Índice

Marina de la Cruz
Alfonso Ortega

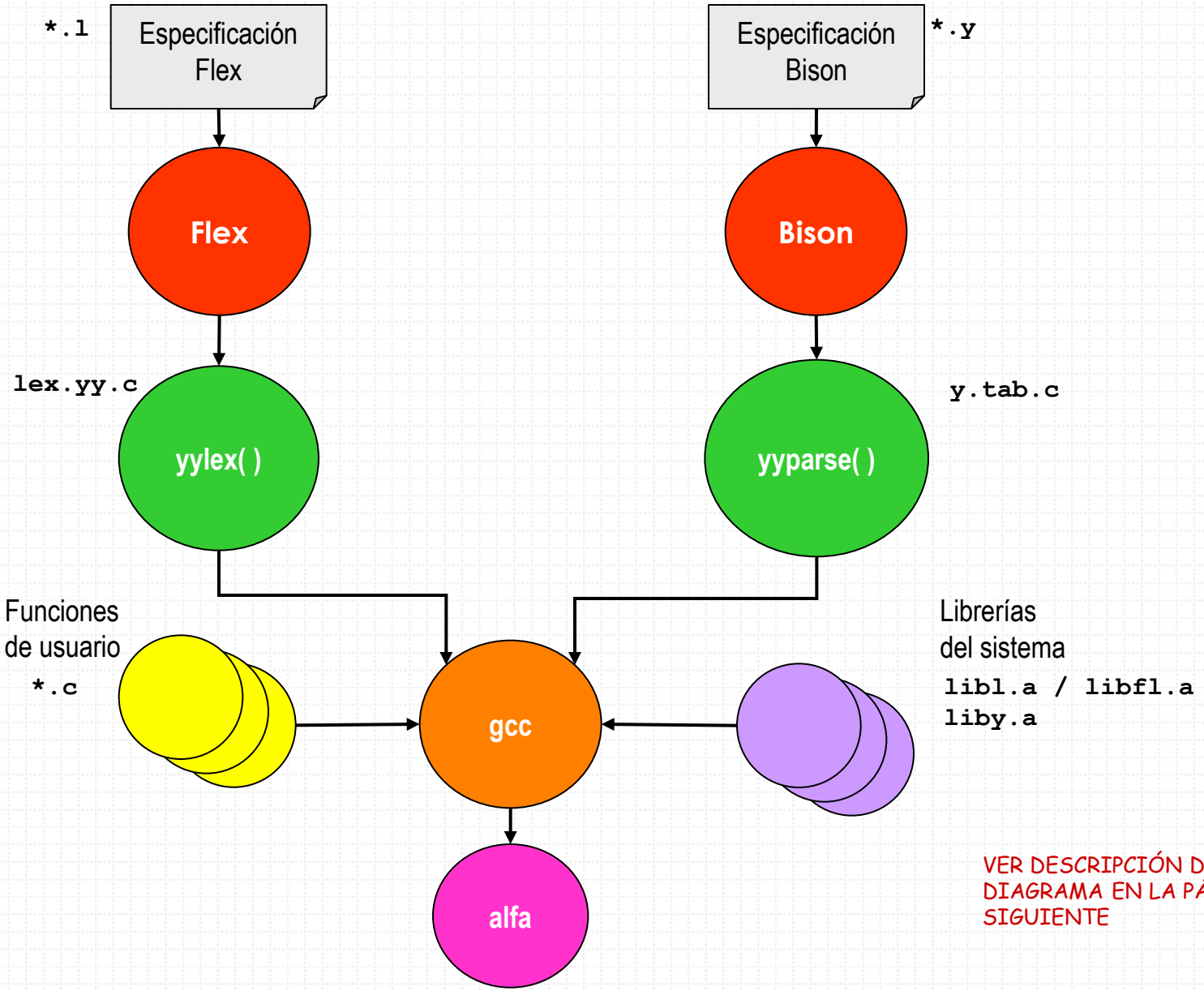
- Introducción
- Uso de Bison con Flex
  - Construcción del programa objetivo "alfa"
  - Comunicación entre las funciones main(), yylex() e yyparse()
- Formato del fichero de especificación de Bison
  - Estructura del fichero
  - Sección de definiciones Bison
  - Sección de reglas
  - Sección de funciones de usuario

Introducción

- Bison es un generador de analizadores sintácticos LALR(1).

- Habitualmente Bison se utiliza junto con Flex.
  - Flex genera un analizador morfológico/léxico: **yylex()**
  - Bison genera un analizador sintáctico: **yyparse()**

Construcción del programa objetivo **alfa** (I)

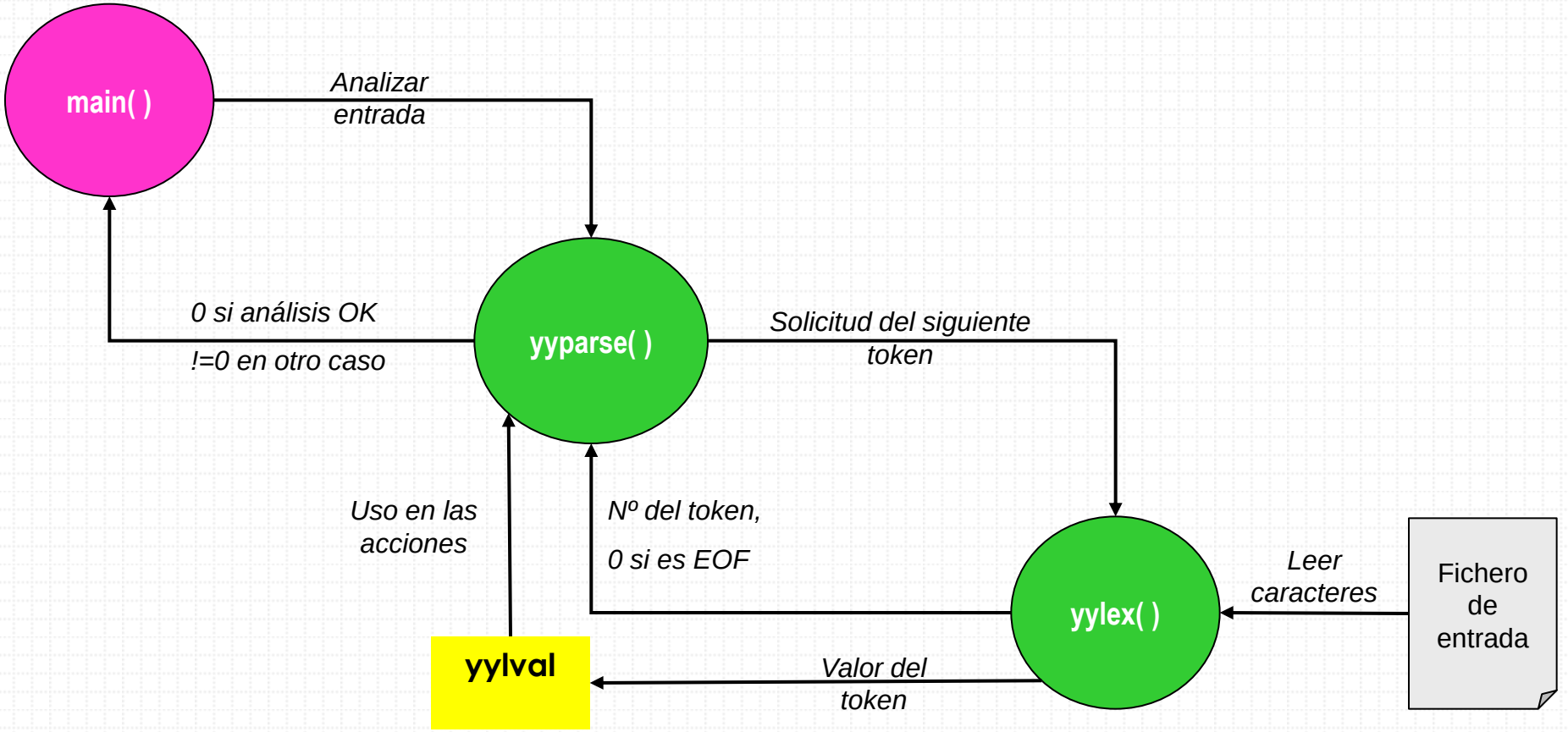


VER DESCRIPCIÓN DEL DIAGRAMA EN LA PÁGINA SIGUIENTE

Construcción del programa objetivo **alfa** (II)

- Flex construye la función **yylex()** de análisis morfológico a partir del fichero de especificación correspondiente. El fichero donde se ubica la función **yylex()** se llama **lex.yy.c**.
- Bison construye la función **yyparse()** de análisis sintáctico a partir del fichero de especificación correspondiente. La función **yyparse()** se ubica en el fichero **y.tab.c.**
- Las funciones de usuario incluyen funciones de soporte como por ejemplo funciones para generar código, funciones invocadas por Flex y Bison como por ejemplo **yywrap()** o **yyerror()**, y también la función principal que invoca al analizador sintáctico para realizar la compilación. Todas estás funciones se pueden organizar en varios ficheros.
- Las librerías del sistema proporcionan versiones sencillas de las funciones que son invocadas por Flex y Bison así como una versión mínima de la función principal que invoca al analizador léxico o al sintáctico. Si el usuario proporciona estas funciones, no es necesario utilizar las librerías. Dependiendo del sistema operativo, las librerías existen o no, por lo tanto para facilitar la portabilidad del código, es recomendable que el usuario aporte sus propias versiones de las funciones, o bien establezca de manera correcta las opciones de las herramientas Flex y Bison.
- Para obtener el programa objetivo **alfa**, se compilan y enlazan los ficheros descritos anteriormente.

Comunicación entre las funciones **main()**, **yylex()** e **yyparse()** (I)



VER DESCRIPCIÓN DEL DIAGRAMA EN LA PÁGINA SIGUIENTE

Comunicación entre las funciones **main()**, **yylex()** e **yyparse()** (II)

- La función **main()** invoca a la función de análisis sintáctico **yyparse()** que devuelve un 0 si termina el análisis con éxito, es decir, si la entrada es sintácticamente correcta. En caso contrario devuelve un valor distinto de 0.

- La función **yyparse()** solicita a la función **yylex()** los tokens de la entrada y comprueba si forman una construcción válida de acuerdo a las reglas de la gramática descritas en el fichero de especificación Bison correspondiente. Cuando detecta un error sintáctico invoca a la función **yyerror()** y termina el proceso de análisis devolviendo un valor distinto de 0.

- La función **yylex()** lee el fichero de entrada para identificar los tokens descritos en el fichero de especificación correspondiente. Cada vez que la función **yylex()** devuelve un token al analizador sintáctico, si el token tiene un valor asociado, **yylex()** guarda ese valor en la variable **yylval** antes de terminar. Por ejemplo, un identificador de una variable, además de tener un número de token que lo identifica y distingue de otros tipos de tokens, también tiene asociado como valor o atributo, el lexema del identificador. Sin embargo, puede considerarse que un paréntesis no tiene asociado ningún valor o atributo. La función **yyparse()** utiliza el valor de la variable **yylval** en las acciones de las reglas de la gramática.

Formato del fichero de especificación de Bison (I)

Estructura del fichero

- Un fichero de especificación de Bison tiene tres secciones separadas por líneas que contienen el separador %% (es muy similar al fichero de entrada de Flex).

sección de definiciones

```
%{
    /* delimitadores de código C */
%}


%%
```

sección de reglas

```
%%
```

sección de funciones de usuario

Formato del fichero de especificación de Bison (II)

Sección de definiciones Bison (I)

- En esta sección se puede:
  - incorporar bloques de código C que se copiarán literalmente en el fichero de salida
  - definir el tipo de la variable **yylval**
  - definir los símbolos (terminales y no terminales) de la gramática
  - definir el axioma de la gramática
  - definir la asociatividad y precedencia de los operadores

- Bloque de código C

  Este bloque contiene:
  - definiciones de macros
  - declaraciones de variables
  - declaraciones de funciones
  - directivas #include

    ... que se van a utilizar en las acciones de las reglas gramaticales.
  - El contenido de esta sección se copia literalmente al principio del fichero **y.tab.c** que genera Bison.

Sección de definiciones Bison (II)

- Declaración `%union`
  - Por defecto, la variable **yylval** mediante la cual Flex le pasa a Bison los valores semánticos de los tokens es de tipo int. Pero habitualmente, los valores semánticos de los tokens son de distintos tipos. Por ejemplo, un token de tipo identificador tiene un valor semántico de tipo char*, mientras que un token de tipo constante numérica entera tiene un valor semántico de tipo int. Con la declaración %union se define indirectamente una estructura union de C con un campo para cada tipo de valor semántico.

  - Por ejemplo, si los tipos de valores semánticos son int y char* se utiliza la siguiente declaración :

```
%union
    {
     char* cadena;
     int numero;
    }
```

Sección de definiciones Bison (III)

- Declaración `%token` (i)
  - Se utiliza para definir los símbolos terminales (tokens) de la gramática.
  - La forma más sencilla es:

```
%token NOMBRE_TOKEN
```

  - Una forma más completa es:

```
%token <campo_union> NOMBRE_TOKEN
```

  - Ejemplo:

```
%token SI
%token ENTONCES
...
%token <cadena> ID
%token <numero> NUM
```

  - Desde FLex, los tokens cualificados se devolverán haciendo, por ejemplo:

```
[A-Z]+    { strcpy(yylval.cadena,yytext); return ID; }
[0-9]+    { yylval.numero = atoi(yytext); return NUM; }
```

Sección de definiciones Bison (IV)

- Declaración `%token` (ii)
  - Se pueden agrupar varios tokens en una línea si tienen el mismo tipo.
  - Los tokens de un único carácter no es necesario declararlos, porque están declarados implícitamente en Bison con su correspondiente valor ASCII.
  - Desde Flex, este tipo de tokens se devuelven haciendo, por ejemplo:

```
"+"     {return '+'; } o bien {return yytext[0];}
"("     {return '('; } o bien {return yytext[0];}
";"     {return ';'; } o bien {return yytext[0];}
```

  - La compilación del fichero de especificación *.y se realiza con la opción **-d** para que Bison genere el fichero **y.tab.h** que contiene las definiciones de los tokens. Posteriormente, este fichero se incluye en la especificación de Flex con el objetivo de que Bison y Flex compartan las definiciones de tokens.

Sección de definiciones Bison (V)

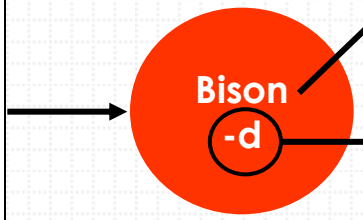
- Ejemplo del fichero **y.tab.h**

ejemplo.y

```
%{
.....
%}
%union
{
  char* cadena;
  int numero;
}

%token SI
%token ENTONCES
%token SINO
%token <cadena> ID
%token <numero> NUM
.....

%%
.....
%%
.....
```



y.tab.c

```
.....
.....
```

y.tab.h

```
typedef union
{
  char* cadena;
  int numero;
}YYSTYPE;
extern YYSTYPE yylval;

#define SI 257
#define ENTONCES 258
#define SINO 259
#define ID 260
#define NUM 261
.....
```

ejemplo.l

```
#include "y.tab.h"
.....
```

Sección de definiciones Bison (VI)

- Declaración `%type`
  - Se utiliza cuando se ha hecho la declaración %union para especificar múltiples tipos de valores.
  - Permite declarar el tipo de los símbolos **no terminales**. Los no terminales a los que no se les asigna ningún valor a través de `$$` no es necesario declararlos (ver sección de reglas de la gramática)
  - La declaración es de la forma:

```
%type <campo_union> nombre_no_terminal
```

  - Se pueden agrupar varios no terminales en una línea si tienen el mismo tipo.

- Declaración `%start`
  - Declara el axioma de la gramática.
  - Si se omite la declaración, se asume que el axioma de la gramática es el primer no terminal de la sección de reglas.
  - La declaración es de la forma

```
%start axioma
```

Sección de definiciones Bison (VII)

✖ Precedencia de operadores

Cuando una expresión contiene varios operadores, la precedencia de los operadores determina el orden de evaluación de los operadores individuales. Por ejemplo, la expresión x + y / z se evalúa como x+(y/z).

✖ Asociatividad de operadores

Cuando un operando se encuentra entre dos operadores con igual grado de prioridad, la asociatividad de los operadores determina el orden en que se ejecutan las operaciones. Por ejemplo la expresión x *y /z se evalúa como (x*y)/z.

Sección de definiciones Bison (VIII)

- Declaraciones `%left` y `%right`
  - Permiten declarar la asociatividad de los operadores de la gramática.
  - La declaración **%left** especifica asociatividad por la izquierda.
  - La declaración **%right** especifica asociatividad por la derecha.
  - La precedencia de los operadores queda establecida por el orden de aparición de las declaraciones de asociatividad en el fichero de especificación, siendo de menor precedencia el operador que antes se declara.
  - Por ejemplo, las declaraciones:

```
%left '+' '-'
%left '*' '/'
```

    establecen que los cuatro operadores son asociativos por la izquierda, que '+' y '-' son de la misma precedencia, pero de menor que '*' y '/' (estos dos operadores tienen la misma precedencia).

Sección de reglas (I)

- Es la sección "más importante" del fichero de especificación.
- Contiene las reglas de la gramática escritas en un formato concreto y opcionalmente con acciones asociadas a las reglas.
- Formato de las reglas:
  - `simboloNoTerminal: simb1 simb2 … simbM [acción]`
    `;`
  - si varias reglas tienen la misma parte izquierda se pueden agrupar:

```
simboloNoTerminal:   parteDerecha1 [acción1]
                   | parteDerecha2 [acción2]
                                ……
                   ;
```

  - por claridad, se comentan las reglas lambda:

```
simboloNoTerminal:   /* vacío */     [acción1]
                   | parteDerecha2 [acción2]
                   ;
```

Sección de reglas (II)

- Las acciones de las reglas:
  - las acciones son un conjunto de instrucciones C encerradas entre llaves, y que se ejecutan cada vez que se reconoce una instancia de la regla.
  - Normalmente se sitúan al final de la regla, aunque también se admiten en otras posiciones de la regla.
  - La mayoría de las acciones trabajan con los valores semánticos de los símbolos de la parte derecha, accediendo a ellos mediante pseudo-variables del tipo `$N`, donde `N` representa la posición del símbolo. El valor semántico del símbolo no terminal de la parte izquierda de la regla se referencia con `$$`.
    El tipo del valor semántico de un símbolo es el que se le haya asociado mediante una declaración `%token` (terminales) o `%type` (no terminales).
  - La acción por defecto es:

```
$$ = $1
```

  - Ejemplo:

```
exp:      ...
        | exp '+' exp {$$ = $1 + $3}
        ;
```

Sección de funciones de usuario

- En esta sección se ubican funciones de soporte:
  - Funciones diseñadas por el usuario para ser utilizadas en las sección de reglas.
  - Función **yyerror()**
    - Cuando **yyparse()** detecta un error sintáctico invoca a la función **yyerror().** Esta función tiene que ser proporcionada por el usuario, y se puede incorporar en esta sección del fichero de especificación (en Linux se puede no proporcionar ya que la proporciona la librería de Bison).
    - El prototipo de la función puede ser:

```
void yyerror(char* s)
```

o bien

```
int yyerror(char* s)
```

    - El único parámetro de la función es el mensaje del error sintáctico ocurrido.
    - La versión mínima de esta función muestra por pantalla el mensaje que recibe como parámetro.
- El contenido de esta sección se copia literalmente en el fichero de salida.
- En aplicaciones grandes, es más conveniente agrupar todas las funciones de soporte en un fichero o conjunto de ficheros, en lugar de incluirlas en esta sección.