INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
MINAS GERAIS
Campus Formiga

Graduação em Ciência da Computação

Disciplina: Linguagens Formais e Autômatos

Professor: Walace de Almeida Rodrigues

1o Trabalho Prático

Instruções Gerais Para o Trabalho:

1. Esta atividade poderá ser resolvida em grupo com no máximo 2 integrantes.
2. Caso você ache que falta algum detalhe nas especificações, você deverá fazer as suposições que julgar necessárias e escrevê-las no seu relatório. Pode acontecer também que a descrição dessa atividade contenha dados e/ou especificações supérfluas para sua solução. Utilize sua capacidade de julgamento para separar o supérfluo do necessário.
3. Para desenvolver esta atividade utilize preferencialmente as linguagens Python ou Java. Se quiser utilizar outra linguagem deverá justificar a sua escolha no seu relatório.
4. Como produtos da atividade serão gerados dois artefatos: códigos fontes da implementação e documentação da atividade.
5. Cada arquivo-fonte deve ter um cabeçalho constando as seguintes informações: nome(s) do(s) aluno(s), matrícula(s) e data.
6. O arquivo contendo a documentação da atividade (relatório) deve ser devidamente identificado com o(s) nome(s) e matrícula do(s) autor(es) do trabalho. O arquivo contendo o relatório deve, obrigatoriamente, estar no formado PDF.
7. Devem ser entregues os arquivos contendo os códigos-fontes e o arquivo contendo a documentação da atividade (relatório). Compacte todos os artefatos gerados num único arquivo no formato ZIP.
8. O prazo final para entrega desta atividade é até 23:59:00 do dia 2/11/2017.
9. O envio é de total responsabilidade do aluno. Não serão aceitos trabalhos enviados fora do prazo estabelecido.
10. Trabalhos plagiados serão desconsiderados, sendo atribuída nota 0 (zero) a todos os envolvidos.

1 Introdução

O Primeiro Trabalho Prático (TP) aborda o desenvolvimento (projeto e implementação) dos algoritmos apresentados em sala de aula para manipulação de Autômatos Finitos (AFs) sem pilha.

A primeira parte desse TP considera a implementação de uma classe para representar os AFs estendidos seguindo a especificação apresentada a seguir. As partes seguintes consideram a implementação dos métodos que manipulam os objetos da classe.

A especificação dos requisitos que é apresentada aqui não é formal, em vez disso será apresentado somente o ponto de vista de um possível usuário da classe, mostrando aquilo que ele esperaria encontrar numa implementação. Serão apresentados exemplos de código mostrando a espectativa de um usuário que deseja utilizar a classe e, com esses exemplos em vista, tente projetar a sua classe de modo a facilitar seu uso pelo usuário. Todavia, esteja ciente de que em computação não vale a máxima "o usuário tem sempre razão". Você tem liberdade para alterar alguma coisa que pareça menos adequada nos exemplos, mas esteja preparado para justificar as suas escolhas e convenver o usuário de que a sua implementação foi projetada e é mais fácil de usar.

Os códigos que exemplificam o uso da classe não estão escritos em nenhuma linguagem de programação particular, então faça as adaptações necessárias.

2 A classe AF

Um Autômato Finito Estendido (AFE) é definido como uma tupla $(E, \Sigma, \delta, I, F)$ com os elementos:

- E – conjunto finito de estados.
- Σ – alfabeto, formado por um conjunto finito de símbolos.
- δ – função de transição estendida, tipada como $\delta : (E \times \Sigma*) \rightarrow E$.
- i – conjunto de estados iniciais, tal que $I \subseteq E$.
- F – conjunto de estados finais, com $F \subseteq E$.

Para simplificação do modelo, considere que cada estado do AFE é representado por um inteiro positivo e o alfabeto pertence ao conjunto de caracteres imprimíveis da tabela ASCii.

3 Os requisitos através de exemplos de uso

1. O usuário deseja aproveitar a ferramenta JFLAP para fazer a entrada dos dados. Desse modo, ele poderá criar um AFE utlizando o JFLAP, salvá-lo no formato **.jff**, e então carregá-lo num objeto da classe AFE. Essa tarefa não será difícil tendo em vista que um arquivo **.jff** nada mais é do que um arquivo texto em XML.

```
AFE m;
...
m.load("entrada.jff");
```

2. O usuário também deseja aproveitar a ferramenta JFLAP para fazer a saída dos dados. Desse modo, ele poderá salvar uma representação do AFE no objeto para um arquivo **.jff**, depois carregar esse arquivo no JFLAP para visualizar graficamente o AFD.

```
AFE m;
...
m.salve("saida.jff");
```

3. Considere que um AFD (autômato determinístico) é um caso particular de AFN (autômato não determinístico), que é um caso particular de AFV (autômato não determinístico com movimentos vazios), que é um caso particular de AFE (autômato estendido). O usuário deseja poder identificar esses casos particulares, e também se o autômato (AFD) está completo ou não (tem estado de erro implícito).

```
AFE m;
boolean resposta;
...
resposta = m.afv();        // o afe é um afv?
resposta = m.afn();        // o afe é um afn?
resposta = m.afd();        // o afe é um afd?
resposta = m.completo(); // os estados de erro estão explícitos?
```

4. O usuário deseja poder realizar as conversões (estudadas em aula) entre os tipos apresentados de autômatos.

```
AFE m;
...
m.conv2AFV();        // converte o afe para afv
m.conv2AFN();        // converte o afv para afn
m.conv2AFD();        // converte o afn para afd
m.conv2COMPLETO()    // torna o estado de erro explícito
```

5. O usuário deseja poder identificar estados equivalentes num AFD, e também obter uma versão mínima do AFD.

```
AFE m, mm;
List eqv;
...
if ( m.afd() ) {
   eqv = m.equivalents();    // calcula lista de estados equivalentes
   mm = m.copy();            // faz uma cópia do autômato
   mm.conv2MINIMUM();        // torna o autômato mínimo
}
```

6. O usuário deseja poder comparar dois AFDs para saber se são ou não equivalentes.

```
AFE m1, m2;
...
if ( m1.afd() && m2.afd() )
   if ( m1.equivalent(m2) )
      print("sim");
   else
      print("não");
```

7. O usuário deseja poder realizar (apenas em AFDs!) operações de complementação, união, intersessão e diferença.

```
AFE m1, m2, m3, m4, m5, m6;
...
m3 = m1.complement();
m4 = m1.union(m2);
m5 = m1.intersection(m2);
m6 = m1.difference(m2);
```

8. O usuário deseja poder consultar o AFD (apenas em AFDs!), testar a pertença de uma palavra na linguagem, testar movimentos.

```
AFE m;
int estado;
...
if (m.accept("aaabbbaa"))
   print("aceita");
else
   print("não aceita");
...
estado = m.initial();
estado = m.move(estado, "aaab");
if (estado in m.finals())
   print("aceita");
else
   print("não aceita");
```

9. O usuário deseja poder alterar o AFE.

```
AFE m;
...
m.addState(id=10, initial=false, final=true);
m.addTransition(source=1, target=2, consume="");
m.addTransition(source=1, target=2, consume="aba");
m.deleteState(3);
m.deleteTransition(source=1, target=4, consume="a");
```

4 Critérios de Correção

Serão adotados os seguintes critérios de correção para o trabalho:

1. **correção:** somente serão corrigidos códigos portáveis e sem de erros de compilação;
2. **precisão:** execução correta numa bateria de testes práticos;
3. **modularização:** projeto da classe;
4. **qualidade do código fonte:** legibilidade, indentação, uso adequado de comentários;
5. **documentação:** relatório, conforme instruções apresentadas a seguir.

Os critérios não receberão pontuação específica, mas funcionarão como "requisitos" que poderão afetar a pontuação se não forem atendidos adequadamente, portanto leve-os em conta durante o desenvolvimento do trabalho. Haverá uma apresentação oral e individual do trabalho. Todos os integrantes do grupo devem participar dessa apresentação. Na ausência de plágio, as notas dos trabalhos corretos serão computadas individualmente da seguinte forma: nota = nota_apresentacao $*$ nota_trabalho, ou seja, a nota final é ponderada pela nota da apresentação.

4.1 Documentação

Esta seção descreve o formato e o conteúdo do relatório que deve ser gerado como produto final do trabalho. Seja sucinto: em geral é possível escrever um bom relatório entre 2 e 4 páginas.

O relatório documentando seu sistema deve conter as seguintes informações:

1. **introdução:** descrever o problema resolvido e apresentar uma visão geral do sistema implementado;
2. **implementação:** descrever as decisões de projeto e implementação do programa. Essa parte da documentação deve mostrar como as estruturas de dados foram planejadas e implementadas. Sugestao: mostre uma figura ilustrativa, os tipos definidos, detalhes de implementação e especificação que porventura estavam omissos no enunciado, etc.
3. **validação:** descrição dos testes que o grupo fez para validar o trabalho, se seguiu alguma metodologia etc.
4. **conclusão:** avaliação do grupo sobre o trabalho considerando a experiência adquirida, a contribuição para o aprendizado da disciplina, as principais dificuldades encontradas ao implementá-lo e como tais dificuldades foram superadas;