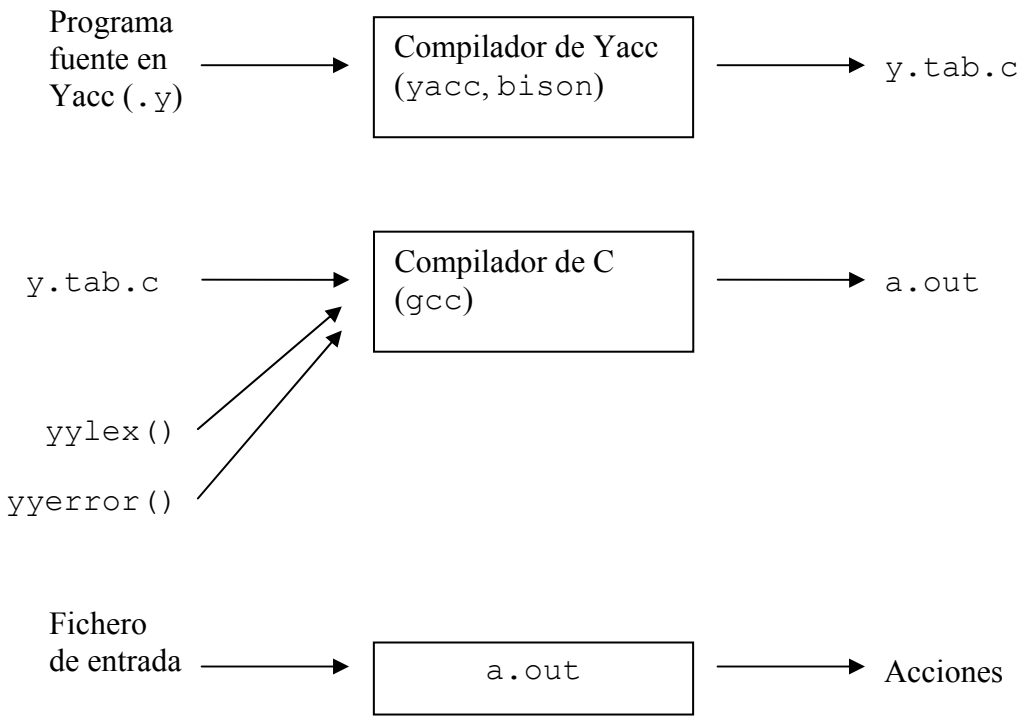
El generador de analizadores sintácticos *yacc*.

Teoría de Autómatas y lenguajes formales
Federico Simmross Wattenberg (fedesim@infor.uva.es)
Universidad de Valladolid

Hemos visto cómo el análisis léxico facilita la tarea de reconocer los elementos de un lenguaje uno a uno. A partir de ahora, vamos a centrarnos en el análisis sintáctico, que nos permitirá averiguar si un fichero de entrada cualquiera respeta las reglas de una gramática concreta. Para el tema del análisis sintáctico vamos a utilizar la herramienta *yacc* (Yet Another Compiler Compiler).

1. Funcionamiento de *yacc*

Igual que sucedía con lex, yacc no es directamente un analizador sino un generador de analizadores. A partir de un fichero fuente en yacc, se genera un fichero fuente en C que contiene el analizador sintáctico. Sin embargo, un analizador sintáctico de yacc no puede funcionar por sí solo, sino que necesita un analizador léxico externo para funcionar. Dicho de otra manera, el fuente en C que genera yacc contiene llamadas a una función `yylex()` que debe estar definida y debe devolver el tipo de lexema encontrado. Además, es necesario incorporar también una función `yyerror()`, que será invocada cuando el analizador sintáctico encuentre un símbolo que no encaja en la gramática.



2. El lenguaje Yacc

2.1. Esquema general

Un programa fuente de Yacc se parece bastante a uno de lex. La diferencia principal está en la sección de reglas, que en vez de expresiones regulares contiene las reglas de la gramática:

```
<sección de definiciones>
%%
<sección de reglas>
%%
<sección de rutinas>
```

- De estas tres secciones, sólo la segunda es obligatoria, y no debe estar vacía (nótese que en lex, las tres secciones pueden estar vacías). Esto quiere decir que el mínimo programa en yacc es:

  %%
  regla gramatical acción en C

- La sección de declaraciones puede incluir varias cosas, tal y como ocurría en lex, pero ahora su función principal no es definir expresiones regulares, sino declarar los símbolos terminales de la gramática mediante la directriz `%token`. Todo lo que no sea un terminal, será considerado un símbolo no terminal, y por tanto debe haber un a regla para él:

  ```
  %token            IF,ELSE,LLAVE_AB,LLAVE_CE,IDENT
  ```

- La sección de reglas contiene la gramática en sí. *Componentes* es una combinación de terminales y no terminales que describe al no terminal de la izquierda de la regla:

  ```
  no_terminal:      componentes       {acciones en C}
  ```

- La sección de rutinas tiene la misma función que la de lex, pero yacc (dependiendo de su variante) no define por defecto las funciones `main()`, `yylex()` e `yyerror()`, así que hay que incluirlas aquí, o bien en otro fichero que se enlazará en la fase final de la compilación.

Yacc genera una función llamada `yyparse()` que contiene el analizador sintáctico. Esta función se comporta como una máquina de estados cuya misión es intentar reducir todo el fichero de entrada al símbolo inicial de la gramática (el primero que se haya definido). Si yacc lo consigue, el analizador sintáctico volverá sin error, y en caso contrario, se invocará a la función `yyerror()`, que debe estar definida también en algún sitio.

2.2. Sección de reglas

Una regla de yacc es parecida a una de lex, pero en vez de un patrón regular, especifica una regla de la gramática. Las reglas deben estar dadas de forma que la parte izquierda conste de un único símbolo no terminal, y la parte derecha indique la combinación de terminales y no terminales de que puede estar compuesto. Además, toda regla debe incluir una acción en C que se ejecutará en cuanto yacc consiga encontrar los componentes del símbolo resultado:

```
símbolo_result:     componentes  acción_en_C
```

- El símbolo resultado debe estar situado en la primera posición de la línea; es decir, que no puede haber espacios antes del símbolo resultado.

- Los componentes son una combinación de terminales y no terminales separados por espacios en blanco.

- La acción puede ser una sola sentencia de C, o una sentencia compuesta, encerrada entre llaves {}.

2.3. Código C en un programa lex

Igual que en lex, muchas veces es útil incluir código en C dentro de un programa en yacc. Es normal utilizar esta posibilidad para incluir las funciones `main()` e `yyerror()` en el mismo fichero. La función `yylex()`, sin embargo, suele proporcionarse en un módulo aparte, generado por lex, que se enlaza en la fase final de la compilación.

Normalmente, el código C se introduce en la sección de declaraciones, o bien en la de rutinas:

- En la sección de declaraciones, entre una línea `%{` y una línea `%}` (nótese que no es `}%`). Este código será externo y se situará antes de la función `yyparse()` en el fichero `y.tab.c`.

- En la sección de rutinas todo lo que haya debe ser código C, que será externo y posterior a la función `yyparse()`.

3. Ejemplo

- Construir un analizador sintáctico que reconozca la palabra ‘Hola’, en mayúsculas o minúsculas, y con un número arbitrario de letras ‘o’:

```
%token H,O,L,A

%{

#define H 257
#define O 258
#define L 259
#define A 260

int yylex() {
    char carac=getchar();

    switch(carac) {
case 'H':
case 'h':
    return H;

case 'O':
case 'o':
    return O;

case 'L':
case 'l':
    return L;

case 'A':
case 'a':
    return A;
    }

    return -1;
}

int yyerror() {
    printf("La expresión de la entrada NO forma parte del lenguaje reconocido\n");
    return 0;
}

%}


%%

hola:          H letra_o L A  {printf("La expresión introducida es parte del lenguaje reconocido\n");}

letra_o:       O |
               letra_o O      ;

%%

int main() {
    yyparse();

    return 0;
}
```