**Teoria dos Autômatos e Linguagens Formais**

**aula 16: O autômato de pilha**

## 1 Introdução

→ *A máquina que lê o que ela mesmo escreve ...*

Suponha que as letras de uma palavra serão mostradas a você uma a uma, digamos em uma tela.

Ao final desse processo você deverá responder se a palavra possui ou não uma certa propriedade — (*que você sabe qual é desde o início*)

Desta vez, além dos seus dedos, você pode escrever símbolos em pequenas folhas de papel.

Cada folha só pode conter um símbolo, mas você pode usar quantas folhas quiser.

A única restrição é que as folhas escritas devem permanecer empilhadas, de modo que você só pode examinar a folha que está no topo da pilha.

Além disso, após examinar uma folha ela deve ser amassada e jogada em uma lixeira, de onde ela não pode ser recuperada mais tarde.

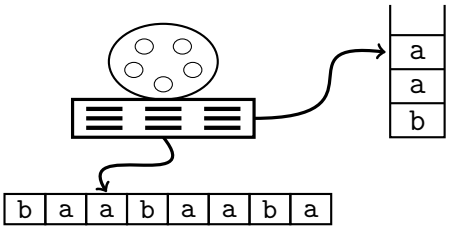
*Que linguagens você consegue reconhecer dessa maneira?*

É fácil ver que você pode reconhecer todas as linguagens regulares: basta ignorar as folhas de papel e você se comportar como um autômato.

*Mas, o que mais você pode fazer?*

## 2 O autômato de pilha

A Figura abaixo mostra um autômato de pilha




Agora, a cada passo de computação, o autômato pode ler 2 símbolos

- o próximo símbolo da palavra de entrada
- o símbolo no topo da pilha

E em cada passo de computação, o autômato pode

- mudar o seu estado atual
- escrever alguma coisa no topo da pilha

Como usual, o controle dessas operações é feito por meio de parâmetros associados às transições

$$q_i \xrightarrow{a[b,a]} q_j$$

Quer dizer,

- o primeiro `a` é o símbolo que é lido na palavra de entrada
- de dentro dos colchetes, nós vemos
  - o símbolo que é lido (e removido) no topo da pilha
  - e o símbolo que é colocado no topo da pilha ao final da transição

Em todos esses lugares, nós podemos colocar o símbolo `e` da palavra vazia

$$q_i \xrightarrow{e[b,a]} q_j \qquad q_i \xrightarrow{a[e,a]} q_j \qquad q_i \xrightarrow{a[b,e]} q_j$$

E frequentemente é conveniente ler e/ou escrever múltiplos símbolos na pilha em um único passo de computação

$$q_i \xrightarrow{a[bb,aaa]} q_j$$

Os autômatos de pilha são máquinas *não-determinísticas* por definição.

Isso significa que

→ *podem existir múltiplas transições disponíveis em um certo passo de computação*

E como usual, nós vamos assumir que o autômato de pilha *aceita* a palavra de entrada se existe um caminho de computação que lê a palavra de entrada inteira e alcança um estado final.

Finalmente, nós vamos assumir que o autômato precisa terminar a computação com a pilha vazia para que a palavra de entrada seja aceita.

Vejamos alguns exemplos concretos.

**Exemplos**

*a.* A linguagem $a^n b^n$

É fácil construir um autômato que reconhece a linguagem $a^n b^n$.

Quer dizer,

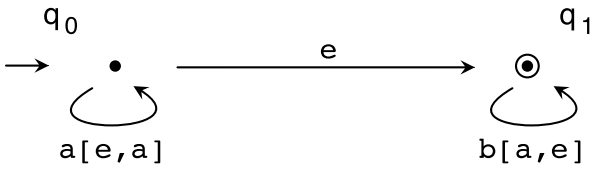
```
- primeiro a gente coloca um a na pilha
      para cada a que a gente lê na palavra de entrada
- e depois a gente remove um a da pilha
      para cada b que a gente lê na palavra de entrada
```

Dessa maneira, se os blocos de `a`'s e `b`'s tiverem o mesmo tamanho, a computação termina com a pilha vazia e a palavra é aceita pelo autômato.

Para organizar a computação, nós vamos utilizar dois estados:

- $q_0$: estado que lê os a's e os coloca na pilha
- $q_1$: estado que lê os b's e remove os a's da pilha

Abaixo nós temos o diagrama do autômato




É fácil ver que esse autômato de pilha aceita todas as palavras de $a^n b^n$ (inclusive a palavra vazia).

Mas, é interessante notar como ele rejeita as palavras que não estão na linguagem.

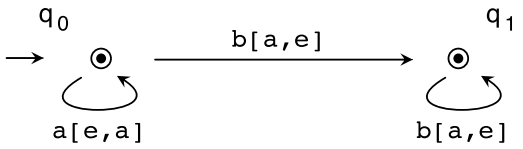
Se o bloco de a's é maior que o bloco de b's, então a computação não termina com a pilha vazia.

Se o bloco de a's é menor que o bloco de b's, então o autômato não consegue terminar a leitura da palavra de entrada, pois a leitura de um b requer um a no topo da pilha.

E a mesma coisa vale se a palavra não tem a forma $a^* b^*$, pois após a leitura de um b o autômato não consegue mais ler a's.

Uma última observação é que esse autômato não precisa ser não-determinístico.

Veja só




◇

*b.* A linguagem PAL

Lembre que PAL é a linguagem dos palíndromos.

E lembre que um palíndromo é uma palavra que é a mesma coisa lida da esquerda para a direita, e lida da direita para a esquerda.

A ideia para a construção do autômato é semelhante à que foi usada no exemplo anterior:

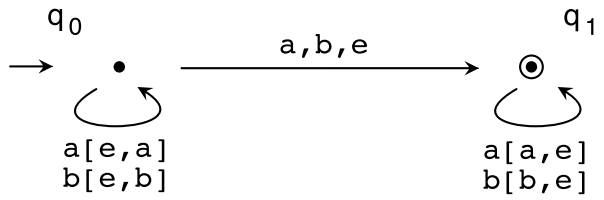
- durante a leitura da primeira metade da palavra, os símbolos são colocados na pilha;
- durante a leitura da segunda metade, verifica-se se eles são iguais aos que estão na pilha

Uma diferença importante é que dessa vez o não-determinismo é realmente necessário.

Porque não há como saber (deterministicamente) quando nós chegamos ao meio da palavra de entrada.

Além disso, no caso de palíndromos de comprimento ímpar, o símbolo central será lido mas não será empilhado.

A partir dessas observações, nós chegamos ao seguinte autômato de pilha para PAL:




É fácil ver que esse autômato aceita qualquer palíndromo.

E se a palavra não é um palíndromo, em algum momento no estado $\mathtt{q_1}$, o autômato vai ter um `a` na palavra de entrada e um `b` no topo da pilha (ou vice-versa), e não vai conseguir continuar a computação.

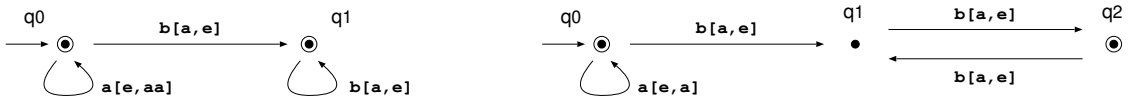
◇

*c.* A linguagem $\mathtt{a}^n\mathtt{b}^{2n}$

Existem duas estratégias para verificar se o bloco de `b`'s tem exatamente o dobro do tamanho do bloco de `a`'s:

1. a cada `a` lido na palavra de entrada colocam-se dois `a`'s na pilha
2. os `a`'s são lidos como antes, mas apenas as ocorrências pares de `b`'s removem um `a` da pilha

A figura abaixo apresenta autômatos de pilha que implementam essas duas estratégias




◇

*d.* Quase do mesmo tamanho

Considere a linguagem $\mathtt{a}^\mathtt{n}\mathtt{b}^\mathtt{m}$, onde $n \leq m \leq 2n$.

Esse é mais um exemplo onde é preciso utilizar o não-determinismo.

(*Você consegue ver porque?*)

A ideia é que o autômato vai advinhar o tamanho do bloco de `b`'s durante a leitura dos `a`'s, armazenando a quantidade correta de símbolos `a` na pilha.

Quer dizer, para cada símbolo `a` lido na palavra de entrada, o autômato vai escolher (não-deterministicamente) se coloca 1 ou 2 símbolos `a` na pilha.

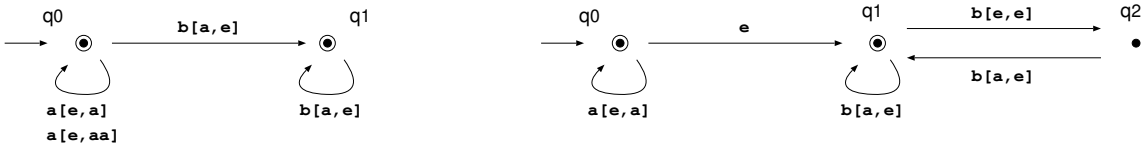
Caso as escolhas não-determinísticas sejam feitas de corretamente, após a leitura do bloco de `a`'s o número de símbolos na pilha é igual ao tamanho do bloco de `b`'s.

E a partir daí, basta fazer a leitura dos `b`'s removendo um símbolo `a` da pilha de cada vez.

Mas, nós também podemos fazer as coisas de maneira diferente.

Quer dizer, nós podemos empilhar o bloco de `a`'s, e escolher (não-deterministicamente) se cada `a` é removido com a leitura de um ou dois `b`'s da palavra de entrada.

A figura abaixo apresenta diagramas que implementam essas duas ideias




◇

*e.* A linguagem BAL

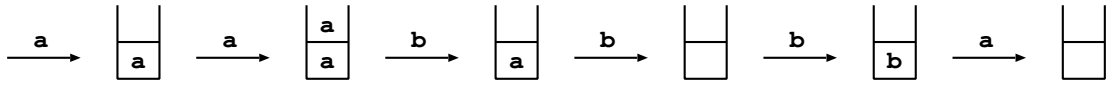
Considere a linguagem

BAL = palavras que possuem a mesma quantidade de a's e b's

A ideia natural aqui consiste em utilizar a pilha para lembrar a diferença entre as quantidades de a's e b's que foram lidos até o momento.

Para uma palavra de BAL essa diferença é igual a zero no final da computação, o que corresponde à pilha vazia, e portanto a palavra é aceita.

Por exemplo, nós teríamos a seguinte sequência de configurações da pilha para a computação sobre a palavra aabbba




Ou seja, quando um a é lido na palavra de entrada e o topo da pilha também é um a, o símbolo é empilhado; caso o topo da pilha seja um b, ele é removido.

A situação é análoga no caso da leitura de um b.

E quando a pilha está vazia, o símbolo lido é simplesmente colocado lá.

Mas, esse último caso introduz uma nova dificuldade:

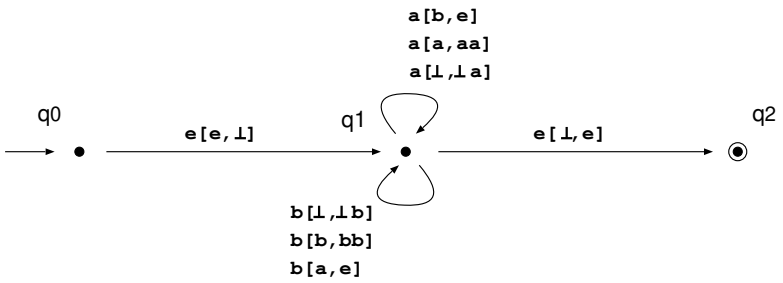
- *Como é que o autômato pode saber que a pilha está vazia?*

Uma solução simples consiste em utilizar um símbolo marcador $\perp$, que permanece na base da pilha durante toda a computação.

Quando $\perp$ é o topo da pilha isso é interpretado como se a pilha estivesse vazia.

Para fazer a ideia funcionar, é preciso colocar o símbolo $\perp$ na pilha no primeiro passo da computação e removê-lo no último passo.

Utilizando essa ideia, é fácil construir um autômato de pilha que reconhece a linguagem BAL:




Note que nos momentos em que se detecta que a pilha está vazia, o símbolo $\perp$ é efetivamente removido do topo da pilha.

Por esse motivo é preciso colocá-lo novamente de volta, como é feito nas transições

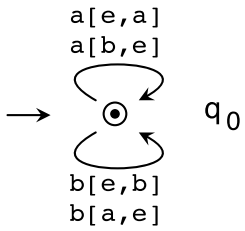
$$q_1 \xrightarrow{a[\perp,\perp a]} q_1 \qquad\qquad q_1 \xrightarrow{b[\perp,\perp b]} q_1$$

Finalmente, quando se chega ao final da palavra de entrada e o símbolo $\perp$ está no topo da pilha, o autômato pode realizar a transição

$$q_1 \xrightarrow{e[\perp,e]} q_2$$

para esvaziar a pilha e passar para o estado final.

*Solução não-determinística:*




◇

*f.* Dois blocos iguais

Considere a linguagem

$L$ = palavras que possuem dois blocos de a's do mesmo tamanho

Esse é mais um exemplo onde nós precisamos utilizar o não-determinismo.

A ideia consiste em advinhar os momentos em que os dois blocos de a's do mesmo tamanho aparecem.

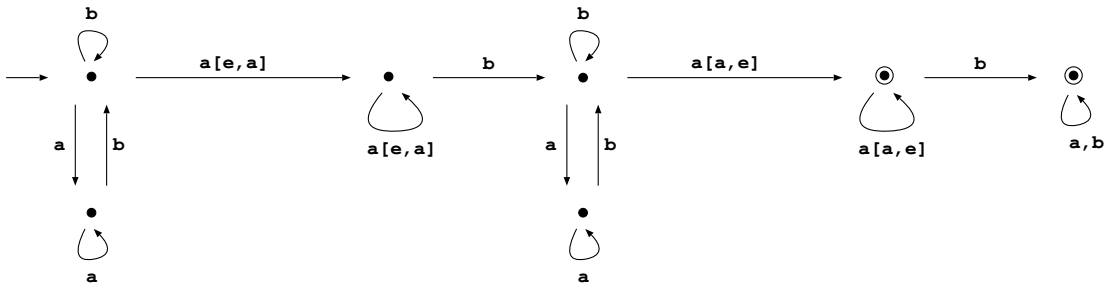
Para fazer isso, todos os símbolos b e todos os outros blocos de a's são ignorados (i.e., os símbolos são lidos e descartados).

Quando se encontra (não-deterministicamente) o primeiro bloco de a's que interessa, os seus símbolos são colocados na pilha.

E quando se encontra o segundo bloco de a's de interesse, é preciso verificar que ele tem o mesmo tamanho que o primeiro.

O único cuidado que é preciso tomar é que os blocos de a's descartados sejam lidos até o fim, para evitar que o autômato utilize apenas uma parte de um bloco na comparação.

Abaixo nós temos o autômato de pilha que implementa essa ideia




◇