Lenguajes y Gramáticas independientes del Contexto

Para que Bison analice un lenguaje, este debe ser descrito por una **gramática independiente del contexto**. Esto quiere decir que debe especificar uno o más **grupos sintácticos** y dar reglas para contruirlos desde sus partes. Por ejemplo, en el lenguaje C, un tipo de agrupación son las llamadas `expresiones'. Una regla para hacer una expresión sería, "Una expresión puede estar compuesta de un signo menos y otra expresión". Otra regla sería, "Una expresión puede ser un entero". Como puede ver, las reglas son a menudo recursivas, pero debe haber al menos una regla que lleve fuera la recursión.

El sistema formal más común de presentar tales reglas para ser leidas por los humanos es la **Forma de Backus-Naur** o "BNF", que fue desarrollada para especificar el lenguaje Algol 60. Cualquier gramática expresada en BNF es una gramática independiente del contexto. La entrada de Bison es en esencia una BNF legible por la máquina.

No todos los lenguajes independientes del contexto pueden ser manejados por Bison, únicamente aquellos que sean LALR(1). Brevemente, esto quiere decir que debe ser posible decir cómo analizar cualquier porción de una cadena de entrada con un solo token de preanálisis. Hablando estrictamente, esto es una descripción de una gramática LR(1), y la LALR(1) implica restricciones adicionales que son difíciles de explicar de manera sencilla; pero es raro en la práctica real que se encuentre una gramática LR(1) que no sea LALR(1). See section Conflictos Misteriosos de Reducción/Reducción, para más información a cerca de esto.

En las reglas gramaticales formales para un lenguaje, cada tipo de unidad sintáctica o agrupación se identifica por un **símbolo**. Aquellos que son construidos agrupando construcciones más pequeñas de acuerdo a reglas gramaticales se denominan **símbolos no terminales**; aquellos que no pueden subdividirse se denominan **símbolos terminales** o **tipos de tokens**. Denominamos **token** a un fragmento de la entrada que corresponde a un solo símbolo terminal, y **grupo** a un fragmento que corresponde a un solo símbolo no terminal.

Podemos utilizar el lenguaje C como ejemplo de qué significan los símbolos, terminales y no terminales. Los tokens de C son los identificadores, constantes (numéricas y cadenas de caracteres), y las diversas palabras reservadas, operadores aritméticos y marcas de puntuación. Luego los símbolos terminales de una gramática para C incluyen `identificador', `número', `cadena de caracteres', más un símbolo para cada palabra reservada, operador o marca de puntuación: `if', `return', `const', `static', `int', `char', `signo-más', `llave-abrir', `llave-cerrar', `coma' y muchos más. (Estos tokens se pueden subdividir en caracteres, pero eso es una cuestión léxica, no gramatical.)

Aquí hay una función simple en C subdividida en tokens:

```
int             /* palabra reservada `int' */
cuadrado (x)    /* identificador, paréntesis-abrir */
                /* identificador, paréntesis-cerrar */
     int x;     /* palabra reservada `int', identificador, punto y coma */
{               /* llave-abrir */
  return x * x; /* palabra reservada `return', identificador, */

         /* asterisco, identificador, punto y coma */
}               /* llave-cerrar */
```

Las agrupaciones sintácticas de C incluyen a las expresiones, las sentencias, las declaraciones, y las definiciones de funciones. Estas se representan en la gramática de C por los símbolos no terminales `expresión', `sentencia', `declaración' y `definición de función'. La gramática completa utiliza docenas

de construcciones del lenguaje adicionales, cada uno con su propio símbolo no terminal, de manera que exprese el significado de esos cuatro. El ejemplo anterios es la definición de una función; contiene una declaración, y una sentencia. En la sentencia, cada `x' es una expresión y también lo es `x * x'.

Cada símbolo no terminal debe poseer reglas gramaticales mostrando cómo está compuesto de construcciones más simples. Por ejemplo, un tipo de sentencia en C es la sentencia `return`; esta sería descrita con una regla gramatical que interpretada informalmente sería así:

> Una `sentencia' puede estar compuesta de una parabra clave `return', una `expresión' y un `punto y coma'.

Aquí existirían muchas otras reglas para `sentencia', una para cada tipo de sentencia en C.

Se debe distinguir un símbolo no terminal como el símbolo especial que define una declaración completa en el lenguaje. Este se denomina **símbolo de arranque**. En un compilador, este representa un programa completo. En el lenguaje C, el símbolo no terminal `secuencia de definiciones y declaraciones' juega este papel.

Por ejemplo, `1 + 2' es una expresión válida en C--una parte válida de un programa en C--pero no es válida como un programa en C *completo*. En la gramática independiente del contexto de C, esto se refleja en el hecho de que `expresión' no es el símbolo de arranque.

El analizador de Bison lee una secuencia de tokens como entrada, y agrupa los tokens utilizando las reglas gramaticales. Si la entrada es válida, el resultado final es que la secuencia de tokens entera se reduce a una sola agrupación cuyo símbolo es el símbolo de arranque de la gramática. Si usamos una gramática para C, la entrada completa debe ser una `secuencia de definiciones y declaraciones'. Si no, el analizador informa de un error de sintaxis.

Fuente de información: http://es.tldp.org/Manuales-LuCAS/BISON/bison-es-1.27.html#SEC13