Licenciatura em Engenharia Informática – DEI/ISEP
Linguagens de Programação 2006/07

Tópicos avançados

Este documento apresenta alguns métodos para a resolução de problemas, susceptíveis de ocorrer na resolução do trabalho prático de LPROG. No final deste documento, é indicado o URL dum arquivo, contendo alguns dos exemplos apresentados.

1 Contextos do *FLEX* (start conditions)

O *FLEX* permite estabelecer contextos, para a aplicação das expressões regulares. Um contexto, é um estado especial, no qual só está activo um conjunto restrito de expressões regulares. Um dos exemplos mais conhecidos, implementado recorrendo a contextos, é o processamento de comentários multi-linha da linguagem de programação C. O contexto em que o *FLEX* se encontra por omissão, é o contexto "`0`" ou "`INITIAL`". Neste contexto todas as expressões regulares que não pertencem a nenhum contexto e as marcadas como "`INITIAL`" são usadas. Para definir novos contextos são usadas as directivas "%s" e "%x" na primeira secção do ficheiro *FLEX*, em que:

- A directiva "%s" define um contexto *inclusivo*, em que são aplicadas as expressões regulares desse contexto e as que não têm contexto;
- A directiva '%x" define um contexto *exclusivo*, isto é, quando o *FLEX* está nesse contexto, somente as expressões regulares marcadas com esse contexto estão activas.

A indicação dos contextos a que se aplica uma expressão regular é realizada, colocando um estado ou uma lista de estados, delimitado pelos símbolos "<" e ">", antes da expressão regular. Para expressões activas em todos os contextos, pode-se usar a notação "<*>".

A mudança de contextos é realizada nas acções com a macro `BEGIN`, indicando o contexto para o qual queremos mudar. De seguida é apresentado um exemplo que ignora os comentários da linguagem C, continuando a contar as mudanças de linha.

```
%x coment

%%

"/*"                    BEGIN coment; /*inicia comentário */
<coment>[^*\n]*         /* ignorar tudo até um '*' ou '\n' */
<coment>"*/"            BEGIN INITIAL; /*finaliza comentário */
<coment>"*"             /* ignorar '*' sozinho */

  /* a regra seguinte está activa nos dois contextos */
<INITIAL,coment>\n      numLinhas++;  // equivante a <*>

[0-9]+                  return INTEIRO;
[0-9]*\.[0-9]+          return REAL;

<*>[ \t\r]              /* ignorar em todos os contextos */
<<EOF>>                 return 0;
%%
```

O processamento de um comentário HTML é semelhante, sendo somente necessário substituir as expressões regulares de início e fim de comentário, pelas expressões regulares "`"<!--"`" e "`"-->"`" respectivamente, e fazer algumas alterações às restantes expressões regulares.

No caso de processamento das linguagens `XML` e `HTML`, pode ser necessária a criação de contextos *FLEX*, para garantir que tudo que se situa fora das etiquetas é tratado como um único *token*. É necessário também permitir que apareçam palavras reservadas fora das etiquetas, e sejam tratadas como texto normal. No exemplo de código seguinte é apresentado o extracto dum ficheiro `XML` com alguns problemas de análise com o *FLEX* e o *BISON*, sem recorrer à utilização de contextos no *FLEX*.

```
<?xml version="1.0"?>
<language>
  <extension>xml</extension>
  <name>extensible markup language</name>
  <update>2005-04-12 12:10:22</update>
</language>
<language>
  <extension>html</extension>
  <name>hypertext markup language</name>
  <update>2003-09-24 19:44:01</update>
</language>
```

No exemplo anterior a palavra `xml` aparece como etiqueta e como texto, é necessário identificar esta diferença recorrendo a dois contextos no *FLEX*, um quando se está dentro duma etiqueta, e outro quando se está fora.

Também fora duma etiqueta é necessário diferençar vários casos, dependendo da etiqueta a tratar. Por exemplo na etiqueta "`extension`" existe

somente uma palavra, já na etiqueta "`name`" existe qualquer coisa (até encontrar o "`<`" seguinte) e na etiqueta "`update`" existe uma data que tem de ser validada do lado do *BISON*, necessitando dum contexto especial *FLEX* que identifique os inteiros e os símbolos "-" e ":" até encontrar o símbolo "`<`".

De seguida apresenta-se uma maneira de identificar a etiqueta "`name`" e "`update`" no *FLEX* e no *BISON*

```
...
language: '<' LANGUAGE '>'
                dados_language
            TAG_FIM LANGUAGE '>' {printf("language...\n");}
;
dados_language: extension name update
                | error {yyerror("unrecognized
                  language");yynerrs++;}
;
...
name:   '<' NAME '>' beginStrxml
           strxml
         TAG_FIM NAME '>' {printf("name ...%s\n",$5);}
;
beginStrxml: {printf("begin stringxml...\n");
               BEGIN stringxml;}
;
update: '<' UPDATE '>' beginData
           dataerr
         TAG_FIM UPDATE '>'{printf("data ...\n");}
;
beginData: {printf("begin data...\n");BEGIN data;}
;
dataerr: data hora
       | error hora {yyerror("malformed data...");yynerrs++;}
       | data error {yyerror("malformed hora...");yynerrs++;}
;
data:   INTEIRO '-' INTEIRO '-' INTEIRO {printf("dia
    ...%d-%d-%d\n",$1,$3,$5);}
;
hora:   INTEIRO ':' INTEIRO ':' INTEIRO {printf("hora
    ...%d:%d:%d\n",$1,$3,$5);}
;
...
```

Sempre que é necessário usar a macro `BEGIN` no *BISON* para mudar o contexto do *FLEX*, deve ser criada uma regra vazia com a respectiva acção semântica, para esse fim (neste exemplo, são usadas as regras beginStrxml e beginData). Este motivo deve-se à necessidade do *FLEX* mudar imediatamente para este contexto, o que pode não acontecer com a inclusão de acções semânticas no meio de regras, tal como foi descrito na Ficha 6.

Para poder usar a macro `BEGIN` e os contextos dentro do *BISON*, torna-se necessário ao correr o comando `flex`, usar a opção `‘‘-header-file=nome.h’’` para gerar um ficheiro *header* que será incluído no *BISON*. Quem utilizar o *FLEX* para Windows, não tem a opção de criar o ficheiro *header*, pelo que terá de incluir o ficheiro C gerado pelo *FLEX* no *BISON* e compilar somente o ficheiro *BISON*.

Um analisador léxico para esta gramática seria:

```

%{
  extern int coluna,linha;
%}

%x tag data stringxml

ID          [a-zA-Z]+
INTEIRO     [0-9]+
REAL        [0-9]*\.[0-9]+

%%

<*>"<"                  BEGIN tag; coluna+=yyleng; return yytext[0];
<*>"</"                 BEGIN tag; coluna+=yyleng; return TAG_FIM;
<tag>">"                BEGIN INITIAL; coluna+=yyleng; return yytext[0];
<tag>xml                coluna+=yyleng; return XML;
<tag>extension          coluna+=yyleng; return EXTENSION;
<tag>language           coluna+=yyleng; return LANGUAGE;
<tag>name               coluna+=yyleng; return NAME;
<tag>update             coluna+=yyleng; return UPDATE;
<tag>[=?/]              coluna+=yyleng; return yytext[0];
<tag>\"{REAL}\" |
<tag>'{REAL}' coluna+=yyleng;yylval.real=atof(yytext+1);return REAL_STR;

<tag,data,INITIAL>[ \t\r]    coluna+=yyleng;
<tag,data,INITIAL>\n         coluna=0;linha++;

<data>{INTEIRO}         coluna+=yyleng; yylval.inteiro=atoi(yytext); return INTEIRO;
<data>[:-]              coluna+=yyleng; return yytext[0];

<stringxml>[^>]*        BEGIN INITIAL; coluna+=yyleng; yylval.str=strdup(yytext); return STRING_XML;

<*>[ \t\r]              coluna+=yyleng;
<*>\n                   coluna=1;linha++;
```

```
<*>.                    coluna+=yyleng; printf("Erro lexico(%d-%d): simbolo desconhecido (%c)\n", linha, coluna, yytext[0]);
<<EOF>>                 return 0;
```

2 Validação da existência de elementos por ordem arbitrária numa lista

Para a validação da existência ou não de elementos numa lista e se eles estão repetidos, é necessário recorrer a acções semânticas para evitar a criação exponencial, de regras alternativas.

O exemplo seguinte aceita dentro da etiqueta uma lista de três campos obrigatórios, mas que podem aparecer por qualquer ordem. Para a sua validação é necessário usar um vector (ou uma estrutura) com um elemento para cada campo. A regra lista é a regra que permite inserir os vários campos, sendo na acção da condição de paragem (a primeira acção a ser realizada) iniciado o vector.

```
regra:
      '<' TAG lista {
            if(vecCampos[0]==0)
              yyerror("Falta campo 0");
            else
              if(vecCampos[0]>1) {
                sprintf(str,"Campo 0 repetido %d vezes",vecCampos[0]);
                yyerror(str);
              }
            if(vecCampos[1]==0)
              yyerror("Falta campo 1");
            else
              ...
        }
      '/' '>
    |
      '<' TAG error '/' '>   {yyerror("Erro dentro da tag");}
;

lista: /* Vazio */ {vecCampos[0]= vecCampos[1]= vecCampos[2]=0;}
  | lista campo
;

campo:CAMPO_0 {
        if(vecCampos[0]>0) yyerror("Campo 0 repetido");
          vecCampos[0]++;
```

```
27          }
28        |CAMPO_1 {
29            if(vecCampos[1]>0) yyerror("Campo 1 repetido");
30              vecCampos[1]++;
31          }
32        |CAMPO_2 {
33            if(vecCampos[2]>0) yyerror("Campo 2 repetido");
34              vecCampos[2]++;
35          }
36        ;
```

3 Passagem de *strings* entre o *FLEX* e o *BISON*

Na passagem de strings entre o *FLEX* e o *BISON* deve-se ter o cuidado de reservar espaço e copiar a string no *FLEX* (por exemplo usando a função `strdup`). No *BISON* deve ser libertado o espaço quando o *token* já não é utilizado. Isto pode ser realizado de duas maneiras diferentes, a primeira é incluindo a instrução `free` directamente nas acções semânticas, a outra é incluindo a instrução `free` na directiva do *BISON* "`%destructor`" (somente disponível a partir da versão 1.9 do *BISON*).

De seguida são apresentados dois pequenos exemplos do *FLEX* e do *BISON* que utilizam este mecanismo.

```
1 ...
2 %%
3 ...
4 [a-zA-Z_][a-zA-Z0-9_]*   yylval.str=strdup(yytext); return ID;
5 ...
6 %%
```

```
1
2 %union{
3   int      inteiro;
4   double   real;
5   char    *str;
6 }
7
8 %token <str> ID
9
10 %type <str> id
11
12 %destructor {free($$);} ID id
13
14 %%
```

```
15 inicio: /* vazio */
16        | inicio id      {printf("id:%s\n",$2);
17                /* se não usar %destructor, fazer free($2); aqui */}
18 ;
19
20 id: ID        {$$=$1;}
21 ;
22
23 %%
```

Os dois exemplos usados neste documento, estão disponíveis no endereço `http://www.dei.isep.ipp.pt/~matos/lprog/topicos.zip`