Teoria dos Autômatos e Linguagens Formais

aula 02: Máquinas e programas

1 Introdução

Na aula passada nós vimos a máquina computadora que só faz leitura.

E vimos que a máquina que só faz leitura não serve para nada.

Quer dizer, se a máquina não escreve nada então ela também não faz nada.

Porque, para fazer alguma coisa a máquina precisa escrever alguma coisa.

Mas a gente também não viu as nossas máquinas escrevendo.

Quer dizer, as nossas máquinas não escreviam em cima da palavra que estava sendo lida.

Mas elas estavam escrevendo.

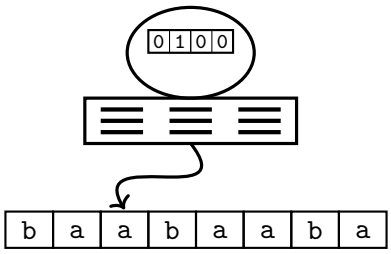
Para ver isso, é preciso olhar direito.

Quer dizer, as nossas máquinas tinham luzinhas, não é?

E as luzinhas ficavam acendendo e apagando, não é?

Pois bem, isso é a escrita acontecendo.

Mas a coisa fica mais fácil de ver se a gente desenha a máquina assim



Quer dizer, a gente pode pensar que existe uma pequena memória dentro da cabeça da máquina — (*onde cada bit é uma luzinha*)

E é nessa memória que a máquina escreve.

Certo.

Isso a gente já entendeu.

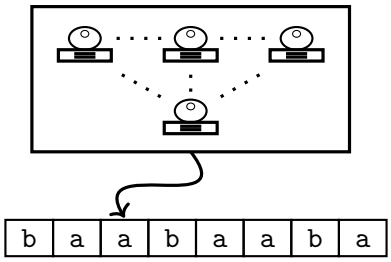
Mas daí, a gente pode perguntar: *Quem é que lê essa memória onde a máquina escreve?*

E a resposta é bem simples: *É a própria máquina quem lê aquilo que ela escreve.*

Quer dizer, a gente pode pensar que a coisa funciona assim

- No momento em que a máquina vai ler um símbolo da palavra de entrada,
  ela examina as suas luzinhas
- E daí, configurações de luzinhas diferentes levam a máquina a fazer coisas diferentes

Ora, mas daí a gente pode pensar que a nossa máquina é uma coleção de maquinhas menores que ficam passando o controle umas para as outras



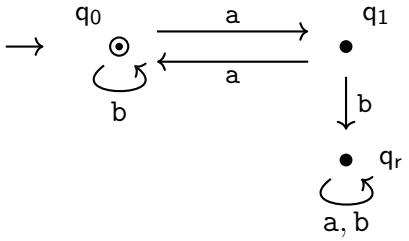
Hmm ...

2 Máquinas e programas

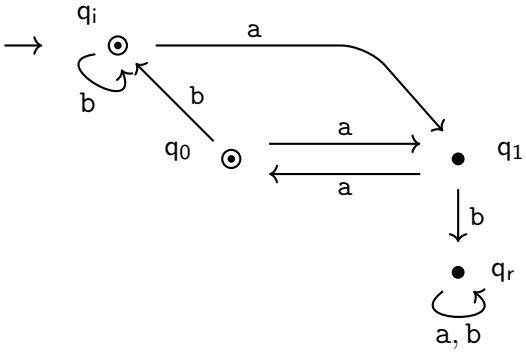
Na aula passada, nós consideramos a seguinte tarefa

→ *Verificar se todos os blocos de* a*'s da palavra de entrada tem tamanho par*

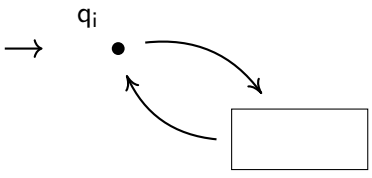
E nós contruímos o seguinte autômato para ela



Mas é mais fácil ver o que está acontecendo se a gente constrói o autômato assim

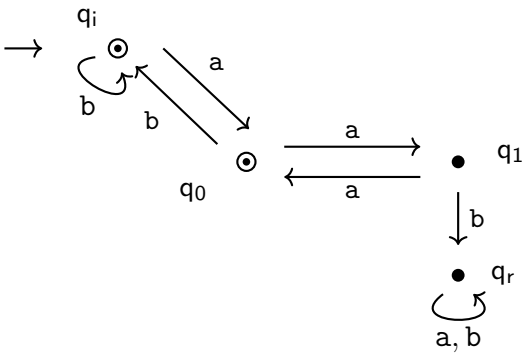


Porque daí, a gente vê que a coisa funciona assim

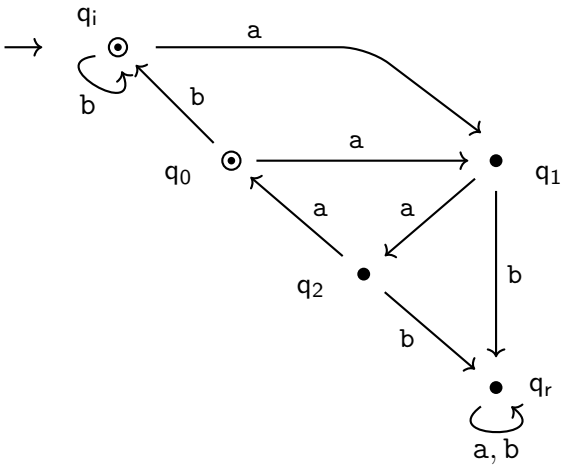


Sim! Um laço de repetição ...

Agora que nós temos uma estrutura de alto nível, é bem fácil modificar a coisa para aceitar palavras com blocos de a's de tamanho ímpar

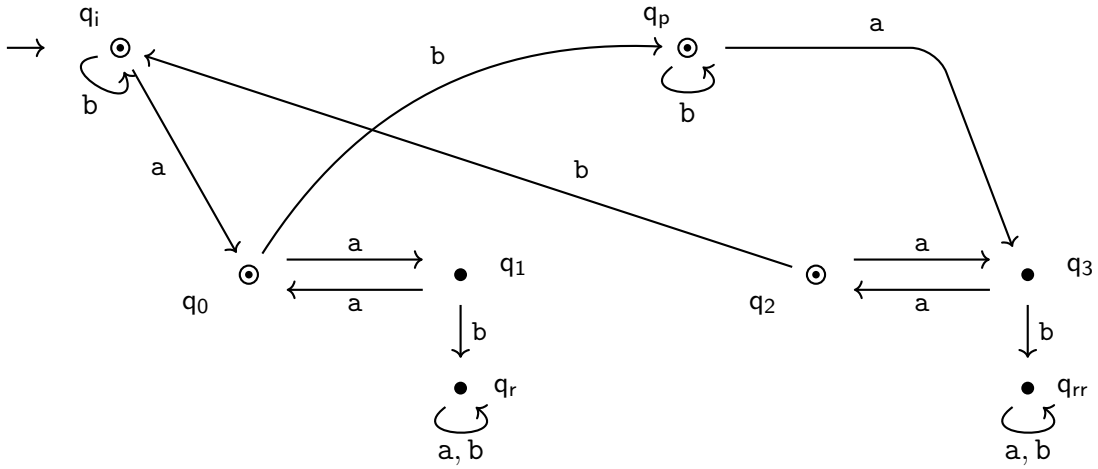


Ou palavras com blocos de a's de tamanho múltiplo de 3



O próximo passo é reconhecer palavras onde os blocos pares e ímpares se alternam — (*começando com um bloco ímpar*)

Para isso, nós precisamos de dois laços e uma estrutura de controle de nível mais alto



Exemplos

a. If-then-else

Agora imagine que nós queremos um autômato que realiza a seguinte tarefa

→ *Verificar se todos os blocos de* a*'s da palavra possuem a mesma paridade*

Por exemplo, todas as palavras abaixo tem essa propriedade

aabaabaa, abaaaba, bbb, bbabbbb, etc.

Mas essas outras não tem

abbaa, aabbabbaa, aaabaaabaa, etc.

Certo.

Não é difícil ver que a paridade é decidida no primeiro bloco de a's.

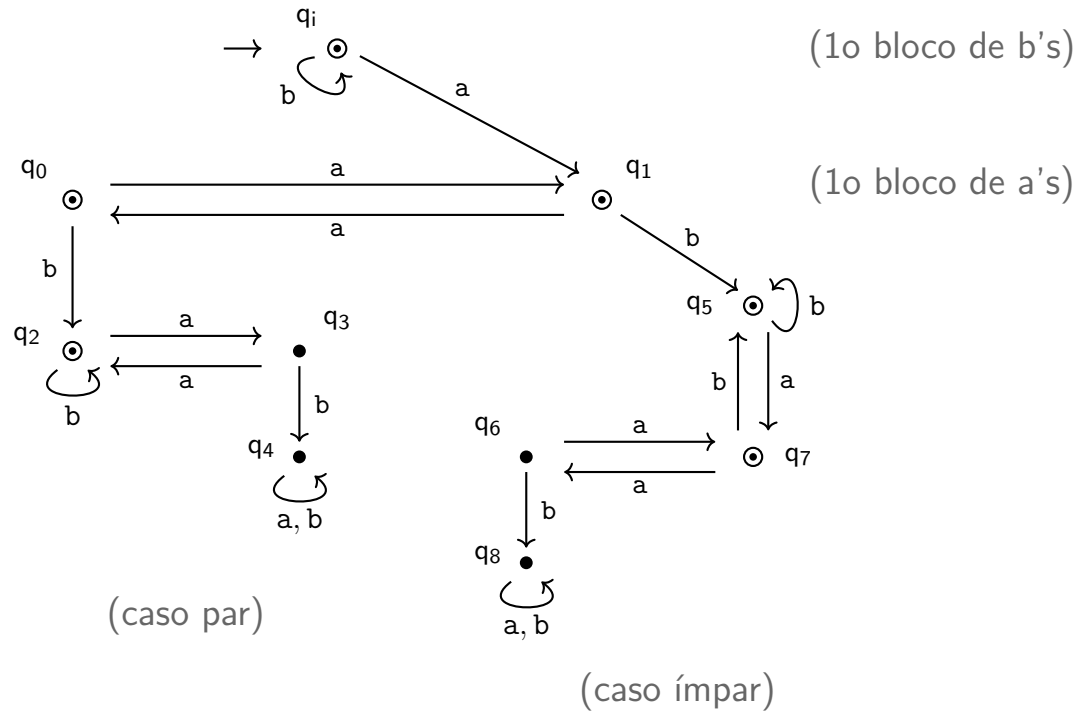
Quer dizer,

- se a leitura do primeiro bloco de a's termina em q_0,
  então todos os blocos deve ter tamanho par
- e se a leitura do primeiro bloco de a's termina em q_1,
  então todos os blocos deve ter tamanho ímpar

Ora, mas nós já temos um autômato que verifica se todos os blocos de a's são pares.

E não é difícil adaptar esse autômato para o caso ímpar.

Daí que, a coisa fica assim



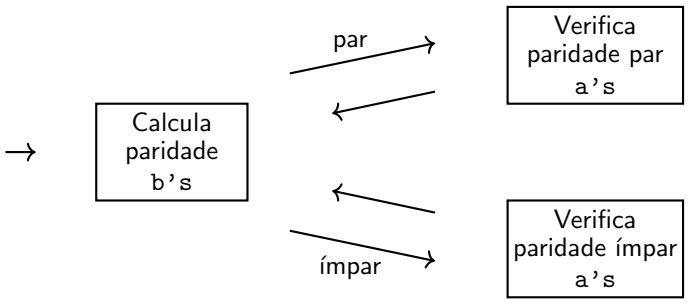
◇

b. Modularidade

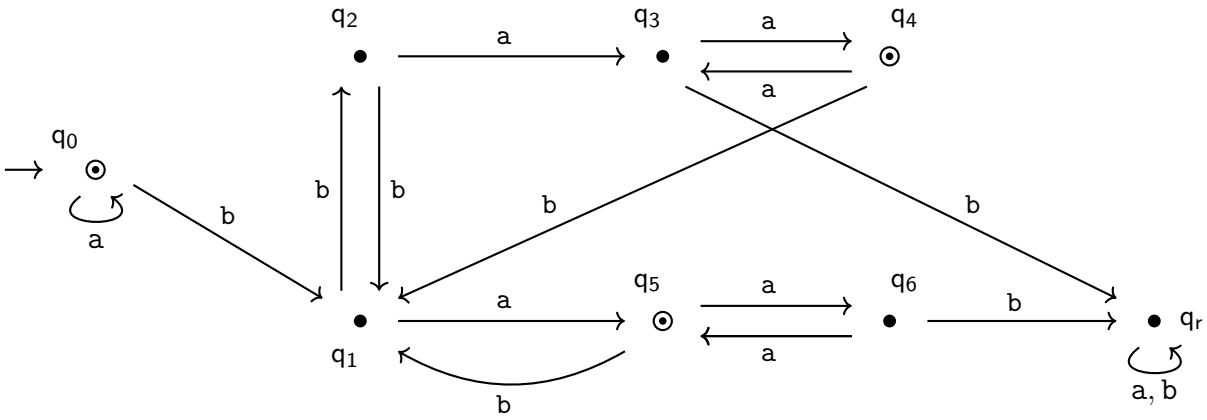
Agora considere a tarefa

→ *Verificar se todo bloco de* `b`*'s na palavra de entrada é seguido por um bloco de* `a`*'s com a mesma paridade*

Bom, raciocinando em alto nível, nós temos o seguinte

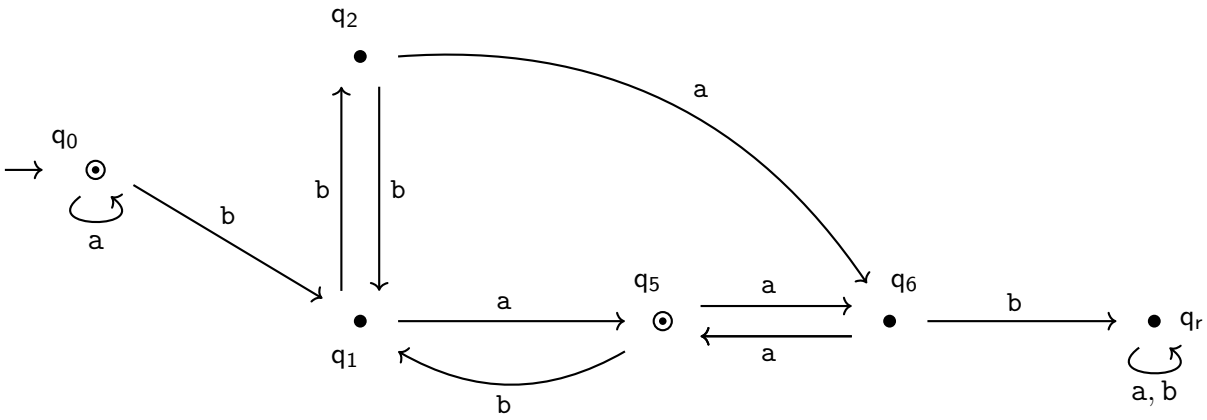


E daí, não é difícil construir o seguinte autômato



Agora, o mais legal é que as verificações de paridade par e ímpar dos blocos de `a`'s podem ser feitas pelo mesmo módulo.

E daí, a coisa fica assim



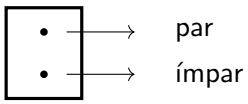
◇

c. Um autômato de autômatos

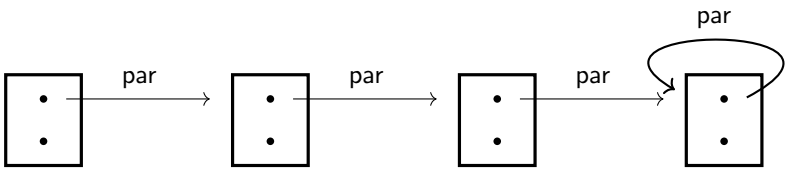
Agora considere a seguinte tarefa

→ *Verificar se todo bloco ímpar de* `a`*'s é precedido por ao menos 3 blocos pares de* `a`*'s*

Aqui, a ideia é imaginar que nós temos uma caixinha que calcula a paridade de um bloco de `a`'s

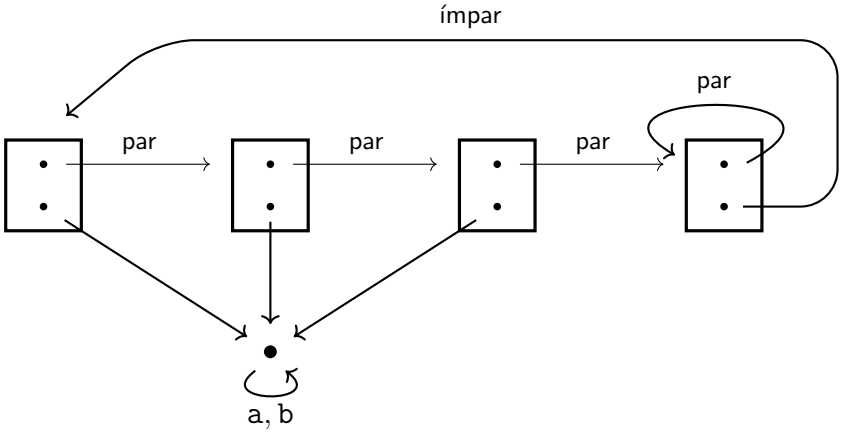


Daí, a gente pode utilizar 4 caixinhas desse tipo para contar os blocos pares de `a`'s



E daí,

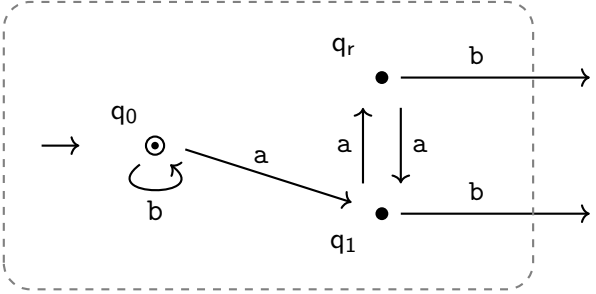
- blocos ímpares de `a`'s antes da hora levam o autômato para o estado de rejeição
- e um bloco ímpar de `a`'s no último estágio nos levam de volta ao início



Pronto!

A lógica de alto nível já está completa.

Agora, a gente pode definir a nossa caixinha assim



◇

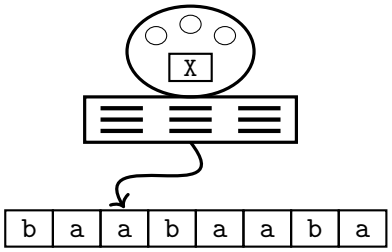
d. Autômato com variável

No início da aula, nós vimos que as mudanças de estado em um autômato podem ser interpretadas como uma operação de escrita.

Mas, a forma padrão de escrita que nós conhecemos consiste em armazenar um valor em uma variável

$$\texttt{X} \longleftarrow \texttt{1}$$

Daí que, faz sentido pensar em um autômato com variável



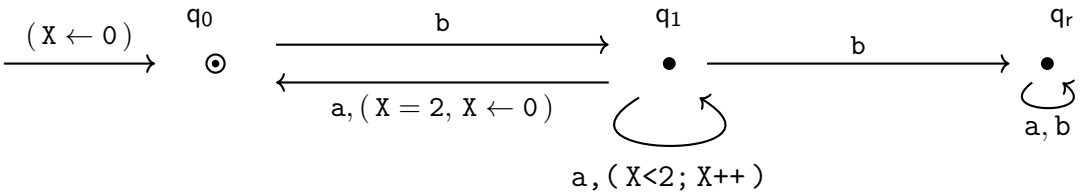
Vejamos como isso funciona.

Considere a seguinte tarefa

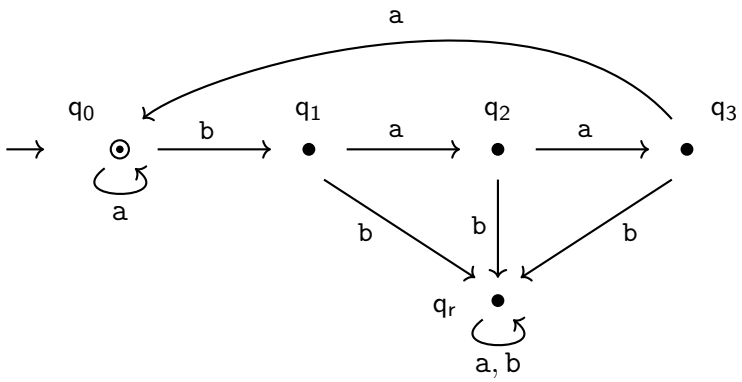
→ *Verificar se todo* `b` *na palavra de entrada é seguido por ao menos 3* `a`*'s*

Bom, a ideia é que se o autômato tem uma variável `X`, então nós podemos utilizá-la para contar os `a`'s que vem após um `b`.

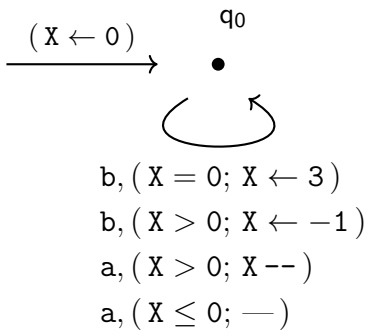
A coisa fica assim



É bem fácil ver que esse autômato com variável pode ser transformado em um autômato comum — (*i.e., cujo controle só possui estados*)



E também não é difícil construir um autômato que utiliza apenas a variável X para controlar a computação



onde o autômato aceita a palavra de entrada quando a computação termina com X = 0 — (*note que a contagem de* a*'s é feita de modo decrescente*)

Mas, um esse autômato é basicamente um programa de computador

```
X  <--  0
Para cada símbolo s da palavra de entrada
 {
    Se ( s = b  e  X = 0 )    X  <--  3
    Se ( s = b  e  X > 0 )    X  <--  -1
    Se ( s = a  e  X > 0 )    X --
 }
Se ( X = 0 )   Responda ( SIM )
Senão          Responda ( NÃO )
```

◇