Linguagens Formais e Autómatos + Compiladores

(Ano letivo de 2015/2016)

Guiões das aulas práticas

Guião #02 — Exercícios usando o `flex`

Sumário

Resolução de exercícios sobre linguagens usando expressões regulares em `flex+C`.

Introdução

`flex` é uma linguagem de programação e uma ferramenta de compilação que permite decompor uma sequência de caracteres de entrada numa sequência elementos lexicais, normalmente designados por *tokens*. A programação é conduzida por padrões (expressões regulares) que definem os elementos lexicais.

Exercício 1 *Sobre o alfabeto* $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$*, considere a linguagem regular*

$$L = \{u \in A^* \ : \ \exists_i \ x_i = x_{i-1} = x_{i-2}\}$$

onde A^* *representa o conjunto de todas as* strings *que se podem construir usando os caracteres de A. O código em flex+C seguinte (`flex-model-1.l`) permite criar um programa que processa as linhas de um ficheiro de entrada dado (por defeito o* stdin*) indicando as que pertencem a L, as que não pertencem e as mal formadas (que contêm caracteres fora do alfabeto considerado).*

```
%{
    #include <stdio.h>

    #define belongs(t) fprintf(yyout, "\"%s\" - belongs",t)
    #define other(t) fprintf(yyout, "\"%s\" - does not belong",t)
    #define invalid(t) fprintf(yyout, "\"%s\" - contains invalid characters",t)
%}

%option noyywrap
%option reentrant
%option noinput nounput

er1     (aaa|bbb|ccc)
any     [abc]*
other   [abc]*

%%

^{any}{er1}{any}$   { belongs(yytext); }
^{other}$           { other(yytext); }
\n                  { ECHO; }
.+                  { invalid(yytext); }

%%
```

```
int main(int argc, char *argv[])
{
    /* init the scanner */
    yyscan_t scanner;
    if (yylex_init(&scanner))
    {
        fprintf(stderr, "Fail initing scanner\n");
        return(EXIT_FAILURE);
    }

    /* prepare input stream */
    switch (argc)
    {
        case 1: // read from stdin, the default
        {
            yyset_in(stdin, scanner);
            break;
        }
        case 2: // read from given file
        {
            FILE* fin;
            if ((fin = fopen(argv[1], "r")) == NULL)
            {
                fprintf(stderr, "Fail openning input file \"%s\"\n", argv[1]);
                exit(EXIT_FAILURE);
            }
            yyset_in(fin, scanner);
            break;
        }
        default:
        {
            fprintf(stderr, "Wrong number of arguments\n");
            exit(EXIT_FAILURE);
        }
    }

    /* do scanning */
    while (yylex(scanner) != 0)
    {
    }

    /* clean up and quit */
    yylex_destroy(scanner);
    return 0;
}
```

(a) Descarregue o programa anterior e analise-o.

(b) Compile-o. É fornecida um `Makefile` *para facilitar o processo de compilação.*

(c) Execute-o, quer sem argumentos (modo interativo), quer passando-lhe um ficheiro como argumento. Use o ficheiro `words.txt` *dado.*

Exercício 2 *Sobre o alfabeto* $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$*, considere as linguagens regulares*

$$L_1 = \{u \in A^* \ : \ \#(\mathtt{ab}, u) > 1\}$$

$$L_2 = \{u \in A^* \ : \ \#(\mathtt{aba}, u) > 1\}$$

$$L_3 = \{u \in A^* \ : \ \#(\mathtt{b}, u) = 0 \ \vee \ \#(\mathtt{a}, u)\%2 = 0\}$$

$$L_4 = \{u \in A^* \ : \ (i < j \ \wedge \ u_i = u_j = \mathtt{b} \ \wedge \ \forall_{i<k<j} \, u_k \neq \mathtt{b}) \implies (\forall_{i<n<j} \forall_{i<m<j} \, u_n = u_m)\}$$

$$L_5 = \{u \in A^* \ : \ \#(\mathtt{ab}, u) = 1\}$$

$$L_6 = \{u \in A^* \ : \ \#(\mathtt{aba}, u) < 3\}$$

$$L_7 = \{u \in A^* \ : \ \#(\mathtt{abc}, u) \geq (\#(\mathtt{ca}, u) + \#(\mathtt{cb}, u))\}$$

Para cada uma das linguagens anteriores:

(a) construa um programa em ***flex+C*** *que reconheça as suas palavras. Use o programa do exercício anterior como modelo. No essencial, bastar determinar e aplicar as expressões regulares que descrevem as várias linguagens.*

Exercício 3 *O código em flex+C do ficheiro* `flex-model-2.l` *é funcionalmente equivalente ao do exercício 1 mas delega na função* `main` *a impressão das palavras e a sua classificação.*

(a) Descarregue-o e analise-o.

(b) Identifique as diferenças em relação ao do exercício 1.

Exercício 4 *Considere que se pretende desenvolver uma aplicação que implemente uma máquina de calcular funcionalmente equivalente ao comando "bc -l". A interação com o utilizador é feita através dos* standards input *e* output*.*

(a) Usando o ficheiro `flex-model-2.l` *como modelo, construa um programa que receba na entrada linhas representando expressões aritméticas e produza a sua decomposição lexical na saída. Os tokens devem ser impressos na forma <*`tokenId,tokenValue`*>, e devem ser considerados os seguintes:*

tokenId	***Descrição***
NUM	*número inteiro ou real*
ID	*identificador de variável*
ADDSUB	*operador de adição ou subtração*
MULDIV	*operador de multiplicação ou divisão*
POW	*operador de potência*
ASIGN	*operador de atribuição*
EOL	*fim de linha*
INVAL	*carácter inválido*

(b) ...