



Linguagens Formais e Autómatos

(Ano lectivo de 2013/14)

Guiões das aulas práticas

Guião

Implementação de Gramaticas de Atributos em `bison+flex`

Sumário

Implementação de gramáticas de atributos em `bison+flex`.

Exercício 1 *Considere o programa em bison seguinte, que representa uma máquina de calcular. Analise-o, tendo em atenção o uso da `%union` para a declaração dos atributos, que `NUM`, `ID` e `OTHER` são símbolos terminais, cada um com um tipo de atributo associado, e que a função `yylex` não está incluída.*

```
%union
{
    double vnum;
    unsigned char vid;
    unsigned char vchar;
}

%{
    #include <stdio.h>

    int yylex(YYSSTYPE* yylval);

    int yyerror(const char *s);
}%

%token <vnum> NUM
%token <vid> ID
%token <vchar> OTHER

%type <vnum> F T E

%pure-parser

%defines

%%

S : E '\n' { printf(" = %g\n", $1); } S
  | error '\n' { printf("--> parse error\n"); } S
  | /* lambda */
  ;

E : T { $$ = $1; }
```

```

    | E '+' T { $$ = $1 + $3; }
    | E '-' T { $$ = $1 - $3; }
    ;

T : F { $$ = $1; }
    | T '*' F { $$ = $1 * $3; }
    | T '/' F { $$ = $1 / $3; }
    ;

F : NUM { $$ = $1; }
    | '(' E ')' { $$ = $2; }
    ;

```

```
%%
```

```

int yyerror(const char *s)
{
    return 0;
}

```

Considere agora o programa em *flex* seguinte, que implementa a função *yylex*, necessária ao programa em *bison* anterior. Analise-o, tendo em atenção a forma como os atributos dos símbolos terminais são usados.

```

%{
    #include <stdlib.h>

    #include "calc-parser.tab.h"

    #define YY_DECL int yylex(YYSTYPE* p)
%}

%option noyywrap
%option nounput
%option noinput

nint    [0-9]+
num      {nint}|{nint}\.{nint}

id       [a-z]

valid    [-+()*/=\n]

spaces   [ \t]+

%%

{num}    { p->vnum = atof(yytext); return NUM; }

{id}     { p->vid = yytext[0]; return ID; }

```

```

{spaces}    { }

{valid}     { return yytext[0]; }

.          { p->vchar = yytext[0]; return OTHER; }

%%

```

A geração do programa final, a partir das duas peças anteriores e do `main`, colocado num ficheiro à parte, é controlada através do *Makefile* seguinte. Analise-o, tentando perceber o seu papel, antes de gerar o programa.

```

CC = g++
CFLAGS = -Wall

target = calc

$(target): $(target).o $(target)-lexer.o $(target)-parser.tab.o
$(CC) -o $@ $^

$(target)-parser.tab.c $(target)-parser.tab.h: $(target)-parser.y
bison $<

$(target)-lexer.c: $(target)-lexer.l $(target)-parser.tab.h
flex -o$@ $<

clean:
rm -f $(target)-lexer.[co] $(target)-parser.tab.[och] $(target).o

cleanall: clean
rm -f $(target)

```

1. *Teste o programa dado.*
2. *Altere-o de modo a que aceite operandos do tipo real, incluindo em notação científica. A intervenção deve ser feita ao nível do `flex`.*
3. *Altere-o de modo a que aceite a operação de potência, com a precedência e associatividade adequadas. A intervenção deve ser feita ao nível do `bison`.*
4. *Altere-o de modo a suportar variáveis e instruções de atribuição. Considere numa primeira abordagem que uma variável é identificada por uma letra minúscula.*
5. *Extenda de modo a aceitar identificadores mais gerais.*