Trabalho 1:
Gerador de Analisadores Léxicos

Linguagens Formais e Compiladores
Prof^a. Jerusa Marchi

1. Objetivo do Trabalho:

   O objetivo deste trabalho é o desenvolvimento de um arcabouço (framework) para gerar analisadores léxicos. O arcabouço consiste na implementação dos algoritmos necessários para a criação de um Autômato Finito para atender as definições regulares expressas no arquivo de entrada. A saída deve ser a lista de tokens gerada, associando ao lexema, seu padrão.

   Para a construção de um gerador de analisador léxico são necessários os seguintes algoritmos:

   (a) Conversão de Expressão Regular para Autômato Finito Determinístico (algoritmo apresentado no livro do Aho);

   (b) Mimimização de Autômatos;

   (c) União de Autômatos via ϵ-transição;

   (d) Determinização de Autômatos;

   O framework deve prover uma interface para o projeto de um novo analisador léxico e uma interface de execução para uso do analisador léxico gerado.

   - A **interface para o projeto de um Gerador de Analisador Léxico** deve permitir a inclusão de expressões regulares para todos os padrões de tokens, usando definições regulares;
   - Para cada ER deve ser gerado o AFD corresponde;
   - Os AFD devem ser minimizados e Unidos com ϵ-transições;
   - O AFND resultante deve ser determinizado gerando a tabela de análise léxica (representação implícita).
   - A **interface de execução** do analisador léxico deve permitir a entrada de um texto fonte (conjunto de palavras, simulando um programa fonte);
   - O texto fonte será analisado, utilizando a tabela de análise léxica gerada na parte de projeto, e deve gerar um arquivo de saída com a lista de todos os tokes encontrados na forma $\langle$lexema, padrão$\rangle$ ou reportar erro $\langle$lexema, erro!$\rangle$ caso a entrada não seja válida.

   Observações:

   - As notações devem ser as utilizadas em sala (para notacionar ε usar o $\&$);

- A tabela de análise léxica deve poder ser “visualisada”;
- Os Autômatos Finitos gerados pela Conversão das ERs também devem poder ser visualisados (arquivo ou tela - na forma de tabela);

2. Realização:

   O trabalho deverá ser realizado em grupos de no mínimo 2 e no máximo 3 integrantes. Os grupos deverão, ao final do semestre apresentar o trabalho em dia e horário agendado (juntamente com o analisador sintático, de forma integrada), onde serão executados testes das partes em separado (por esta razão é muito importante que as partes possam ser testadas de forma independente). Além dos fontes, executáveis e de um HowTo, o grupo deve prover exemplos variados de testes, incluindo diferentes definições léxicas para que o trabalho possa ser melhor avaliado.

3. Avaliação:

   O trabalho será avaliado pela robustez e corretude dos algoritmos. Além disso, quesitos como legibilidade do código e organização dos fontes também serão considerados.

1 Anexo I - Formato de entrada de ER

Os arquivos com ER devem seguir o padrão:

def-reg1: ER1
def-reg2: ER2
. . .
def-regn: ERn

As ER devem aceitar grupos como [a-zA-Z] e [0-9] e os operadores usuais de * (fecho), + (fecho positivo), ? (0 ou 1) e | (ou).

- Exemplo1:

  id: [a-zA-Z]([a-zA-Z] | [0-9])*
  num: [1-9]([0-9])* | 0

- Exemplo 2:

  er1: a?(a | b)+
  er2: b?(a | b)+

2 Anexo II - Exemplo de entrada do arquivo de teste e possível saída

Os arquivos de teste dependem das definições regulares introduzidas, assim considerando o exemplo 1 e 2 do anexo 1, tem-se

- Exemplo entrada para ER que reconhece identificadores e números:

  a1
  0
  teste2
  21
  alpha123
  3444
  a43teste

  que irá gerar a seguinte lista de tokens:

  <a1,id>
  <0, num>
  <teste2, id>
  <21, num>
  <alpha123, id>
  <3444, num>
  <a43teste, id>

- Exemplo entrada para a ER que reconhece palavras que começam com a ou palavras que começam com b:

  aa
  bbbba
  ababab
  bbbbb

  que irá gerar a seguinte lista de tokens:

  <aa , er1>
  <bbbba, er2>
  <ababab, er1>
  <bbbbb, er2>