Bison II: Atributos y acciones semánticas

Lenguajes de Programación y Procesadores de Lenguajes
Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Curso 2020-21

Índice

1. Presentación

En este práctica aprenderás a incorporar acciones semánticas en la especificación Bison, definir los tipos de los atributos y asociar atributos sintetizados a los símbolos de la gramática Bison.

Para realizar este trabajo se recomienda, además de estudiar detenidamente este documento, consultar el manual de Bison.

2. Esquemas de traducción dirigidos por la sintaxis (ETDS) en Bison

En un programa Bison se pueden asociar a las reglas gramaticales acciones semánticas escritas en el lenguaje C. Cada vez que se aplique una regla gramatical se ejecutarán las acciones semánticas que tiene asociadas. El orden de ejecución de las acciones vendrá determinado por el análisis sintáctico y la posición que ocupan estas acciones en el lado derecho de las reglas.

En la mayoría de programas se necesitarán diferentes tipos de atributos. Por esta razón es necesario definir el tipo de dato usado por cada uno de los atributos. Para especificar el tipo de dato asociado a cada símbolo de la gramática se deben seguir los siguientes pasos:

1. Especificar la colección completa de tipos de datos (atributos) posibles mediante la declaración de Bison %union.

   Así por ejemplo se puede definir una unión como:

   ```
   %union {
        int cent ;       /* Valor de la cte numerica entera para el terminal "cte" */
        char *ident ;    /* Nombre del identificador */
   }
   ```

   Tras esta declaración cualquier símbolo terminal o no-terminal podrá tener asociado un valor entero (*cent*), o un puntero a un carácter (∗*ident*).

2. Elegir un tipo para cada símbolo (terminal o no-terminal) que necesite un valor semántico. Para los terminales esto se hace con la declaración de Bison *%token* y para los no-terminales con la declaración %type.

   Si por ejemplo se desea que el símbolo terminal $CTE\_$ tenga asociado un valor entero (campo *cent* de la unión definida), deberá escribirse en Bison:

   ```
   %token <cent> CTE_
   ```

   De igual forma, si se desea que el símbolo no-terminal *exp* tenga asociado un valor entero (campo *cent*), será necesario indicarlo mediante *%type*:

   ```
   %type <cent> exp
   ```

   Si algún símbolo gramatical necesita más de un valor semántico (más de un atributo), será necesario declarar un tipo de dato estructura (*struct*) cuyos miembros (campos) sean de los tipos empleados por el símbolo.

Los valores semánticos de los no-terminales se asignarán en las acciones de las reglas Bison que los definen. Los de los terminales se asignan a la variable *yylval* en las reglas de Flex donde se define cada token. Así por ejemplo, podríamos encontrar en el fichero *Flex*.

```
{entero}      {ECHO ; yylval.cent= atoi(yytext); return(CTE_); }
```

Donde *cent* es un campo de la %union definida en la sección de declaraciones de Bison, y $CTE\_$ el nombre dado al símbolo léxico correspondiente a una constante entera.

Como puede observarse en el ejemplo anterior, Flex proporciona información sobre el token reconocido mediante varias variables globales, entre las que se pueden destacar *char* yytext*, que contiene el lexema analizado, y *int yyleng* que contiene la longitud del lexema. Mediante la función estándar de C *atoi* se convierte la secuencia de caracteres leída en un número entero.

3. ETDS con atributos sintetizados

Una acción semántica en el lado derecho de una producción consiste en instrucciones escritas en el lenguaje C encerradas entre llaves. Se pueden situar en cualquier posición dentro de la regla, aunque frecuentemente aparecen al final.

El código C en una acción puede usar o dar valor a los valores semánticos de los símbolos y acciones de la regla mediante la construcción $n, que hace referencia al valor semántico del símbolo[1] n-ésimo del lado derecho de la regla. El valor semántico (atributo sintetizado) para el no-terminal del lado izquierdo de la regla viene representado por $$[2]. Veamos un pequeño ejemplo:

```
fac: PARA_ exp PARC_ { $$ = $2 ; }
```

Esta regla devuelve como valor semántico del no-terminal fac (representado en la acción semántica por $$), el valor semántico del símbolo no-terminal *exp* (representado por $2 ya que se trata del segundo símbolo del lado derecho de la regla).

Si no se especifica una acción para una regla, Bison asume una por defecto: $$ = $1. De este modo el valor del primer símbolo del lado derecho se convierte en el valor del no-terminal del lado izquierdo. Esta acción por defecto solo es válida si concuerdan los dos tipos de datos. No hay una acción por defecto con significado para la regla vacía.

Si para la escritura de la acción semántica se necesita alguna variable local temporal, se puede declarar dentro de las llaves de la acción semántica, del mismo modo que se haría en el lenguaje C.

[1] o reglas semántica, como se verá a continuación

[2] Pero solo cuando aparece en la última acción del lado derecho.

4. Acciones a mitad de una regla

Frecuentemente se necesita poner una acción semántica entre los símbolos de una regla. Estas acciones se escriben como las acciones al final de la regla, pero se ejecutan antes de que el analizador llegue a reconocer los componentes que aparecen a su derecha. Una acción en mitad de una regla puede hacer referencia a los símbolos (o acciones semánticas) que aparecen a su izquierda utilizando $n, pero no puede hacer referencia a los símbolos que aparecen a su derecha porque ésta se ejecuta antes de que sean analizados (solo se permiten gramáticas L-atribuidas).

Las acciones en mitad de una regla por sí mismas *cuentan* como uno de los símbolos de la regla. Esto significa que cuando se usa un atributo de un símbolo en una acción semántica, para saber el número que debe usarse con el $, es necesario contar, no solo los símbolos que aparecen a su izquierda, si no también las acciones semánticas que aparecen a su izquierda.

Cualquier acción en mitad de una regla puede tener también su propio valor semántico. Para asignar un valor a una acción semántica basta con asignar *dentro de la acción* un valor a $$. Las acciones que aparezcan a la derecha de ésta pueden hacer referencia a este valor utilizando $n (hay que recordar que a la hora de numerar los símbolos del lado derecho también hay que contar las acciones que haya entre ellos). Ya que no hay un símbolo que identifique a una acción semántica, no hay manera de declarar por adelantado un tipo de dato para su valor semántico. Por esta razón Bison ofrece la construcción $< *tipo* >, (donde *tipo* representa uno de los tipos definidos en la %union para asignar valor semántico a la acción semántica que lo incluye).

Por ejemplo, en las dos reglas siguientes, las acciones semánticas generarían el mismo resultado:

```
f:  exp MAS_ term  {$$ = $1+$3 ;}        /* Equivalente a la siguiente */
f:  exp {$<cent>$ = $1 ;}  MAS_ term  {$$ = $<cent>2+$4 ;}
```

Donde en la segunda producción:

- $< *cent* >$ hace referencia al valor semántico de la propia acción (que será del tipo < *cent* > que aparece en %union).
- $< *cent* >2 hace referencia al valor semántico de la acción a mitad de regla, el 2 corresponde a la posición que esta acción ocupa dentro de la parte derecha de la producción, contando símbolos y acciones semánticas.

Es importante destacar que no se puede asignar un valor al no-terminal del lado izquierdo en una acción que aparezca en medio de la regla, ya que *$$ representa en este caso el valor semántico de la propia acción*. La única forma de asignar un valor para el no-terminal del lado izquierdo es mediante una acción *al final* de la regla.

5. Ejemplo

A modo de ejemplo se muestra a continuación el código de la especificación Bison para un sencillo lenguaje que reconoce expresiones matemáticas con números enteros y que calcula y muestra por pantalla el valor de la expresión.

```
%{
#include <stdio.h>
#include "header.h"
%}

%union {
int   cent;                 /* Para el terminal "cte" entera */
}

%token PARA_ PARC_ MAS_ MENOS_ POR_ DIV_
%token <cent> CTE_
%type  <cent> exp term fac

%%

expMat : exp  { printf("\nValor de la expresion = %d\n",$1); }
       ;
exp : exp MAS_ term    { $$ = $1 + $3; }
    | exp MENOS_ term  { $$ = $1 - $3; }
    | term             { $$ = $1;      }
    ;
term  : term POR_ fac    { $$ = $1 * $3; }
      | term DIV_ fac    { $$ = $1 / $3; }
      | fac              { $$ = $1;      }
      ;
fac  : PARA_ exp PARC_  { $$ = $2;      }
     | CTE_             { $$ = $1;      }
     ;
%%
```