



Implementação de uma maquina de Turing



Engenharia da Computação

Linguagens Formais e Automatos

Antonio Lister Azevedo Sousa
lister.wd@hotmail.com

Prof: THALES LEVI AZEVEDO

Pacelle Henrique Parnaiba
pacelle.henrique@discente.ufma.br

Raphael Camara Sá
raphael.sa@discente.ufma.br

```
self.fingerprints = set()
self.logdups = True
self.debug = debug
self.logger = logging.getLogger()
if path:
    self.file = open(path, "a")
    self.file.seek(0)
    self.fingerprints.update(self.file.read().split())
@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean("debug")
    return cls(job_dir=settings["job_dir"])
def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + "\n")
def request_fingerprint(self, request):
    return request_fingerprint(request)
```

01

Função Grafo





Analisando o código

Grafo estático da Maquina de Turing. Apenas será gerado se as condições forem atendidas.

→ **from graphviz import Digraph:** Usamos a biblioteca Graphviz para gerar os grafos.

→ **Criação de uma instância 'afe':** Dentro da classe, instanciamos **afe** para realizar operações.

→ **afe.attr('node', shape='doublecircle')** define o formato dos nós como círculos duplos.

→ **Estados:** **afe.attr('node', shape='doublecircle')** define o formato dos nós (estados) como círculos duplos para o estado final (qf). Depois, **afe.node('qf')** adiciona o estado qf. Os outros estados (q0, q1, q2, q3) são adicionados como círculos simples.

Analisando o código

```
grafo():
    from graphviz import Digraph
    # Grafo estático da Máquina de Turing. Apenas será gerado se as condições forem atendidas
    afe = Digraph('L={anbrlh}', filename='turing', format='png')
    afe.attr(rankdir='LR', size='10,7')

    # Estados
    afe.attr('node', shape='doublecircle')
    afe.node('qf')
    afe.attr('node', shape='circle')
    afe.node('q0')
    afe.node('q1')
    afe.node('q2')
    afe.node('q3')

    # Transições
    afe.edge('q0', 'q0', label='*', *, D)
    afe.edge('q0', 'q1', label='a, A, D')
    afe.edge('q0', 'q3', label='B, B, D')
    afe.  ('q0', 'q1', 'a, A, D')
    afe. (function) edge: Any ('q0', 'q3', 'B, B, D')
    afe.edge('q1', 'q1', label='B, B, D')
    afe.edge('q1', 'q2', label='b, B, E')
    afe.edge('q2', 'q2', label='a, a, E')
    afe.edge('q2', 'q2', label='B, B, E')
    afe.edge('q2', 'q0', label='A, A, D')
    afe.edge('q3', 'q3', label='B, B, D')
    afe.edge('q3', 'qf', label='_, _, D')

afe.view()
```

Destacamos aqui nesta função, a criação do nosso grafo. Juntamente com seus Estados (**afe.attr('node', shape='doublecircle')**) e Transições.

Aqui também definimos a estética de nosso Grafo, com nosso atributo **afe.attr(rankdir='LR', size='10,7')**, que define a hierarquia e o tamanho.

```
self.fingerprints = set()
self.logduplicates = True
self.debug = debug
self.logger = logging.getLogger()
if path:
    self.file = open(os.path.join(path, 'fingerprint.log'), 'w')
    self.file.seek(0)
    self.fingerprints.update(fingerprint_for_file(self.file))
@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean('debug')
    return cls(job_dir=settings['job_dir'],
               logduplicates=settings.getboolean('logduplicates'),
               fingerprints=settings.getset('fingerprints'),
               request_fingerprint=settings.getset('request_fingerprint'),
               request_seen=settings.getset('request_seen'),
               request_log=settings.getset('request_log'),
               logduplicates=settings.getboolean('logduplicates'),
               debug=debug,
               logger=settings.getlogger('logger'))
```

02

Função Maquina de Turing



Função Maquina de Turing

</> O método `_init_`

inicializa a máquina com a fita, símbolo de espaço vazio, estado inicial, estados finais e transições. Também inicializa a posição da cabeça de leitura/escrita e armazena os movimentos para animação.



O método `movimento`

obtém o símbolo atual sob a cabeça, verifica se há uma transição definida, atualiza o estado e a fita conforme a transição, e armazena o movimento.



O método `animacao`

anima a execução da máquina, limpando o terminal e imprimindo o estado da fita em cada passo com um atraso de 1 segundo.



O método `run`

executa a máquina até alcançar um estado de aceitação, retornando a fita final e o estado final.

```
self.fingerprints = set()
self.logduplicates = True
self.debug = debug
self.logger = logging.getLogger()
if path:
    self.file = open(os.path.join(path, 'fingerprint.log'), 'w')
    self.file.seek(0)
    self.fingerprints.update(fingerprint_for_file(self.file))
@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean('debug')
    return cls(job_dir=settings['job_dir'],
               logduplicates=settings['logduplicates'],
               debug=debug,
               logger=settings['logger'])
def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + '\n')
def request_fingerprint(self, request):
    return request_fingerprint(request)
```

03

Função de transição



Analizando o código

Define as transições da máquina de Turing como um dicionário onde as chaves são tuplas (estado atual, símbolo lido) e os valores são tuplas (próximo estado, símbolo escrito, direção do movimento).

```
# Função de transição: define as regras de transição da máquina
transicoes = {
    ('q0', '*'): ('q0', '*', 'D'), # No estado q0 e ler '*', escreva '*', vá para q1 e move a cabeça para a direita
    ('q0', 'a'): ('q1', 'A', 'D'), # No estado q0 e ler 'a', escreva 'A', vá para q1 e move a cabeça para a direita
    ('q0', 'B'): ('q3', 'B', 'D'), # No estado q0 e ler 'B', escreva 'B', vá para q3 e move a cabeça para a direita
    ('q0', '_'): ('qf', '_', 'D'), # No estado q0 e ler '_', escreva '_', vá para qf (estado de aceitação)
    ('q1', 'a'): ('q1', 'a', 'D'), # No estado q1 e ler 'a', escreva 'a', continue em q1 e move a cabeça para a direita
    ('q1', 'B'): ('q1', 'B', 'D'), # No estado q1 e ler 'B', escreva 'B', continue em q1 e move a cabeça para a direita
    ('q1', 'b'): ('q2', 'B', 'E'), # No estado q1 e ler 'b', escreva 'B', vá para q2 e move a cabeça para a esquerda
    ('q2', 'a'): ('q2', 'a', 'E'), # No estado q2 e ler 'a', escreva 'a', continue em q2 e move a cabeça para a esquerda
    ('q2', 'A'): ('q0', 'A', 'D'), # No estado q2 e ler 'A', escreva 'A', vá para q0 e move a cabeça para a direita
    ('q2', 'B'): ('q2', 'B', 'E'), # No estado q2 e ler 'B', escreva 'B', continue em q2 e move a cabeça para a esquerda
    ('q3', 'B'): ('q3', 'B', 'D'), # No estado q3 e ler 'B', escreva 'B', continue em q3 e move a cabeça para a direita
    ('q3', '_'): ('qf', '_', 'D') # No estado q3 e ler '_', escreva '_', vá para qf (estado de aceitação) e move a cabeça para a direita
}
```

04

Inicialização e execução da máquina

```
self.fingerprints = set()
self.logdups = True
self.debug = debug
self.logger = logging.getLogger()
if path:
    self.file = open(os.path.join(job_dir, path))
    self.file.seek(0)
    self.fingerprints.update(fingerprint_from_file(self.file))

@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean('debug')
    return cls(job_dir=settings['job_dir'],
               logdups=settings.getboolean('logdups'),
               debug=debug)

def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + '\n')

def request_fingerprint(self, request):
    return request_fingerprint(request)
```



Analisando o código

Nesta fase do código, devemos inicializar a fita, símbolo de espaço vazio, estado inicial e estados finais. Nossa máquina de Turing (teste), executa a máquina com teste.run(), anima a execução com teste.animacao(), gera o grafo e imprime o resultado final da fita e o estado final.

```
# Inicializando a máquina de Turing
fita = "*aabb__" # A fita inicial com a palavra a ser processada. São dois espaços finais para visualização da animação!!!
espaço_vazio = '_' # Símbolo em branco
estado_inicial = 'q0' # Estado inicial
estados_finais = {'qf'} # Conjunto de estados de aceitação

# Cria uma instância da máquina de Turing
teste = MaquinaTuring(fita, espaço_vazio, estado_inicial, estados_finais, transicoes)
resultado_fita, estado_final = teste.run() # Executa a máquina

if teste:
    teste.animacao() # Anima a execução da máquina
    grafo()
    print("Resultado final da fita:", resultado_fita) # Imprime o estado final da fita
    print("Estado final:", estado_final) # Imprime o estado final da máquina
```

```
self.fingerprints = set()
self.logduplicates = True
self.debug = debug
self.logger = logging.getLogger()
if path:
    self.file = open(os.path.join(path, 'fingerprint.log'), 'w')
    self.file.seek(0)
    self.fingerprints.update(fingerprint_for_file(self.file))
@classmethod
def from_settings(cls, settings):
    debug = settings.getboolean('debug')
    return cls(job_dir=settings['job_dir'],
               logduplicates=settings['logduplicates'],
               debug=debug,
               logger=settings['logger'],
               fingerprints=settings['fingerprints'])
def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + '\n')
def request_fingerprint(self, request):
    return request_fingerprint(request)
```

05

Analisando as transições



Analizando nossas transições

```
transicoes = {  
    ('q0', '*'): ('q0', '*', 'D'),  
    ('q0', 'a'): ('q1', 'A', 'D'),  
    ('q0', 'B'): ('q3', 'B', 'D'),  
    ('q0', '_'): ('qf', '_', 'D'),  
    ('q1', 'a'): ('q1', 'a', 'D'),  
    ('q1', 'B'): ('q1', 'B', 'D'),  
    ('q1', 'b'): ('q2', 'B', 'E'),  
    ('q2', 'a'): ('q2', 'a', 'E'),  
    ('q2', 'A'): ('q0', 'A', 'D'),  
    ('q2', 'B'): ('q2', 'B', 'E'),  
    ('q3', 'B'): ('q3', 'B', 'D'),  
    ('q3', '_'): ('qf', '_', 'D')  
}
```

Vamos analisar todos os passos possíveis da Máquina de Turing baseada na função de transição. A função de transição é definida como um dicionário, onde cada chave é uma tupla (estado_atual, simbolo_lido) e cada valor é uma tupla (proximo_estado, simbolo_escrito, direcao).

A nossa fita inicial é ***aabb_**

Descrevendo Estados e transições

Q0 Fita: `*aabb_`

- Cabeça de leitura/escrita na posição 0 (símbolo: `*`)
- Estado: `q0`

Transição: `('q0', '*') -> ('q0', '*', 'D')`

- Escreve `*` na posição 0.
- Move a cabeça para a direita.
- Permanece no estado `q0`.

Q1 Fita: `*Aabb_`

- Cabeça de leitura/escrita na posição 2 (símbolo: `a`)
- Estado: `q1`

Transição: `('q1', 'a') -> ('q1', 'a', 'D')`

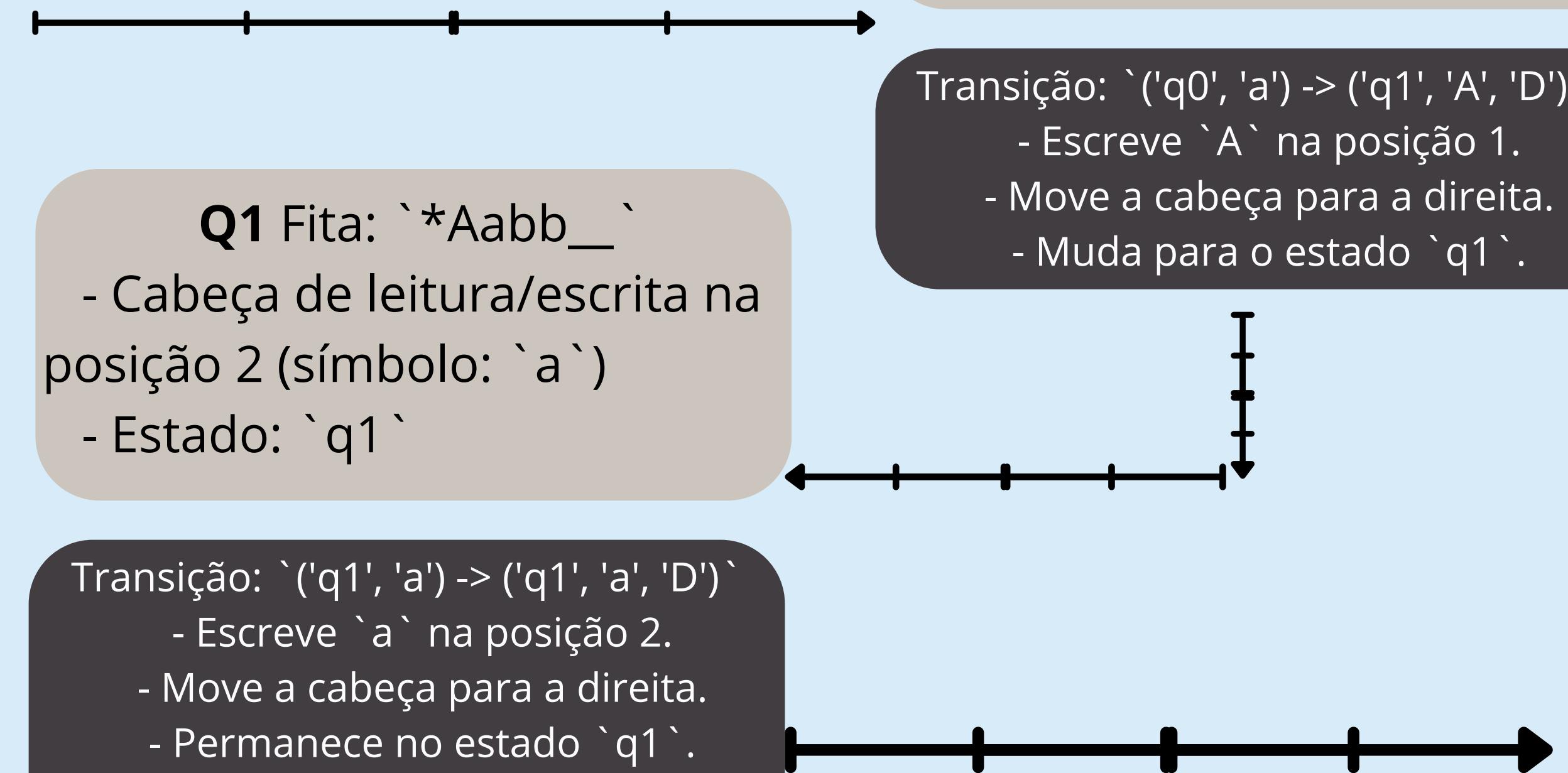
- Escreve `a` na posição 2.
- Move a cabeça para a direita.
- Permanece no estado `q1`.

Q0 Fita: `*aabb_`

- Cabeça de leitura/escrita na posição 1 (símbolo: `a`)
- Estado: `q0`

Transição: `('q0', 'a') -> ('q1', 'A', 'D')`

- Escreve `A` na posição 1.
- Move a cabeça para a direita.
- Muda para o estado `q1`.



Descrevendo Estados e transições

Q1 Fita: `*Aabb_`

- Cabeça de leitura/escrita na posição 3 (símbolo: `b`)
- Estado: `q1`

Transição: `('q1', 'b') -> ('q2', 'B', 'E')`

- Escreve `B` na posição 3.
- Move a cabeça para a esquerda.
- Muda para o estado `q2`.

Q2 Fita: `*AaBb_`

- Cabeça de leitura/escrita na posição 1 (símbolo: `A`)
- Estado: `q2`

Transição: `('q2', 'A') -> ('q0', 'A', 'D')`

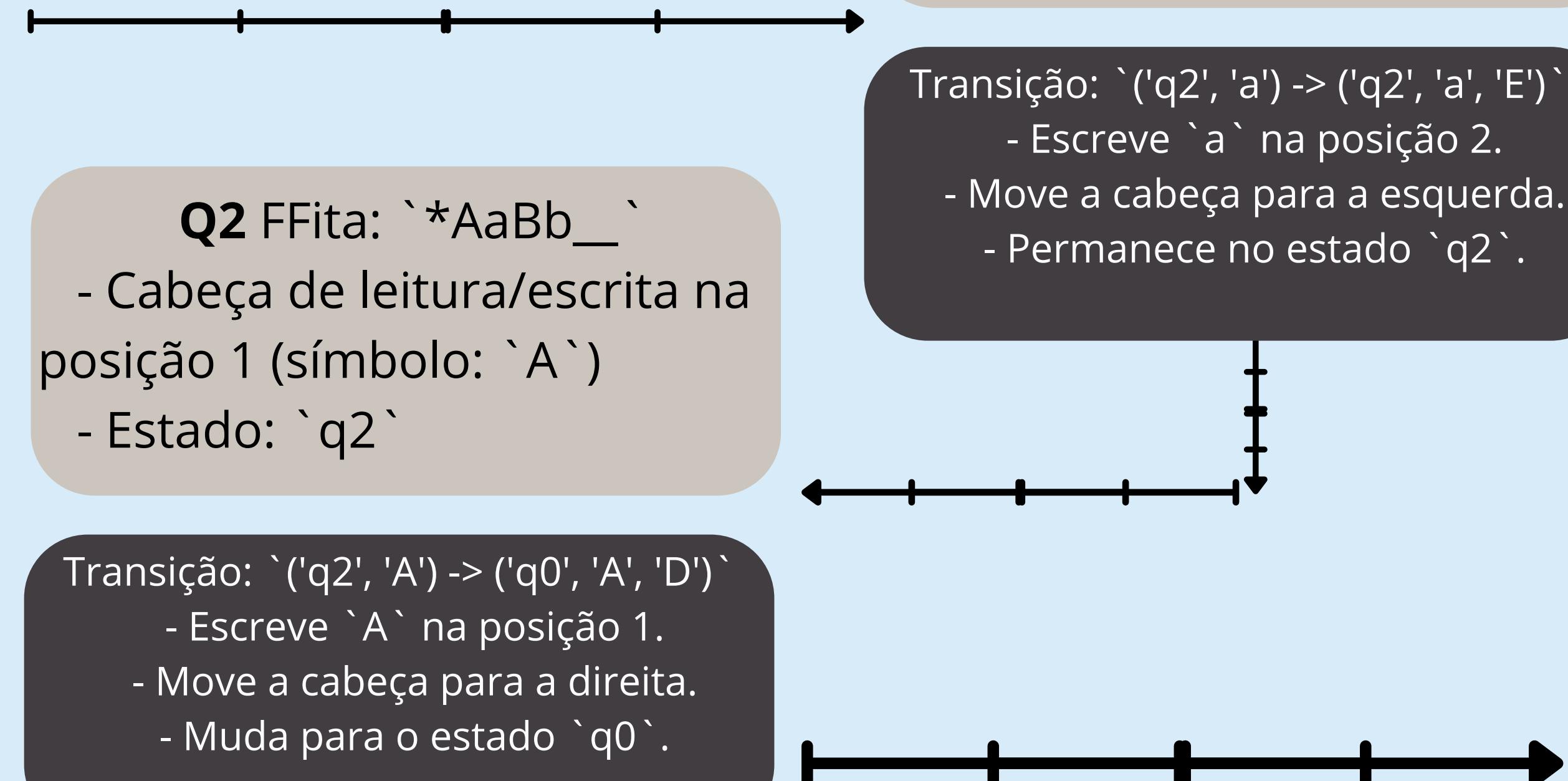
- Escreve `A` na posição 1.
- Move a cabeça para a direita.
- Muda para o estado `q0`.

Q2 Fita: `*AaBb_`

- Cabeça de leitura/escrita na posição 2 (símbolo: `a`)
- Estado: `q2`

Transição: `('q2', 'a') -> ('q2', 'a', 'E')`

- Escreve `a` na posição 2.
- Move a cabeça para a esquerda.
- Permanece no estado `q2`.



Descrevendo Estados e transições

Q0 Fita: `*AaBb_`

- Cabeça de leitura/escrita na posição 2 (símbolo: `a`)
- Estado: `q0`

Transição: `('q0', 'a') -> ('q1', 'A', 'D')`

- Escreve `A` na posição 2.
- Move a cabeça para a direita.
- Muda para o estado `q1`.

Q1 Fita: `*AABb_`

- Cabeça de leitura/escrita na posição 4 (símbolo: `b`)
- Estado: `q1`

Q1 Fita: `*AABb_`

- Cabeça de leitura/escrita na posição 3 (símbolo: `B`)
- Estado: `q1`

Transição**: `('q1', 'B') -> ('q1', 'B', 'D')`

- Escreve `B` na posição 3.
- Move a cabeça para a direita.
- Permanece no estado `q1`.

Transição**: `('q1', 'b') -> ('q2', 'B', 'E')`

- Escreve `B` na posição 4.
- Move a cabeça para a esquerda.
- Muda para o estado `q2`.

Descrevendo Estados e transições

Q2 Fita: `*AABB_`

- Cabeça de leitura/escrita na posição 3 (símbolo: `B`)
- Estado: `q2`



Transição: `('q2', 'B') -> ('q2', 'B', 'E')`

- Escreve `B` na posição 3.
- Move a cabeça para a esquerda.
- Permanece no estado `q2`.

Q0 Fita: `*AABB_`

- Cabeça de leitura/escrita na posição 3 (símbolo: `A`)
- Estado: `q0`



Transição: `('q0', 'B') -> ('q3', 'B', 'D')`

- Escreve `B` na posição 3.
- Move a cabeça para a direita.
- Muda para o estado `q3`.



Q2 Fita: `*AABB_`

- Cabeça de leitura/escrita na posição 2 (símbolo: `A`)
- Estado: `q2`

Transição: `('q2', 'A') -> ('q0', 'A', 'D')`

- Escreve `A` na posição 2.
- Move a cabeça para a direita.
- Muda para o estado `q0`.

Descrevendo Estados e transições

Q3 Fita: `*AABB_`

- Cabeça de leitura/escrita na posição 4 (símbolo: `B`)
- Estado: `q3`



Q3 Fita: `*AABB_`

- Cabeça de leitura/escrita na posição 5 (símbolo: `_`)
- Estado: `q3`

Transição: `('q3', 'B') -> ('q3', 'B', 'D')`

- Escreve `B` na posição 4.
- Move a cabeça para a direita.
- Permanece no estado `q3`.

Transição**: `('q3', '_') -> ('qf', '_', 'D')`

- Escreve `_` na posição 5.
- Move a cabeça para a direita.
- Muda para o estado `qf`.

Estado Final

- Fita: `*AABB_`
- Estado final: `qf`



Estado de Aceitação

Portanto, a Máquina de Turing aceita a entrada e termina com a fita modificada para `*AABB_` no estado final `qf`.