Definição e Tradução de Máquina de Post

Arthur Jardim
Gabriel Velasco
Marcelo Cordeiro
Rodrigo Segui

1. Definição de uma Máquina de Post. É uma sétupla $MP = (K, \Sigma, \Delta, L, E, s, H)$ onde:

K é um conjunto de estados finitos.

Σ é o alfabeto finito de símbolos.

Δ é a relação de transição definida como $\Delta : T\ x\ \Gamma$ tal que T é pode pertencer a E ou L.

L são as leituras realizadas na forma de $(X \leftarrow Ler(X))$, onde X representa o conteúdo da fila onde ocorrerá a leitura e destruição (leitura destrutiva) por algum símbolo $\sigma\varepsilon\Sigma$;

E são as escritas definidas como o processo de atribuição que é explicitado por $X \leftarrow XS,$ onde S é $\delta\varepsilon\Sigma \cup \{\#\}$ e este é concatenado à direita da variável X ao fim da fila.

s é o estado inicial.

$H \subseteq K$ é o conjunto de estados de parada, no qual pode ser de aceitação ou rejeição.

2. Definição de uma linguagem para Máquina de Post reconhecida por um analisador sintático

{a, b, #, U} → um alfabeto qualquer
(9, 2, 2) → (número de leituras, número de escritas, número de decisões)
8 → número de estados da máquina de post(incluindo 1 de rejeição e 1 de aceitação)
→ conjunto de transições definidos pelo estado inicial, símbolo (a ser escrito ou lido), tipo de operação(escrita, leitura,decisão)
{
(0, #, 1, inicio),
(1, a, 2, leitura),
(1, b, 4, leitura),
(1, #, 5, leitura),
(4, U, 4, decisao),
(5, U, 5, decisao),
(2, a, 3, leitura),
(3, a, 2, escrita),
(2, #, 4, leitura),

(2, b, 6, leitura),
(6, a, 4, leitura),
}

3. Definição da descrição da linguagem da Máquina de Post através da notação de Backus-Naur

$$< MP >::= \ < transição >$$
$$< transição >::= \ < leituras > | < escritas >$$
$$< leituras >::= \ (< estado >, < simbolo >, < estado >, < decisao >)$$
$$< decisao >::= \ (< aceita > | < rejeita >)$$
$$< escritas >::= \ (< estado >, < simbolo >, < estado >)$$

4. Implementação da tradução para Máquina de Turing

Primeiramente, foi realizado a leitura do arquivo de entrada com a descrição da Máquina de Post explicitada no tópico 2 e foi criado listas para armazenar o alfabeto e suas transições.

Na construção do algoritmo de tradução optou-se por criar a Máquina de Turing de saída, baseada no processo de funcionamento da Máquina de Post. Na Máquina de Post é utilizada uma fila com o carácter "#" que marca o final dessa fila, nas leituras o símbolo contido na fila é excluído e nas escritas o símbolo é apagado e escrito ao final da fila. Na Máquina de Turing de saída, a fita infinita é utilizada como uma fila usando o mesmo carácter "#" para marcar o final, assim foram criados processos para cada situação. As situações são as seguintes: início, leitura e escrita. Foi utilizada a linguagem de programação *Python* para o desenvolvimento da tradução.

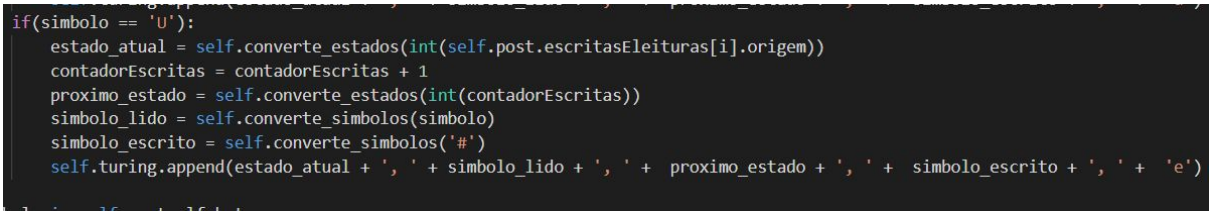
No início (Figuras 1, 2 e 3) é necessário passar por toda palavra e colocar o carácter # no final e voltar para o início da palavra, é utilizado sempre no início de qualquer tradução no sistema. Refere-se ao $X \leftarrow X\#$ da Máquina de Post.

Figura 1: Implementação da tradução "Início".

```
elif('inicio' == (str(self.post.escritasEleituras[i].tipo))):
    contadorEscritas = contadorEscritas + 1
    estado_atual = self.converte_estados(int(contadorEscritas))
    proximo_estado = self.converte_estados(int(self.post.escritasEleituras[i].origem))
    simbolo_lido = self.converte_simbolos('U')
    simbolo_escrito = self.converte_simbolos('U')
    self.turing.append(estado_atual + ', ' + simbolo_lido + ', ' +  proximo_estado + ', ' +  simbolo_escrito + ', ' +  'd')
    for simbolo in self.post.alfabeto:
            if(simbolo != '#' and simbolo != 'U'):
                estado_atual = self.converte_estados(int(self.post.escritasEleituras[i].origem))
                proximo_estado = self.converte_estados(int(self.post.escritasEleituras[i].origem))
                simbolo_lido = self.converte_simbolos(simbolo)
                simbolo_escrito = self.converte_simbolos(simbolo)
                self.turing.append(estado_atual + ', ' + simbolo_lido + ', ' +  proximo_estado + ', ' +  simbolo_escrito + ', ' +  'd')
```

Fonte: autores, 2019.

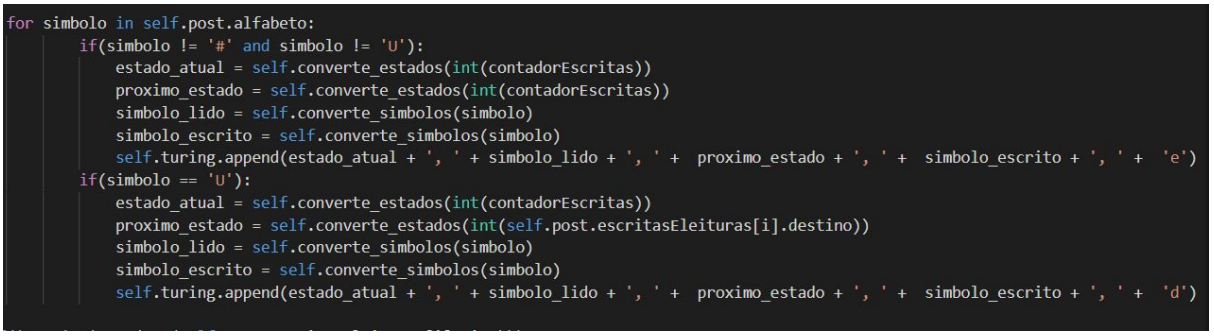
Figura 2: Implementação da tradução “Início”.



```
if(simbolo == 'U'):
    estado_atual = self.converte_estados(int(self.post.escritasEleituras[i].origem))
    contadorEscritas = contadorEscritas + 1
    proximo_estado = self.converte_estados(int(contadorEscritas))
    simbolo_lido = self.converte_simbolos(simbolo)
    simbolo_escrito = self.converte_simbolos('#')
    self.turing.append(estado_atual + ', ' + simbolo_lido + ', ' +  proximo_estado + ', ' +  simbolo_escrito + ', ' +  'e')
```

Fonte: autores, 2019.

Figura 3: Implementação da tradução “Início”.

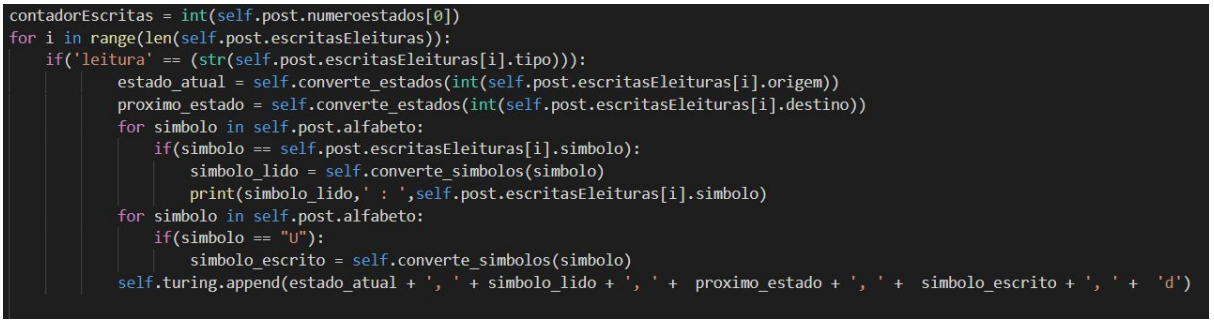


```
for simbolo in self.post.alfabeto:
        if(simbolo != '#' and simbolo != 'U'):
            estado_atual = self.converte_estados(int(contadorEscritas))
            proximo_estado = self.converte_estados(int(contadorEscritas))
            simbolo_lido = self.converte_simbolos(simbolo)
            simbolo_escrito = self.converte_simbolos(simbolo)
            self.turing.append(estado_atual + ', ' + simbolo_lido + ', ' +  proximo_estado + ', ' +  simbolo_escrito + ', ' +  'e')
        if(simbolo == 'U'):
            estado_atual = self.converte_estados(int(contadorEscritas))
            proximo_estado = self.converte_estados(int(self.post.escritasEleituras[i].destino))
            simbolo_lido = self.converte_simbolos(simbolo)
            simbolo_escrito = self.converte_simbolos(simbolo)
            self.turing.append(estado_atual + ', ' + simbolo_lido + ', ' +  proximo_estado + ', ' +  simbolo_escrito + ', ' +  'd')
```

Fonte: autores, 2019.

As leituras (Figura 4) na Máquina de Post são simuladas como transições, onde é lido o símbolo da leitura, escrito um símbolo branco e o deslocamento para a direita na Máquina de Turing. Refere-se ao $ler\,X$ da Máquina de Post.

Figura 4: Implementação da tradução “Leituras”.



```
contadorEscritas = int(self.post.numeroestados[0])
for i in range(len(self.post.escritasEleituras)):
    if('leitura' == (str(self.post.escritasEleituras[i].tipo))):
            estado_atual = self.converte_estados(int(self.post.escritasEleituras[i].origem))
            proximo_estado = self.converte_estados(int(self.post.escritasEleituras[i].destino))
            for simbolo in self.post.alfabeto:
                if(simbolo == self.post.escritasEleituras[i].simbolo):
                    simbolo_lido = self.converte_simbolos(simbolo)
                    print(simbolo_lido,' : ',self.post.escritasEleituras[i].simbolo)
            for simbolo in self.post.alfabeto:
                if(simbolo == "U"):
                    simbolo_escrito = self.converte_simbolos(simbolo)
            self.turing.append(estado_atual + ', ' + simbolo_lido + ', ' +  proximo_estado + ', ' +  simbolo_escrito + ', ' +  'd')
```

Fonte: autores, 2019.

Nas escritas (Figuras 5 e 6) é necessário criar um estado a mais do que os já criados nas transições de leituras, assim é preciso passar por toda a palavra na fita e adicionar o valor que vai ser escrito ao final e depois voltar para o início da palavra. Refere-se ao $X \leftarrow XS$ onde S são os símbolos do alfabeto, exceto $\#$ e U que são símbolos que fazem parte de um alfabeto auxiliar das máquinas.

Figura 5: Implementação da tradução “Escritas”.

```
elif('escrita' == (str(self.post.escritasEleituras[i].tipo))):
    #x<=xa || x<=xb
    simboloEscrita = self.post.escritasEleituras[i].simbolo
    for simbolo in self.post.alfabeto:
            if(simbolo != 'U'):
                estado_atual = self.converte_estados(int(self.post.escritasEleituras[i].origem))
                proximo_estado = self.converte_estados(int(self.post.escritasEleituras[i].origem))
                simbolo_lido = self.converte_simbolos(simbolo)
                simbolo_escrito = self.converte_simbolos(simbolo)
                self.turing.append(estado_atual + ', ' + simbolo_lido + ', ' +  proximo_estado + ', ' +  simbolo_escrito + ', ' +  'd')
            if(simbolo == 'U'):
                estado_atual = self.converte_estados(int(self.post.escritasEleituras[i].origem))
                contadorEscritas = contadorEscritas + 1
                proximo_estado = self.converte_estados(int(contadorEscritas))
                simbolo_lido = self.converte_simbolos(simbolo)
                simbolo_escrito = self.converte_simbolos(simboloEscrita)
                self.turing.append(estado_atual + ', ' + simbolo_lido + ', ' +  proximo_estado + ', ' +  simbolo_escrito + ', ' +  'e')
```

Fonte: autores, 2019.

Figura 6: Implementação da tradução “Escritas”.

```
for simbolo in self.post.alfabeto:
        if(simbolo != 'U'):
            estado_atual = self.converte_estados(int(contadorEscritas))
            proximo_estado = self.converte_estados(int(contadorEscritas))
            simbolo_lido = self.converte_simbolos(simbolo)
            simbolo_escrito = self.converte_simbolos(simbolo)
            self.turing.append(estado_atual + ', ' + simbolo_lido + ', ' +  proximo_estado + ', ' +  simbolo_escrito + ', ' +  'e')
        if(simbolo == 'U'):
            estado_atual = self.converte_estados(int(contadorEscritas))
            proximo_estado = self.converte_estados(int(self.post.escritasEleituras[i].destino))
            simbolo_lido = self.converte_simbolos(simbolo)
            simbolo_escrito = self.converte_simbolos(simbolo)
            self.turing.append(estado_atual + ', ' + simbolo_lido + ', ' +  proximo_estado + ', ' +  simbolo_escrito + ', ' +  'd')
```

Fonte: autores, 2019.

Foram implementadas duas funções para codificar a Máquina de Turing na forma solicitada no trabalho. Na função converte símbolos (Figura 7) temos como parâmetro o símbolo e um dicionário, e retorna o símbolo codificado. Para isso foi implementado um dicionário com os símbolos do alfabeto (chave) e um número real (valor do conteúdo). Cada vez que encontrar o símbolo no dicionário ele armazenará o número real referente na váriavel simbolo_para_codificar, por fim o simbolo_tranformado é a concatenação do marcador inicial “a” com a conversão em binário do valor real referente ao símbolo.

Figura 7: Função de conversão dos símbolos.

```
def converte_simbolos(self, simbolo,dicionario):
    tamanho_Alfabeto = len(self.post.alfabeto)
    for chave in dicionario:
        if(chave == simbolo):
            simbolo_para_codificar = dicionario[chave]
    qnt_simbolos = round(math.log(tamanho_Alfabeto, 2) + 0.5)
    marcador_inicial = "a"
    simbolo_transformado = marcador_inicial + bin(simbolo_para_codificar)[2:].zfill(qnt_simbolos)
    return simbolo_transformado
```

Fonte: autores, 2019.

Na função converte estados (Figura 8) temos como parâmetro o número do estado e como retorno, o estado transformado implementado com a concatenação do marcador inicial “q” e o valor do estado em binário. Também foi utilizada a função *zfill* para concatenar zeros a esquerda.

Figura 8: Função de conversão dos estados.

```
def converte_estados(self,estado):
    num_estado = (int(self.post.numerosLED[1]) + 2 + int(self.post.numeroestados[0]))
    qnt_estados = round(math.log(int(num_estado), 2) + 0.5)
    marcador_inicialEstados = "q"
    estado_transformado = marcador_inicialEstados + bin(estado)[2:].zfill(qnt_estados)
    return estado_transformado
```

Fonte: autores, 2019.

Na função “escreveMaquina” (Figura 9) é realizada a escrita das transições da Máquina de Turing em um arquivo de saida.txt.

Figura 9: Função de gravação no arquivo de saída.

```
def escreveMaquina(self, turing):
    texto = ""
    for i in range(len(turing)):
        texto += turing[i] + "\n"
    return texto
```

Fonte: autores, 2019.

5. Testes

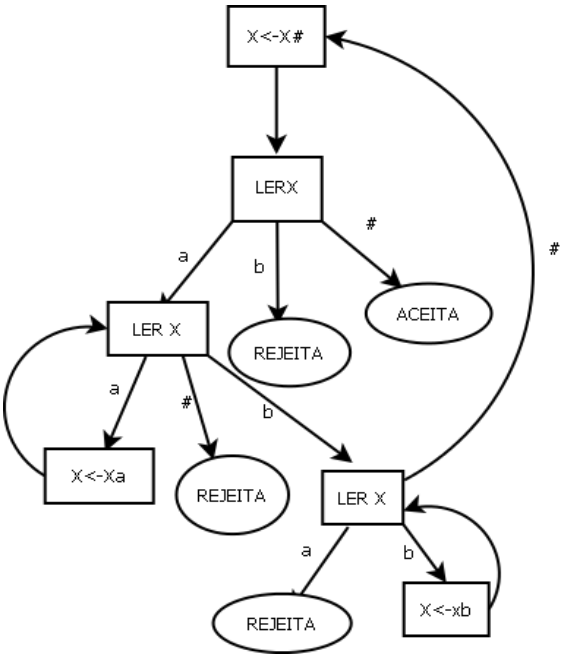
Foram realizados alguns testes, dentre eles: Máquina de Post que reconhece $a^n b^n$ e Máquina de Post que reconhece $mesmo\ número\ de\ 0s\ e\ 1s$. Os testes foram realizados em 4 etapas:

- Etapa 1: escolha de algumas máquinas de post.
- Etapa 2: descrevê-las de acordo com a descrição do tópico 2.
- Etapa 3: após a conversão fazer análise da descrição da máquina de turing gerada no arquivo de saída.
- Etapa 4: desenhar manualmente a máquina gerada e validá-la.

Após a realização dos testes verificou-se que a conversão estava sendo realizada corretamente e de acordo com os pré requisitos do projeto.

5.1 Teste 1: Máquina de Post que reconhece $a^n b^n$

Figura 10 (ETAPA 1): Máquina de Post que reconhece $a^n b^n$



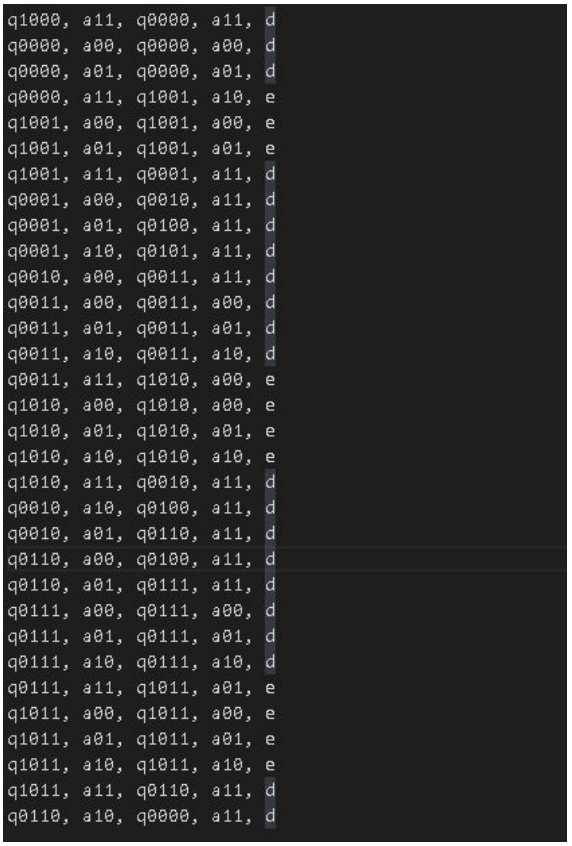
Fonte: autores, 2019.

Figura 11: ETAPA 2, Descrição da Máquina de Post que reconhece $a^n b^n$

```
s > entradas > entrada1
{a, b, #, U}
(9, 2, 2)
8
{
    (0, #, 1, inicio),
    (1, a, 2, leitura),
    (1, b, 4, leitura),
    (1, #, 5, leitura),
    (4, U, 4, decisao),
    (5, U, 5, decisao),
    (2, a, 3, leitura),
    (3, a, 2, escrita),
    (2, #, 4, leitura),
    (2, b, 6, leitura),
    (6, a, 4, leitura),
    (6, b, 7, leitura),
    (7, b, 6, escrita),
    (6, #, 0, leitura)
}
```

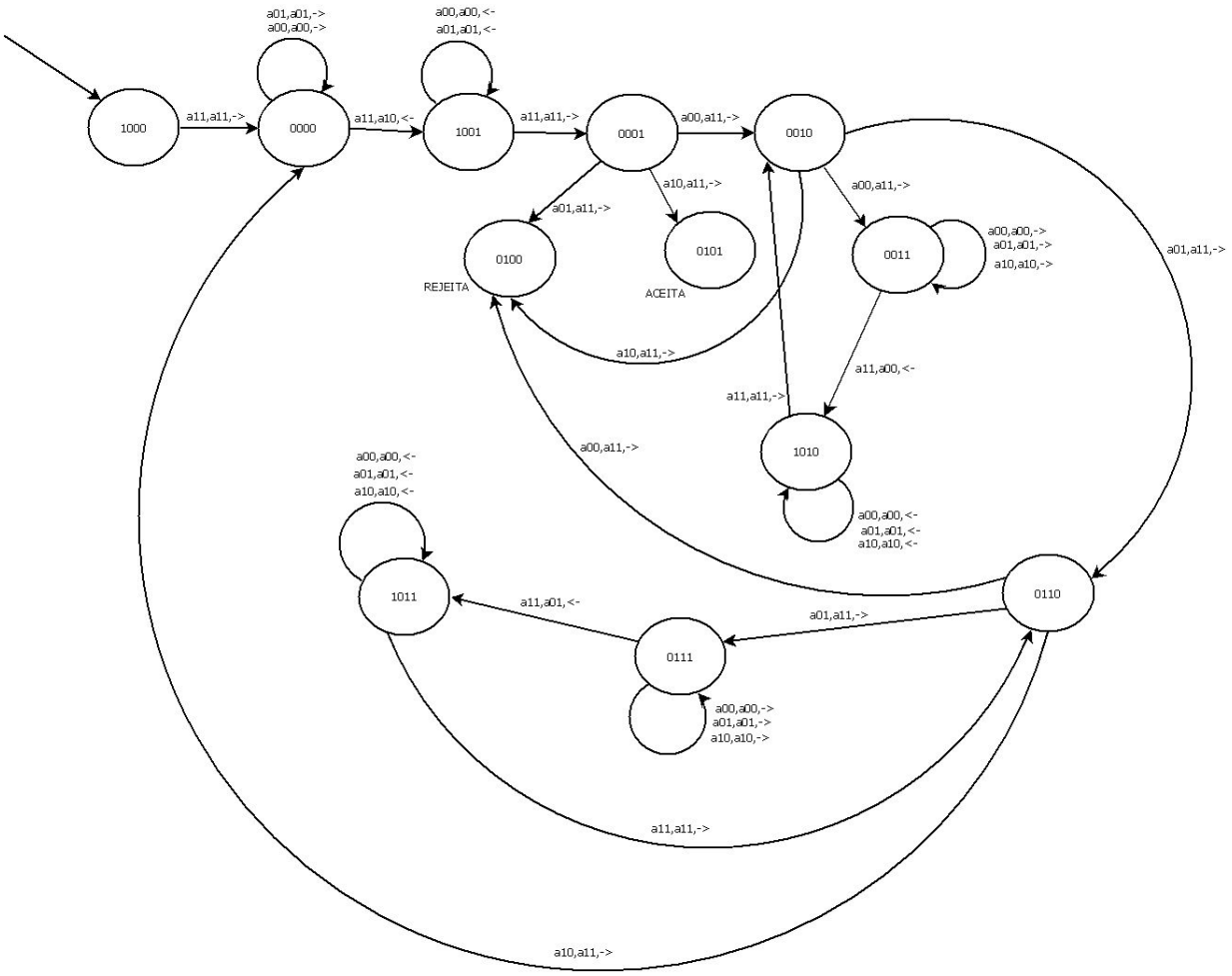
Fonte: autores, 2019.

Figura 12: ETAPA 3, descrição da Máquina de Turing que reconhece $a^n b^n$ gerada após a conversão



Fonte: autores, 2019.

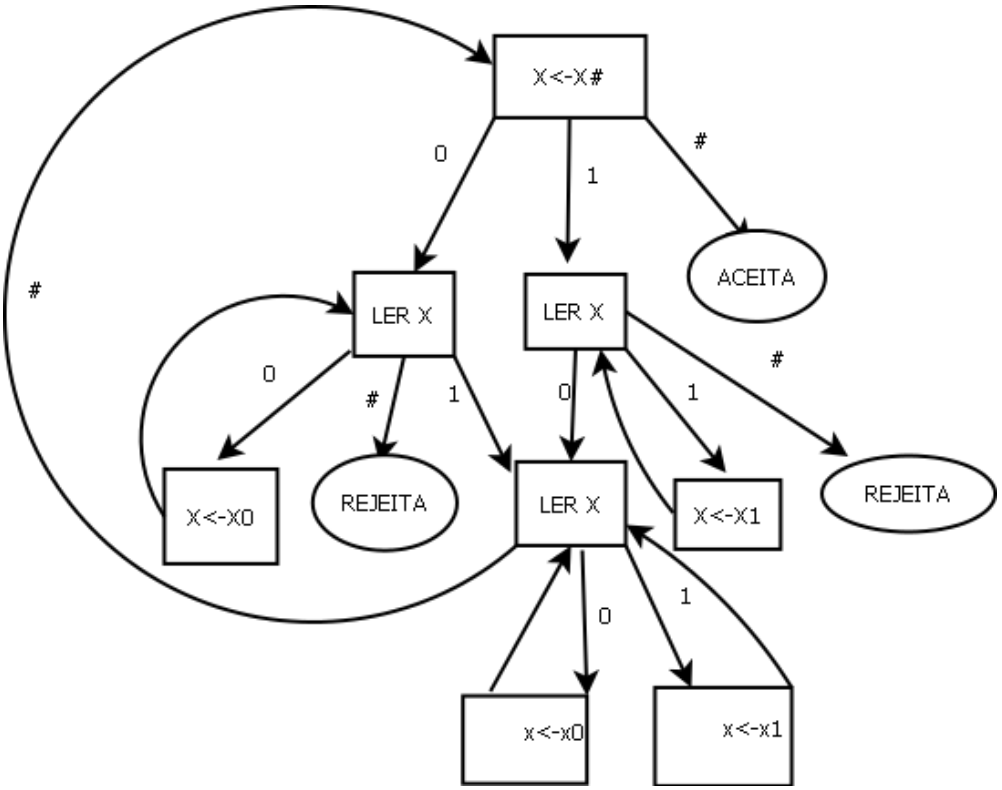
Figura 13: ETAPA 4, Máquina de Turing que reconhece $a^n b^n$ construída manualmente de acordo com a conversão gerada.



Fonte: autores, 2019.

5.2 Teste 2: Máquina de Post que reconhece mesmo número de 0’s e 1’s

Figura 14 : Máquina de Post que reconhece *mesmo número de 0's e 1's*



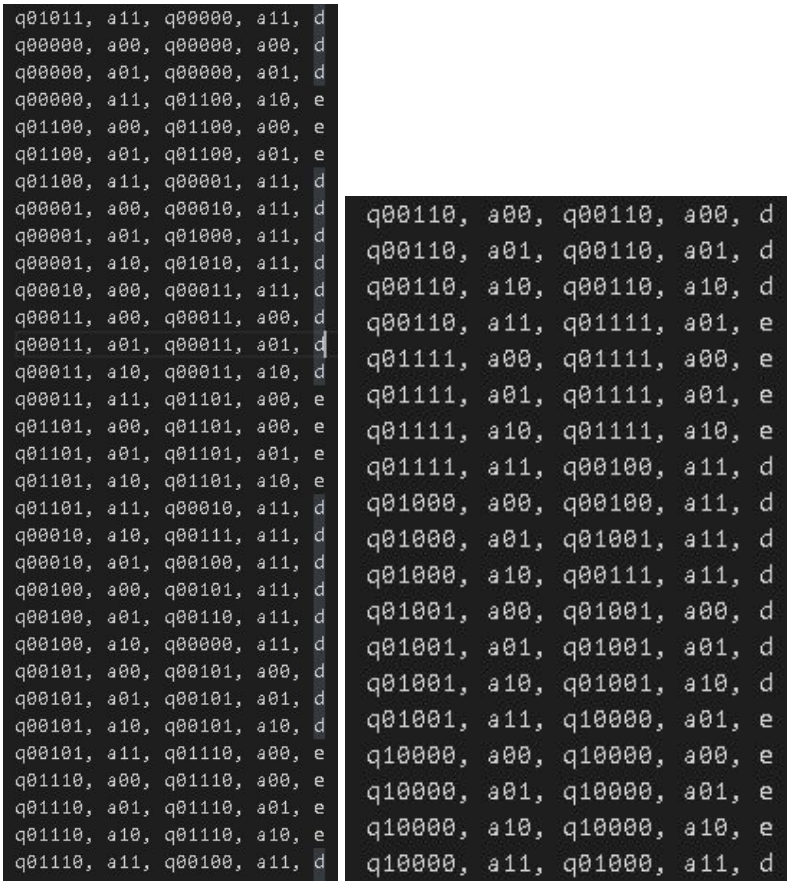
Fonte: autores, 2019.

Figura 15: ETAPA 2, Descrição da Máquina de Post que reconhece *mesmo número de 0's e 1's* .

```
> entradas > entrada2
{0, 1, #, U}
(12, 4, 2)
11
{
    (0, #, 1, inicio),
    (1, 0, 2, leitura),
    (1, 1, 8, leitura),
    (1, #, 10, leitura),
    (10, U, 10, decisao),
    (2, 0, 3, leitura),
    (3, 0, 2, escrita),
    (2, #, 7, leitura),
    (2, 1, 4, leitura),
    (7, U, 7, decisao),
    (4, 0, 5, leitura),
    (4, 1, 6, leitura),
    (4, #, 0, leitura),
    (5, 0, 4, escrita),
    (6, 1, 4, escrita),
    (8, 0, 4, leitura),
    (8, 1, 9, leitura),
    (8, #, 7, leitura),
    (9, 1, 8, escrita)
}
```

Fonte: autores, 2019.

Figura 16 e 17 (ETAPA 3): Descrição da Máquina de Turing que reconhece *mesmo número de 0's e 1's* gerada após a conversão.

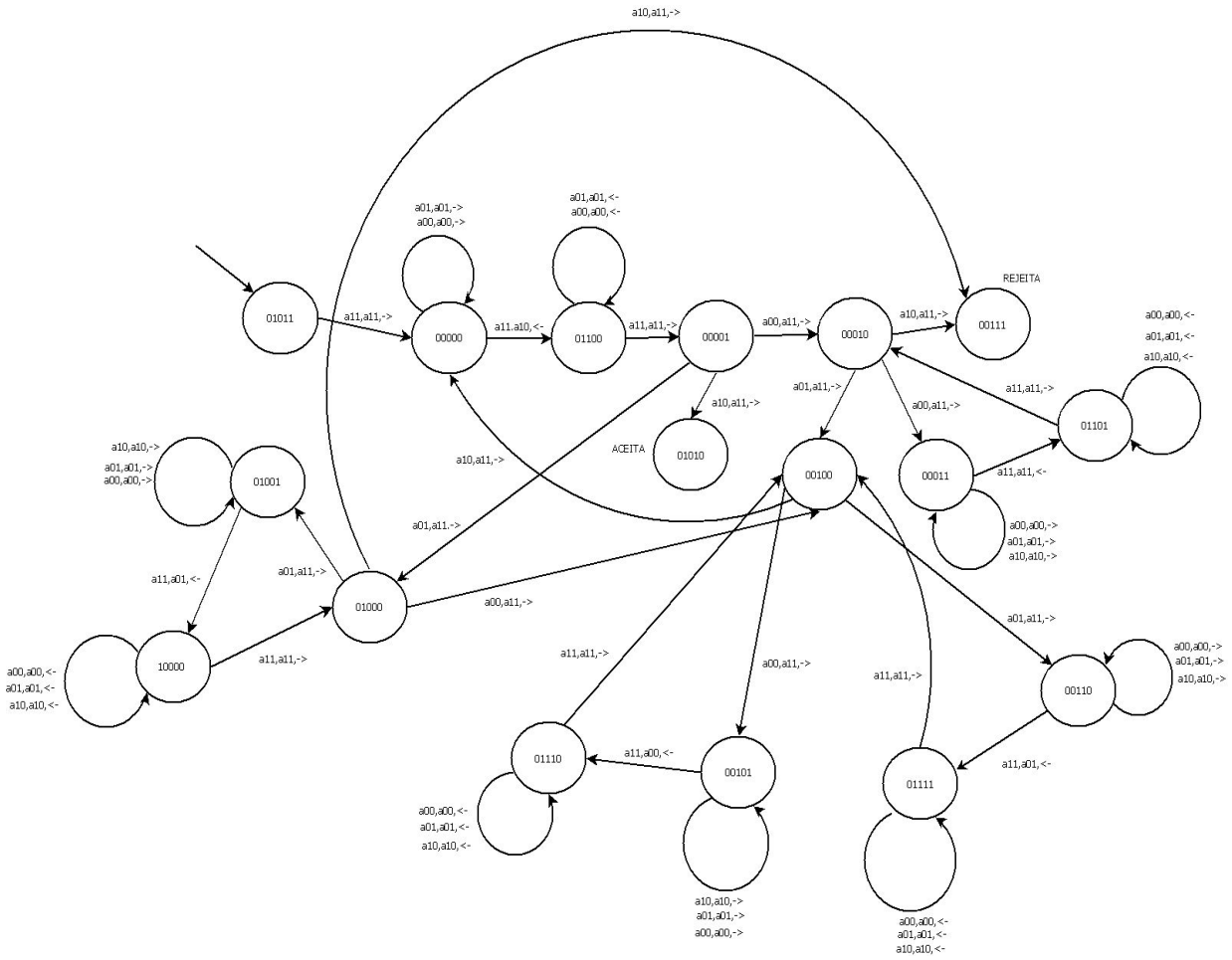


```
q01011, a11, q00000, a11, d
q00000, a00, q00000, a00, d
q00000, a01, q00000, a01, d
q00000, a11, q01100, a10, e
q01100, a00, q01100, a00, e
q01100, a01, q01100, a01, e
q01100, a11, q00001, a11, d
q00001, a00, q00010, a11, d
q00001, a01, q01000, a11, d
q00001, a10, q01010, a11, d
q00010, a00, q00011, a11, d
q00011, a00, q00011, a00, d
q00011, a01, q00011, a01, d
q00011, a10, q00011, a10, d
q00011, a11, q01101, a00, e
q01101, a00, q01101, a00, e
q01101, a01, q01101, a01, e
q01101, a10, q01101, a10, e
q01101, a11, q00010, a11, d
q00010, a10, q00111, a11, d
q00010, a01, q00100, a11, d
q00100, a00, q00101, a11, d
q00100, a01, q00110, a11, d
q00100, a10, q00000, a11, d
q00101, a00, q00101, a00, d
q00101, a01, q00101, a01, d
q00101, a10, q00101, a10, d
q00101, a11, q01110, a00, e
q01110, a00, q01110, a00, e
q01110, a01, q01110, a01, e
q01110, a10, q01110, a10, e
q01110, a11, q00100, a11, d
```

```
q00110, a00, q00110, a00, d
q00110, a01, q00110, a01, d
q00110, a10, q00110, a10, d
q00110, a11, q01111, a01, e
q01111, a00, q01111, a00, e
q01111, a01, q01111, a01, e
q01111, a10, q01111, a10, e
q01111, a11, q00100, a11, d
q01000, a00, q00100, a11, d
q01000, a01, q01001, a11, d
q01000, a10, q00111, a11, d
q01001, a00, q01001, a00, d
q01001, a01, q01001, a01, d
q01001, a10, q01001, a10, d
q01001, a11, q10000, a01, e
q10000, a00, q10000, a00, e
q10000, a01, q10000, a01, e
q10000, a10, q10000, a10, e
q10000, a11, q01000, a11, d
```

Fonte: autores, 2019.

Figura 18: ETAPA 4, Máquina de Turing que reconhece *mesmo número de 0's e 1's* construída manualmente de acordo com a conversão gerada



Fonte: autores, 2019.

6. Manual de Compilação

Para executar o programa (via terminal) é necessário estar no diretório “códigos” deste projeto. O comando para execução é o seguinte: python Main.py entrada1 saida1, onde:

entrada1: deve ser o nome do arquivo texto com a descrição da Máquina de Post a ser convertida.

saída1: deve ser o nome do arquivo onde será gravado a descrição da Máquina de Turing convertida.

Figura 19: comando no terminal para compilação.

```
\GitHub\MaquinaPost-Turing\codigos> python Main.py entrada1 saida1
```

Fonte: autores, 2019.

7. Datas das realizações das tarefas com a lista de temas abordados

Assunto/Data	11/11 à 17/11	18/11 à 24/11	25/11 à 02/12	03/12 à 09/12
Divisão de tarefas	X			
Definição da MP	X			
Definição da Linguagem para a MP		X		
Implementação da tradução		X	X	X
Testes				X
Manual de compilação				X

8. Referências

Copeland, B. Jack, "The Church-Turing Thesis", *The Stanford Encyclopedia of Philosophy* (Spring 2019 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/spr2019/entries/church-turing/>.

Alhazov, A., Kudlek, M., Rogozhin, Y. (2002). *Nine Universal Circular Post Machines*. Computer Science Journal of Moldova., vol 10, no 3(30).

Post, E., L. (1947). *Recursive Unsolvability of a Problem of Thue*. The Journal of Symbolic Logic., vol 12, no 1, pp. 1-11.

Post, Emil L. *A variant of a recursively unsolvable problem*. Bull. Amer. Math. Soc. 52 (1946), no. 4, 264--268. https://projecteuclid.org/euclid.bams/1183507843.

Post, E., L. (1943). *Formal reductions of the general combinatorial decision problem*. American journal of mathematics., vol 65, no 2, pp. 197-215.

Post, E., L. (1936). *Finite Combinatory Processes-Formulation 1*. The Journal of Symbolic Logic., vol 1, no 3, pp 103-105.

Kudlek, M., Rogozhin, Y. (2001). *Small Universal Circular Post Machines*. Computer Science Journal of Moldova., vol 9, no 1(25).

Minsky, M., L. (1961). *Recursive Unsolvability of Post´s Problem of "Tag" and other Topics in Theory of Turing Machines*. Annals of Mathematics Second Series., vol 74, no 3, pp. 437-455.

Alves, D., P. *Equivalência de Máquinas Universais:Demonstração, Análise e Simulação*. 2007. Monografia de Bacharelado - Universidade do Vale do Rio dos Sinos, 2007.