**1° Laboratório de CTC-34 - Simulação de AF Bidirecional**

Autores: **Luciano Holanda Gomes** & **Felipe Tuyama de Faria Barbosa**

**1. Introdução**

O projeto consiste na simulação do funcionamento passo a passo de um Autômato Finito Bidirecional (2AF), a partir das especificações contidas em um arquivo texto implementada em Python 2.7. O arquivo de entrada foi montado em formato amigável pra ser facilmente preenchido. A partir dessa entrada são demonstradas todas as transições e no final é indicado se a cadeia foi aceita ou rejeitada.

**2. Metodologia de Implementação**

A implementação do simulador se divide em duas tarefas principais: a leitura do arquivo de texto (.txt) contendo as especificações do 2AF e a subsequente simulação de seu funcionamento, imprimindo na tela passo a passo a configuração instantânea do 2AF.

**2.1) Leitura**

O arquivo texto se divide em blocos ordenados de informação, delimitados por ‘#’:

|  |  |  |  |
| --- | --- | --- | --- |
| Dado | Símbolo | Tipo de dado | Disposição |
| Alfabeto da Linguagem |  | Lista | Um elemento da lista em  cada linha do bloco.  ‘Tabs’ e ‘Espaços’ ignorados. |
| Estados do 2AF |  | Lista |
| Estados de Aceitação |  | Lista |
| Função de Transição |  | Matriz (TFE) | Colunas separadas por Tabs, Linhas separadas por Linhas.  (1ª coluna começa após Tab) |
|  | Matriz (Tdir) |
| Cadeia processada |  | String | Informação em String. |

**Tabela de informações da especificação do 2AF.**

Um exemplo de especificação pode ser encontrado comentado em anexo. Note que qualquer informação fora dos blocos é completamente ignorada durante a leitura, o que permite facilmente a adição de comentários.

Note que as matrizes referentes à função de transição funcionam como funções Lookup Table. Ou seja, dado o estado atual *q* (linhas da matriz) e o símbolo lido (colunas da matriz), temos na matriz ‘*TFE’* (Tabela de Fluxo de Estados) o próximo estado, enquanto na outra ‘*Tdir’* (Tabela de Direções) temos a direção em que o “cabeçote de leitura” deve se mover para prosseguir a leitura da cadeia u.

**2.2) Simulação**

A simulação é considerada o *kernel* do programa, consistindo basicamente em uma iteração sobre os símbolos da cadeia, enquanto houver leitura a ser realizada (ou até a detecção de um loop infinito). Neste processo, o estado atual do 2AF é atualizado, mediante à aplicação da função de transição sobre o estado atual e o símbolo lido da cadeia.

Note que é preciso empregar uma variável *Pointer* para guardar a posição do “cabeçote de leitura” na cadeia. Esse *Pointer* será incrementado/decrementado sempre que a matriz *Tdir* contiver o caractere ‘R’/’L’, respectivamente. A simulação será finalizada quando o seu valor atingir o comprimento da cadeia lida.

Outro modo de se encerrar a computação é detectando um loop infinito, que ocorre sempre que o sistema repetir uma dada configuração instantânea em seu processamento (pois o AF é determinístico, de modo que ele está fadado a retornar a esta configuração incontáveis vezes).

Essa situação foi implementada inserindo um novo campo (uma nova matriz TPT - Tabela de Posição de Transição) de saída para a função de transição, que associará cada transição à posição da cadeia (valor de Pointer) em que ela ocorre.

Matrizes da Função de transição: TFE, Tdir, TPT

Se a tríplice for igual em duas iterações distintas (basta comparar o valor atual de Pointer com a informação na TPT), temos repetição da configuração instantânea do AF, que incorrerá em loop infinito. A computação é então abortada e a cadeia é rejeitada.

O algoritmo em pseudo-código da simulação pode ser descrito como:

|  |
| --- |
| * **Enquanto** Pointer < *length* ( cadeia ) |
| > *Print* configuração instantânea do AF. |
| > Símbolo cadeia[Pointer] |
| > Pointer++ \ Pointer--, conforme Tdir[ q ][] |
| > Próximo estado TFE[ q ][] |
| > Checa loop infinito - verifica se Pointer = TPT[ q ][] |
| > TPT[q][] Pointer |
| > *return* ( |

**3. Metodologia de Teste**

Foram montados no total 4 casos de teste onde tentamos simular o funcionamento normal e os casos que poderiam causar problema. Sendo assim os testes foram para os seguintes casos:

1. Aceitação: Caso de teste para aceitação normal. Com um AF que aceita a cadeia que foi colocada
2. Rejeição: Caso de teste para reijeição normal. Com um AF que rejeita a cadeia que foi colocada
3. Cadeia Vazia: Caso com cadeia vazia começando em estado não final.
4. Loop Infinito: Caso em que o AF entra em loop infinito e portanto deve rejeitar a cadeia colocada.

Esses testes serão descritos em mais detalhes a seguir.

**3.1. Teste de Aceitação:**

Foi montado para demonstrar o funcionamento normal do código. O caso escolhido foi:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | 0 |  |
|  | q0 | **q1(Final)** | q2 |

As transições desse AF são dadas pelas seguintes tabelas de transições:

|  |  |  |
| --- | --- | --- |
| **1** | **0** |  |
| q0/R | q1/R |
| q1/R | q2/L |
| q0/R | q2/L |

A cadeia usada foi: 101001. Desta forma o esperado seria que o AF terminasse no estado q1 portanto a cadeia seria aceita. Rodando o código foi encontrado que realmente ela foi aceita. (Ver resultado em 3.5).

**3.2 Teste de rejeição**  
 Foi montado para demonstrar o funcionamento normal do código. O caso escolhido foi:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | 0 |  |
|  | q0 | q1 | **q2 (Final)** |

As transições desse AF são dadas pelas seguintes tabelas de transições

|  |  |  |
| --- | --- | --- |
| **1** | **0** |  |
| q0/R | q1/R |
| q1/R | q2/L |
| q0/R | q2/L |

A cadeia usada foi: 101001. Desta forma o esperado seria que o AF terminasse no estado q1 portanto a cadeia seria rejeitada. Rodando o código foi encontrado que realmente ela foi rejeitada. (Ver resultado em 3.5).

**3.3 Teste de Cadeia Vazia**

Utilizamos o mesmo AF do teste 3.2 porém com uma cadeia vazia. Desta forma o esperado seria que o AF terminasse no estado q0 (inicial) e portanto a cadeia seria rejeitada. Rodando o código foi encontrado que realmente ela foi rejeitada. (Ver resultado em 3.5).

**3.4 Teste de Loop Infinito**

Foi montado para demonstrar a rejeição de loops infinitos pelo código. Todos os estados são finais para demonstrar como a cadeia nunca será aceita. O caso escolhido foi:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b |  |  |
|  | **q0 (Final)** | **q1** **(Final)** | **q2 (Final)** | **q3 (Final)** |

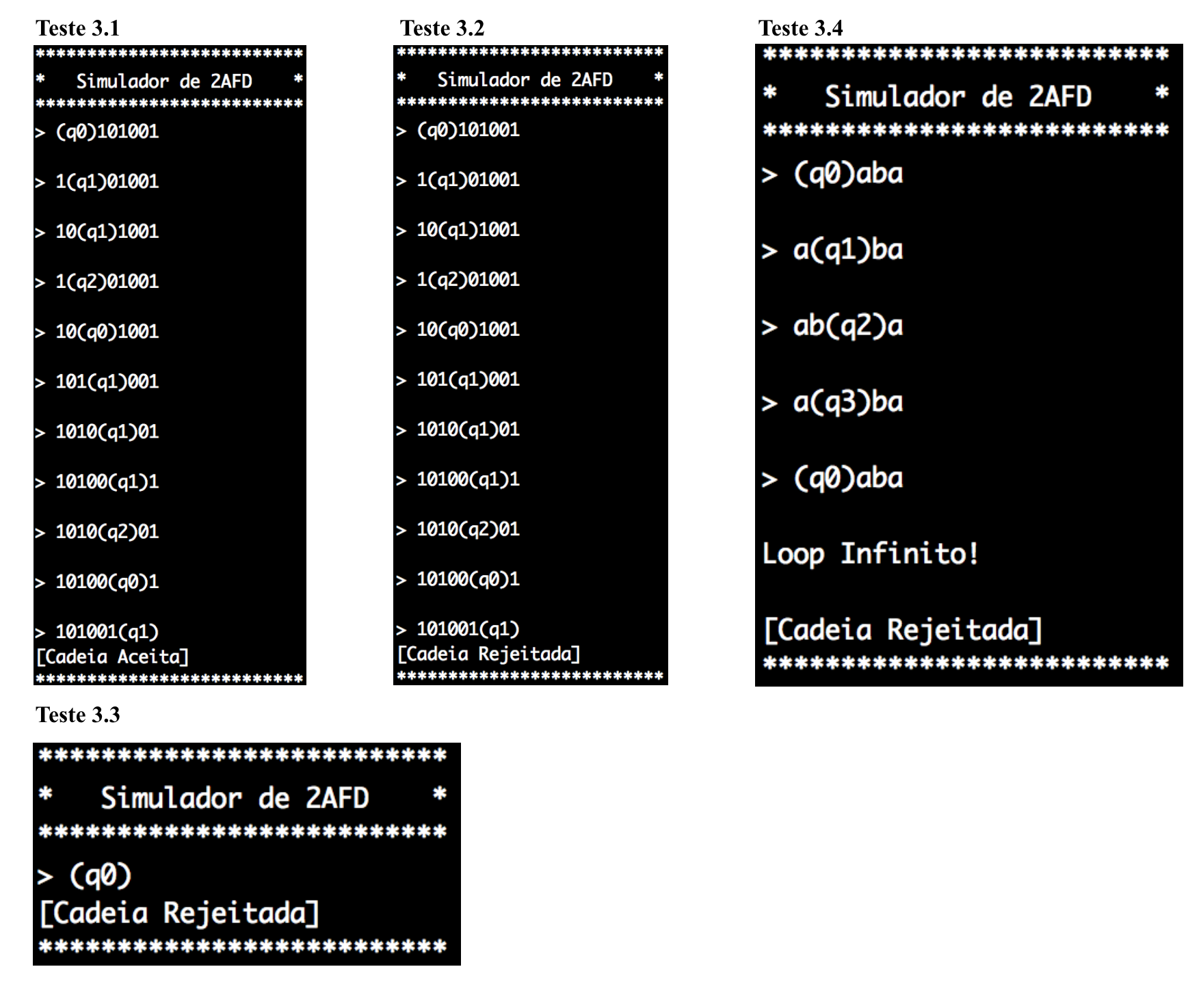
As transições desse AF são dadas pelas seguintes tabelas de transições:

|  |  |  |
| --- | --- | --- |
| **a** | **b** |  |
| **q1/R** | q0/R |
| q0/R | **q2/R** |
| **q3/L** | q2/R |
| q3/L | **q0/L** |

A cadeia usada foi: aba. Sendo assim, esperamos que ocorra um loop infinito e portanto, a cadeia deve ser rejeitada. Esse comportamento foi identificado e rodando o código foi encontrado que realmente ela foi rejeitada. (Ver resultado em 3.5).

**3.5 Resultado de testes**

Todos os testes se comportaram como esperado, como pode ser verificado a seguir:



**4. Conclusão**

Foi possível observar o comportamento de um AF bidirecional e sua progressão em uma cadeia. A implementação de AFs bidirecionais deve sempre tomar cuidado com loops infinitos que podem causar um comportamente não esperado no código e esses casos devem ser tratados, bem como a interação com cadeias vazias.

Essas máquinas são facilmente implementáveis em linguagens Turing completas, como Python, devido a serem Máquinas de Turing, pois elas atendem à *Turing Completude* tendo um comportamento em que regras seguidas em sequência sobre dados arbitrários podem produzir o resultado de qualquer cálculo, ou seja, podem implementar qualquer Máquina de Turing de fita única.

**5. Bibliografia**

Brainerd, W.S., (1974), *Theory of Computation*, Wiley. [ISBN 0-471-09585-0](https://pt.wikipedia.org/wiki/Especial:Fontes_de_livros/0471095850)<http://technology.newscientist.com/article/dn12826-simplest-universal-computer-wins-student-25000.html> [Simplest 'universal computer' wins student $25,000 by Jim Giles, *New Scientist*, October 24, 2007.](https://pt.wikipedia.org/wiki/Especial:Fontes_de_livros/0471095850)

[*The Universal Turing Machine: A Half-Century Survey* (1995), ed.[Rolf Herken, Springer Verlag.](https://pt.wikipedia.org/wiki/Especial:Fontes_de_livros/0471095850) [ISBN 3-211-82637-8](https://pt.wikipedia.org/wiki/Especial:Fontes_de_livros/3211826378)

Ferramenta JFLAP 7.0

**6. Anexos**

**6.1 Código: {2AFD.py}**

|  |
| --- |
| # -\*- coding: cp1252 -\*-  #  # Simulador de 2AFD  #  # Autores: Felipe Tuyama & Luciano Holanda  import sys  # Remove da string ‘Tabs’ e ‘Blankspaces’.  def wash(saida):  saida = saida.replace('\t','')  saida = saida.replace(' ','')  return saida  # Remove da string o ‘\n’.  def read():  return arquivo.readline().rstrip()    # Retorna próxima informação em ‘reader’, referente  # à linha da matriz, que virá após um Tab.  # Encurta ‘reader’ descartando informação retornada.  def nextData():  global reader  begin = 1  while begin < len(reader) and reader[begin] != '\t':  begin += 1  info = reader[1:begin]  reader = reader[begin:len(reader)]  return info    # Leitura da Especificação do 2AFD: (Comentários):  # Para cada bloco de informação do arquivo .txt:  # Procura o início do bloco, começando em ‘#’  # Enquanto o bloco não acabar em ‘#’, faz a leitura de:  # - Lista de símbolos do alfabeto.  # - Lista de estados & estados de aceitação.  # - Matrizes TFE & Tdir, inicializa TPT com ‘0’s  # - String relativa à cadeia a ser processada.  def read2AFD():  global reader  for i in range(0, 6):  reader = ""  while reader != "#":  reader = read()  reader = read()  while reader != "#":  if i == 0: alphabet.append(wash(reader))  elif i == 1: states.append(wash(reader))  elif i == 2: accept.append(wash(reader))  elif i == 3:  TFE.append([])  for a in range(0, len(alphabet)):  TFE[len(TFE)-1].append(nextData())  elif i == 4:  Tdir.append([])  TPT.append([])  for a in range(0, len(alphabet)):  Tdir[len(Tdir)-1].append(nextData())  TPT[len(TPT)-1].append(0)  elif i == 5:  alphabet.append(wash(reader))  reader = read()    # Simulação da 2AFD: (Comentários):  # Inicializa estado **q** e pointer **stt** em ‘0’  # **Enquanto** stt não tiver percorrido toda a cadeia:  # **Print** configuração instantânea do AF.  # **Get** índice do símbolo em cadeia[pointer]  # Pointer++\Pointer-**- conforme** Tdir[q][a]  # **Update** estado q = TFE[q][a]  # **If** loop infinito (pointer = TPT[q][a])  # **Print** configuração instantânea do AF.  # **return** false - cadeia não aceita  # **Else** (implícito) atualiza TPT[q][a]  # **Wait** tecla do usuário  # **Print** configuração instantânea do AF.  # **return** Aceitação (se o estado atual é de aceitação)  def simulate():  q = stt = 0  while stt < len(u):  print "> "+u[0:stt]+"("+states[q]+")"+u[stt:len(u)]  alpha = alphabet.index(u[stt])  if (Tdir[q][alpha]=="R"): stt += 1  elif (Tdir[q][alpha]=="L"): stt -= 1  q = states.index(TFE[q][alpha])  if(TPT[q][alpha] == stt):  print "\n> "+u[0:stt]+"("+states[q]+")"+u[stt:len(u)]  print "\nLoop Infinito!\n"  return False  TPT[q][alpha] = stt  sys.stdin.read(1)  print "> "+u[0:stt]+"("+states[q]+")"+u[stt:len(u)]  return (states[q] in accept)  # Dados relativos ao nosso 2AFD:  alphabet = [] # Alfabeto de entradas  states = [] # Estados do 2AFD  accept = [] # Estados de aceitação  TFE = [] # (Tabela de Fluxo de Estados)  TPT = [] # (Tabela de posição de transição)  Tdir = [] # (Tabela de direções)  # Rotina main(), centralizando programa.  print "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"  print "\* Simulador de 2AFD \*"  print "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"  arquivo = open('2AFD.txt', 'r')  read2AFD()  u = alphabet.pop()  result = simulate()  if result: print "[Cadeia Aceita]"  else: print "[Cadeia Rejeitada]"  print "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" |
|  |

**6.2 Exemplo de arquivo de entrada: {2AFD.txt}**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Simulador de 2AFD \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Notação:

-> Para a Leitura das Listas:

--------------------------------------------------

-> Toda a informação deve estar disposta nos blocos

delimitados por ‘#’s, um dado por linha.

-> Possíveis (espaço/tabs) nos dados serão ignorados.

-> O primeiro estado informado é o inicial.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Alfabeto \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

a

b

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estados \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q0

q1

q2

q3

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Estados Finais \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

q0

q1

q2

q3

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Função de Transição \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Notação:

-> Para a Leitura das Matrizes:

--------------------------------------------------

-> Na Função de Transição, a tabela terá suas colunas

espaçadas por ‘Tab’.

-> A primeira coluna da matriz é precedida por ‘Tab’

-> A 1ª matriz é relativa aos estados resultantes da transição.

-> A 2ª matriz informa a direção que o Pointer deve se mover após a transição: “Right” ou “Left”.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

a b

#

q1 q0

q0 q2

q3 q2

q3 q0

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

a b

#

R R

R R

L R

L L

#

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Cadeia a ser lida \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Notação:

-> Para a Leitura da Cadeia

--------------------------------------------------

-> A cadeia deve ser informada em uma String, em uma única linha.

-> Cada caractere da String deve ser um símbolo do alfabeto dado.

#

aba

#