Linguagens Formais e Autómatos – LFAU

Aulas práticas
Constantino Martins

Introdução ao Bison:

Resumo:

Bison (equivalente ao yacc- Yet Another Compiler Compiler - mas no dóminio público) é um gerador de analisador sintáctico para linguagens LALR(1) - 1-Look-Ahead, Leftmost, Right-derivation.

A partir da descrição da gramática de uma linguagem constrói-se um programa **C** que efectua a análise sintáctica de frases da linguagem.

A descrição da gramática é geralmente feita num ficheiro com a extensão **.y**. Este ficheiro é constituído de três partes separadas por "%%":

```
%{
Declarações em C
%}
Declarações do Bison
%%
Regras
%%
Funções  C
```

Declarações:
As declarações são de dois tipo: as declarações C e as declarações ligadas à gramática.

As **declarações C** são colocadas entre %{ e %} como no flex.

Exemplo:

```
%{
#include <stdio.h>
extern int nlinhas;
```

%}

Nas **declarações ligadas à gramática (declarações do Bison)** vem **a definição dos símbolos não terminais e terminais (token)**. Não será preciso a declaração de todos os símbolos porque o bison fornece declarações por defeito.

Os símbolos não terminais são representados em letras minúsculas (expressao, termo,factor ....) e os tokens costumam estar representados com letras "grandes" (IF,WHILE..).

Símbolos terminais podem estar representados por (+,-,;,*,(,etc....). Do ponto de vista do analisador sintáctico, os tokens são constantes inteiras. Aos tokens corresponde o código ASCII. O valor das outras constante é fixado pelo bison.

O tokens representados por um carácter não precisam de ser declarados.

Tipo Variáveis:

O tipo dos **valores semânticos** é definido através da macro **YYSTYPE**.

Exemplo:

```
%{
#define YYSTYPE double
%}
```

%token

Declaração para os símbolos terminais (sem precedência ou associatividade especificada).

Exemplo:

```
%token NUM
```

%union

Declaração dos tipos que os valores semânticos possam ter. O parser fica a saber que vai ser usada uma union para registar valores semânticos. A cada símbolo gramatical pode ser atribuído um "tipo" que corresponde a um dos campos da union.

Exemplo:

```
%union {    /* define stack type */
double val;
symrec *tptr;
}
%token <val> NUM      /* define token NUM and its type */
```

%type

Declaração do tipo de valores semânticos para os símbolos não terminais

Exemplo:

```
%type <type> nonterminal
```

Nota: O aluno deve consultar o manual para ver outros tipos de declarações.

Regras:

Nesta secção são definidas as regras da gramática. As regras são compostas por um símbolo **não terminal**, o símbolo **':'** seguido de uma **sequência de símbolos e acções terminadas pelo ';'**.

Exemplos:

```
regra:   regra1-coponentes...
       | regra2-componentes...
       ...
;
```

Acções podem ser introduzidas nas regras entre **chavetas**. As acções são instruções em linguagem C.

Várias regras podem serem agrupadas usando o símbolo '|' (=ou). As acções são geralmente usadas para associar valores semânticos aos símbolos da gramática.

Os valores semânticos são passados através de variáveis especiais cujo nome começa por '$'.

nota: a macro YYSTYPE define valores semânticos

ex: #define YYSTYPE double

As variáveis são usadas da seguinte maneira:

ex: **expr: expr '+' term {$$=$1+$3;}**

$$ representa o valor associado ao símbolo à **esquerda da regra**. Os valores associados aos símbolos da parte direita são numerados de **$1 até $n**. Recorde que o **algoritmo de parsing do bison é bottom-up**, uma frase da linguagem é analisada a partir dos símbolos terminais, que são reduzidos, usando as regras da gramática, até obter o símbolo inicial.

Funções C:

Na última parte do ficheiro pode ser colocado código C que será copiado ao fim do ficheiro resultante.

Pode se definida aqui a função **yyparse()**.

A função **yyerror(char *)** também pode ser definida aqui. É chamada quando o parser encontra um erro de sintaxe.

Pode ser definida pelo utilizador, de maneira a escrever no ecrã o número da linha onde ocorreu o erro.

Comunicação flex/bison.

Por enquanto só iremos ver que a função de análise lexical (yylex()) devolve um valor (o token).

O valor semântico é passado através de uma variável global chamada **yylval**. Isto é, quando o flex é usado como analisador lexical, a atribuição da variável **yylval** é feita na acção de uma das regras do analisador lexical.

O valores semânticos de um token devem serem guardados na variável global **yylval**.

Se o valor for do tipo **int** basta escrever na parte do Flex:

```
yylval = value;  /* Put value onto Bison stack. */

return INT;      /* Return the type of the token. */
```

Se utilizar múltiplos tipos de variável, deve utilizar a declaração da **%union** para poder especificar qual é o tipo para yylval´s que irá usar.

Exemplo:

```
%union {
int intval;
double val;
symrec *tptr;
}
```

O código no flex pode ser assim:

```
yylval.intval = value; /* Put value onto Bison stack. */

return INT;         /* Return the type of the token. */
```

Depois iremos voltar sobre este assunto nas próximas aulas.

Comandos utilização do Bison

% which flex

/usr/ucb/flex

% which bison

/usr/ucb/bison

% bison -d -v fileparser.y

% flex filelexer.flex

% ls

lex.yy.c fileparser.tab.h fileparser.tab.c fileparser.output ...

% gcc -g -c lex.yy.c

% gcc -g -c fileparser.tab.c

% gcc -o executavelparser lex.yy.o fileparser.tab.o –lfl

% ./executavelparser < infile

Chamando o bison com a opção -d produz um ficheiro adicional fileparser.tab.h que pode ser incluído na secção de declarações do analisador lexical.

Chamando o bison com a opção -v produz um ficheiro adicional filepareser.output com indicação dos estados, das regras e dos conflitos (shift, reduce) caso existem.

O primeiro programa a ser implementado de ser o "hello world". No entanto a título de exemplo e para experimentar os comandos o aluno pode fazer, executar e explicar o seguinte exemplo:

Nota: O aluno deve modificar o exemplo nas partes do código que achar menos "correctas".

File Bison:

```
%union {
        double valreal;
        int valint;
}
%token <valint> INT
%token <valreal> REAL
%type <valreal> termo  final
%%
calculadora:/* vazio */
        | calculadora expressao '\n'
        ;
expressao: termo            {printf("resultado %f\n", $1);}
;
termo: final '+' termo {$$=$1+$3;}
        | final '-' termo {$$=$1-$3;}
        | final  {$$=$1;}
        ;
final: '(' termo ')'   {$$=$2;}
        | REAL              {$$=$1;}
        | INT   {$$=$1;}
        ;
%%
main(){
        yyparse();
}

int yyerror(char *s){
        printf("erro sintatico/semantico: %s\n",s);
}
```

File Flex:

```
%{
        #include"exemplo.tab.h"
        #include<math.h>
%}
digito  [0-9]
%%
[ \t]+
<<EOF>>      return 0;
```

```
{digito}+      yylval.valint=atoi(yytext);return INT;
({digito}*\.)?{digito}+([eE][-+]?{digito}+)?  yylval.valreal=atof(yytext);return
REAL;
"("|")"|"+"|"-"|"*"|"/"|"="|\n    return yytext[0];
.                    printf("erro lexico: %s\n",yytext);
%%
int yywrap () {
}
```

Exercícios:

1 - Escreva o primeiro programa em bison, isto é, o "Hello World".

2 - Implemente uma maquina de calcular, utilizando o bison, para as operações +, -, / e *. A maquina só sabe realizar operações com inteiros.

3 - Implemente um analisador sintáctico para reconhecimento duma expressão aritmética, utilizando o bison. A gramática é a seguinte:

S -> ID '=' E | E

E ->E '+' E | E '-' E | E '*' E | E '/' E | '-' E | (E) | ID| INT | REAL

onde ID é um identificador (letra de 'a' a 'z'), INT um número inteiro e REAL um número real.

O parser deve analisar múltiplas expressões e obter resultados, apresentando-os. Sempre que haja uma atribuição esse valor deve ser guardado para ser utilizado com o identificador respectivo.

Nota: Para se implementar o parser no bison é necessário rescrever a gramática para obedecer às precedências entre os operadores.

3 - Implemente uma calculadora usando o bison e a resolução do exercícios 2 acrescentando outros operadores.