Uma Hierarquia de Classes para Construção de Autômatos Finitos

Kely Teixeira Soares, Lilian Ribeiro, Walace de Almeida Rodrigues
Curso de Ciência da Computação – Centro Universitário de Formiga (UNIFOR-MG)
Av. Dr. Arnaldo de Senna, 328 - Água Vermelha - Cep: 35570-000 - Formiga - MG - Brasil
(ksoares, lribeiro, walace)@comp.uniformg.edu.br

Resumo. Este artigo descreve um conjunto de classes desenvolvidas em Java para a construção de Autômatos Finitos. As classes em questão permitem representar autômatos finitos determinísticos e exemplos de utilização serão apresentados. O trabalho aqui descrito faz parte de outro maior que está sendo desenvolvido como projeto de conclusão do Curso de Ciência da Computação por alunos da UNIFOR-MG.

Palavras-Chave: Autômatos Finitos; Java; conjunto de classes.

An Hierarchy of Classes for Development of Finite Automata

Abstract. This article describes a set of JAVA classes for development of finite automata. The classes allows the representation of deterministic finite automata. Several examples are presented. This article is part of another research work that is being developed in the Department of Computer Science of UNIFOR-MG.

Keywords: Finite Automata, programming languages, JAVA, hierarchy, set of classes.

(Received October 29, 2004 / Accepted December 12, 2004)

1 Introdução

Autômatos são máquinas utilizadas para tratar diversos problemas computacionais. Um autômato é um modelo que trabalha com estados, sendo que cada estado representa a situação atual do processo; nenhum estado de um autômato se preocupa com situações anteriores, por isso, estados não têm memória. Existem diversos tipos de autômatos e esses podem ser caracterizados segundo o número de estados e os tipos de transições que possuem. O tipo mais comum é o AF (autômato finito) que possui um número finito de estados.

Neste trabalho, essas máquinas são representadas por meio de três classes: Estado, Transição e Autômato. As classes foram implementadas em Java, explorando os recursos da orientação a objetos, para serem utilizadas em implementações práticas que incluem: tratamento de correção, minimização e conversão de autômatos e a construção de um simulador de Autômatos. Uma das motivações para este trabalho foi a constatação das limitações dos materiais disponíveis para o ensino dos autômatos. Um dos objetivos almejados é auxiliar estudantes cursando disciplinas de Teoria da Computação.

2 Autômatos

Um autômato finito funciona como uma máquina composta de três partes: fita, unidade de controle e função de transição. A fita é um dispositivo finito que contém a entrada a ser processada. A unidade de controle representa o estado corrente da máquina.

A unidade de leitura (cabeça da fita) movimenta-se para a direita acessando uma célula da fita de cada vez. A função de transição ou programa comanda a leitura e define o estado da máquina. A função programa pode ser representada como um grafo finito direto, como ilustrado na figura 1 apresentada abaixo.

O processamento de um AF, para uma entrada qualquer, consiste na sucessiva aplicação da função de transição para cada símbolo da entrada (da esquerda para a direita), até ocorrer uma condição de parada.

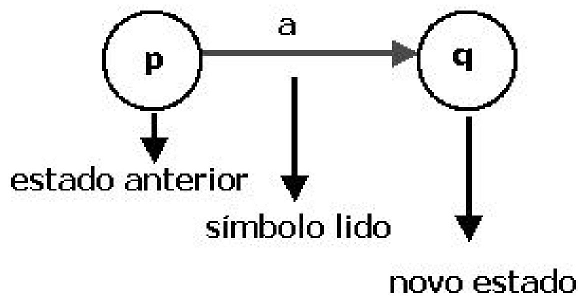

Figura 1: Representação da função programa em um grafo

Todo AF deve ter apenas um estado inicial e, no mínimo, um estado final. Após processar o último símbolo da fita, se o AF assumir um estado final, a entrada é aceita, caso contrário, é rejeitada.

3 As Classes

O uso da programação orientada a objetos tornou mais fácil a construção e a manipulação das unidades necessárias para a implementação dos AFs. As três classes utilizadas – Estado, Transição, Autômato – encapsulam os conceitos e escondem os detalhes de implementação, o que facilita a utilização da estrutura.

3.1 Classe Estado

A classe Estado implementa o modelo que será utilizado por todos os estados do AF. A cada novo estado adicionado ao AF, corresponderá um novo objeto da classe Estado que será armazenado na representação do AF. O novo objeto possuirá: um identificador; as coordenadas de posição na interface gráfica; o nome do estado formado da junção da string "q" com o identificador; e o rótulo que pode ser um comentário referente ao estado.

Estado

`int identificador`
`Point coordenadas`
`String nome`
`String rotulo`

`public Estado(int identicador, Point ponto)`
`public void definePonto(Point ponto)`
`public Point retornaPonto()`
`public void defineID(int identificador)`
`public int retornaID()`
`public void defineNome(String nome)`
`public String retornaNome()`
`public void defineRotulo(String rotulo)`
`public String retornaRotulo()`

Tabela 1: Esquema da classe Estado

3.2 Classe Transição

A classe Transição implementa o modelo que será utilizado por todas as transições do AF. A cada nova transição adicionada pelo usuário, corresponderá um novo objeto da classe transição. O novo objeto possuirá estado de origem, estado de destino e símbolo.

A classe Transição implementa diversos métodos, os quais aparecem no esquema mostrado na tabela 2.

3.3 Classe Autômato

A classe Autômato permite ao usuário criar e manipular Autômatos Finitos determinísticos utilizando as classes Estado e Transição. Após a criação de um objeto da classe Autômato, o usuário poderá realizar todas as operações necessárias para manipulação de Autômatos.

A classe Autômato oculta detalhes de implementação e permite um acesso intuitivo aos serviços disponibilizados, facilitando a utilização das classes. Por exemplo, os métodos eEstado(int identificador) e eEstado(Estado estado) retornam *true*, caso exista um certo estado, ou *false*, caso ele não exista; em ambos os casos, a identificação do estado pode ser feita por meio do identificador do objeto ou pelo próprio objeto. A sobrecarga nos métodos torna mais fácil sua utilização.

Para armazenamento dos objetos estado e transição dentro do AF, foram escolhidas estruturas que utilizam Hash, pois outras estruturas como vetores e listas possuem um inconveniente: a remoção ou inserção de um elemento exigirá que todos sejam reorganizados. Nas estruturas Hash, cada elemento possui um código de identificação que será utilizado nos algorítmos de armazenamento e busca, para se obter maior eficiência [2].

Para criar um estado, o usuário precisa informar apenas um identificador do tipo inteiro que deve ser único, já que o autômato não permite a existência de identificadores duplicados. Depois de criar um estado, o usuário pode defini-lo como sendo estado inicial ou estado final. Os objetos da classe Estado são armazenados em duas estruturas HashMap, chamadas estados e estadosFinais, sendo a chave de acesso o identificador. No caso do estado ser definido como inicial, este não será armazenado em nenhuma das estruturas citadas, uma vez que existe apenas um estado inicial. No caso do estado ser definido como final, será armazenado na estrutura estadosFinais. Para remover um estado, basta informar o identificador do mesmo para o método removeEstado(int identificador), já que na remoção de um estado, todas as transições que

contêm o estado como destino ou origem também são removidas.

O armazenamento dos objetos da classe Transição é feito em uma estrutura HashSet, chamada transições. Para cada objeto da classe Transição será criado um código único, obtido através de cálculos feitos a partir dos elementos de criação dos objetos, o qual será utilizado para garantir que não existam objetos duplicados. Para criar uma transição, o usuário precisa informar o identificador do estado de origem, o identificador do estado de destino e o símbolo de entrada. Se os identificadores informados pelo usuário forem identificadores de estados já existentes e a transição ainda não existir, então a transição será criada. Na remoção de uma transição, os parâmetros são os mesmos informados durante a criação da mesma, pelo método removeTransicao(int origem, int destino, String simbolo). Essas classes foram desenvolvidas para a construção de aplicações práticas, como, por exemplo, a construção de um sistema para resolução de problemas com autômatos finitos. Este sistema será composto de um editor e um simulador, mais os algorítmos para minimização [3], [5], conversão [1] e checagem de correção [4] de autômatos.

A classe Autômato implementa diversos métodos que, em conjunto, cuidam do bom funcionamento da estrutura. Alguns métodos aparecem no esquema mostrado na tabela 3.

Transicao
`Estado origem`
`Estado destino`
`String símbolo`
`public Transicao(Estado origem, Estado destino, String simbolo)`
`public void defineEstadoOrigem(Estado origem)`
`public Estado retornaEstadoOrigem()`
`public void defineEstadoDestino(Estado destino)`
`public Estado retornaEstadoDestino()`
`public void defineSimbolo(String simbolo)`
`public String retornaSimbolo()`
`public int hashCode()`
`public boolean equals(Object obj)`
`public String toString()`

Tabela 2: Esquema da classe Transição

Automato
HashMap estados, estadosFinais
HashSet transicoes
Estado estadoInicial
public void defineEstado(int identificador)
public void defineEstadoFinal(int identificador)
public void defineEstadoInicial(int identificador)
public void removeEstado(int identificador)
public void defineTransicao(int origem, int destino, String simbolo)
public void removeTransicao(int origem, int destino, String simbolo)

Tabela 3: Esquema da classe Autômato

4 Exemplos de Utilização

Os AFDs (Autômatos Finitos Determinísticos) podem ser utilizados para resolver vários problemas práticos, e muitas aplicações simples os utilizam como suporte na resolução de suas tarefas. Entre os exemplos comuns, freqüentemente apresentados em disciplinas que abordam autômatos, estão: máquina de refrigerantes, jogo do vinte e um, torre de hanói, jogo dos canibais, simulador de cartão telefônico, dentre outros.

Para exemplificar a utilização das classes, encontra-se em avançada fase de desenvolvimento um sistema para resolução de problemas com AFDs. Dentre as muitas funções que o sistema oferece, foi escolhida uma como exemplo. O ponto de partida é o problema: como verificar se um determinado AFD construído está ou não correto?

O sistema oferece um meio para efetuar essa checagem, uma vez que traz implementados, embutidos, muitos dos algoritmos clássicos empregados com os autômatos. No caso presente, utilizando as propriedades da equivalência entre dois AFDs, é possível verificar se as linguagens de dois AFDs são equivalentes do seguinte modo: o usuário forneceria ao sistema dois AFDs, sendo que um é o AFD sabidamente correto, e outro é o AFD candidato, para o qual o usuário deseja verificar a correção. A verificação é realizada através de comparações: se as linguagens do AFD candidato e do AFD correto forem equivalentes, pode-se afirmar que o AFD candidato está correto [4].

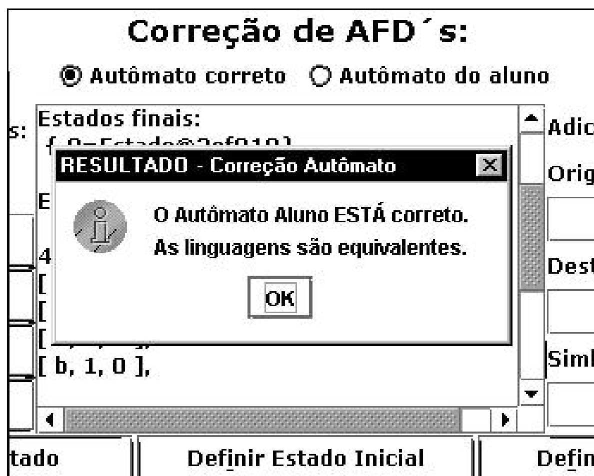

Figura 2: Mensagem do sistema para um AFD correto

Para efetuar a entrada dos dados, e para criar os autômatos, o sistema faz uso dos métodos já apresentados anteriormente. A apresentação foi breve, mas o entendimento é claro e o seu uso é intuitivo.

4.1 Implementação de um caso real

A seguir será apresentado como exemplo um programa que utiliza a hierarquia de classes. Este exemplo mostra como criar um AFD reconhecedor de palavras com número par de b's e alfabeto = {a,b}.

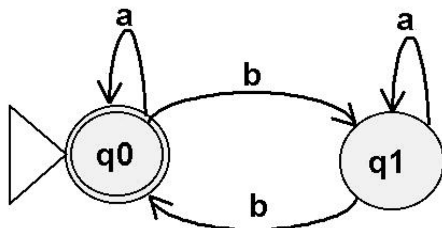

Figura 3: AFD para palavras com número par de b's

```
// Exemplo de Utilização das classes.
// AFD reconhecedor de palavras com número
// par de b's.  Alfabeto ={a, b}.

import java.io.*;

public class ExemploAFD {
   public static void main(String args[]) {
       Automato meuAutomato =
                        new Automato();

       String simbolo = "abababb";
       Transicao trans;
       int origem = 0;
       Estado destino;

       //criação os estados
       meuAutomato.defineEstado(0);
       meuAutomato.defineEstado(1);
       //definição do estado Inicial
       meuAutomato.defineEstadoInicial(0);

       //definição do estado Final
       meuAutomato.defineEstadoFinal(0);
       //criação das Transições
       meuAutomato.defineTransicao(0, 0, "a");
       meuAutomato.defineTransicao(0, 1, "b");
       meuAutomato.defineTransicao(1, 1, "a");
       meuAutomato.defineTransicao(1, 0, "b");
       int i =0;
      System.out.println("\nAFD reconhecedor"+
         " de palavras com número par de b's");
      System.out.println("Verifica a entrada "+
                                simbolo + "\n");

       while (i < simbolo.length())  {
          // O método retornaTransicao retorna
          // a transição que tem como origem e
          // símbolo os parametros passados.
         trans  = meuAutomato.retornaTransicao(
               origem, "" + simbolo.charAt(i));

          // O método retornaEstadoDestino
          // retorna o Estado de Destino da
          // transição trans.
        destino = trans.retornaEstadoDestino();

          // O método retornaID retorna o
          // identificador do Estado destino.
          origem = destino.retornaID();

          System.out.println("Leu o símbolo " +
        simbolo.charAt(i) + " foi pro estado "+
              origem);
           i ++;
       }// Fim while
       // O método eEstadoFinal retorna true se
       // origem for o identificador de um
       // estado final.
       if(meuAutomato.eEstadoFinal(origem))
          System.out.println("\nA entrada " +
                  simbolo +" foi aceita!!!\n");
       else
          System.out.println("\nA entrada " +
               simbolo +" foi rejeitada!!!\n");
   }// Fim main
}// Fim ExemploAFD
```

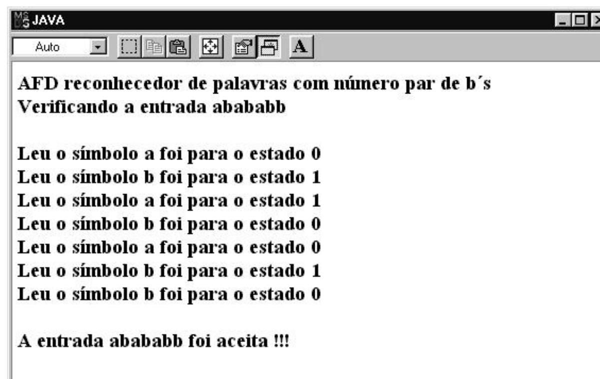

Figura 4: Tela de saída do ExemploAFD

5 Conclusões

A utilização de AFs em aplicações práticas pode ser tediosa, principalmente quando esses AFs são muito grandes. A hierarquia de classes implementada irá trazer grande auxílio para desenvolvedores de aplicações que utilizam AFs, pois tornam mais simples sua manipulação. Nas fases de teste, diversos exemplos de problemas envolvendo autômatos foram implementados, e a experiência confirmou que a utilização da hierarquia de classes proposta facilitou em muito o tratamento e a manipulação dos algoritmos.

No decorrer deste artigo, na descrição da interface das classes, por razões didáticas, somente os métodos mais utilizados foram apresentados, mas as classes também implementam outros métodos que podem se fazer necessários em outras aplicações. A classe Autômato, por exemplo, ainda oferece os métodos:

```
public boolean eEstadoFinal(int id)

public Estado retornaEstadoInicial()
```

Esses métodos podem ser utilizados para pesquisar se um estado é final e para identificar qual o estado inicial, respectivamente. O significado de cada um dos métodos implementados é óbvio, o que torna simples a compreensão e agradável a utilização da estrutura.

Uma documentação detalhada está sendo finalizada, e juntamente com as classes objeto do presente artigo, será disponibilizada em breve no serviço SourceForge (http://www.sourceforge.net). Atualmente, as classes podem ser obtidas em:
http://comp.uniformg.edu.br/~ksoares/classesautomato.zip.

O código está sob a licença de software livre GNU GPL.

Já foi citado anteriormente que esse trabalho faz parte de outro trabalho maior, sendo um dos objetivos a implementação de um sistema para resolução de problemas envolvendo autômatos. Desta forma, para o futuro, serão integrados os módulos responsáveis pelo tratamento da interface do simulador e do editor, além dos algoritmos anteriormente relacionados. Em um período próximo, será criado um site para disponibilizar o sistema e uma coletânea de material didático que poderá ser utilizado em cursos de graduação de Ciência da Computação.

Se consideradas as dificuldades que alunos, por vezes, enfrentam em disciplinas de teoria da computação e a necessidade de material que possa ser utilizado no processo de ensino dessas ferramentas, o presente trabalho pode ser considerado uma contribuição interessante.

6 Referências

[1] AHO, Alfred V.; SETHL, Ravi & ULLMAN, Jefrey D. Compiladores princípios, técnicas e ferramentas, 1995.

[2] HORSTMANN, Cay S. & CORNELL, Gary. Core Java 2 - Recursos Avançados, 2003.

[3] MENEZES, Paulo B. Linguagens formais e autômatos, 2000.

[4] ULLMAN, Jefrey D.; HOPCROFT, Jonh E. & MONTWANI, Rajeev. Introdução à Teoria dos Autômatos, Linguagens e Computação, 2002.

[5] SIPSER, M. Introduction of the Theory of Computation, PWS Co., 2004