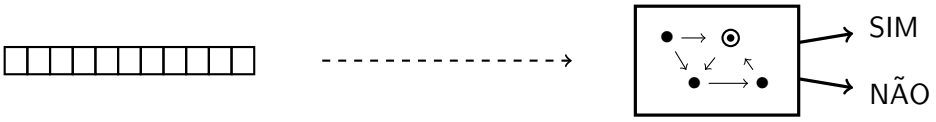
Teoria dos Autômatos e Linguagens Formais

aula 03: Palavras, padrões e propriedades

1 Introdução

Os autômatos são máquinas computadoras muito simples.

Só o que há lá dentro são estados e transições



Mas combinando os estados e transições de maneira adequada, a gente consegue construir autômatos que reconhecem toda sorte de propriedades interessantes.

Hoje nós vamos olhar para o outro lado dessa história.

E vamos começar a examinar as propriedades que os autômatos conseguem reconhecer.

Daí, vai acontecer uma coisa engraçada.

A gente vai ver que os autômatos conseguem reconhecer algumas propriedades bem simples.

E daí a gente vai ver que combinando essas propriedades simples dá para definir propriedades complicadas — (*que os autômatos também conseguem reconhecer*)

Quer dizer, existe combinação dos dois lados da história.

No lado dos autômatos a gente combina estados e transições.

E no lado das palavras a gente combina blocos e padrões — (*utilizando o vocabulário lógico e a operação de concatenação*)

Então no final das contas, é como se estivesse acontecendo a mesma coisa dos dois lados.

E como se cada lado fosse o reflexo do outro lado ...

Nota: *Na prática, é ainda um pouco mais do que isso.*

Porque a gente pode se apoiar de um lado para realizar construções do outro lado.

Um dos lados dessa história é fácil de ver.

Mas o outro ainda não é ...

2 Articulação lógica

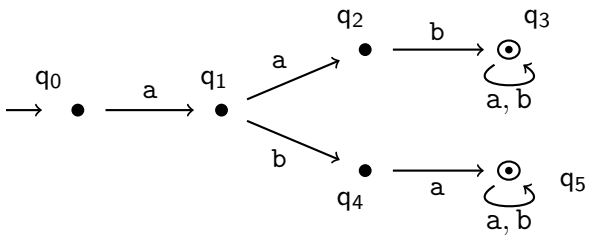
Considere a seguinte tarefa

→ *Verificar se a palavra contém o padrão* `aab` *ou o padrão* `aba`

Nós já sabemos que para procurar um padrão na palavra de entrada é preciso ter um caminho no autômato com a mesma sequência de símbolos.

Nesse caso nós temos dois padrões, logo nós vamos precisar de dois caminhos

Mas, como os dois padrões começam com o mesmo símbolo, os dois caminhos podem compartilhar a primeira transição

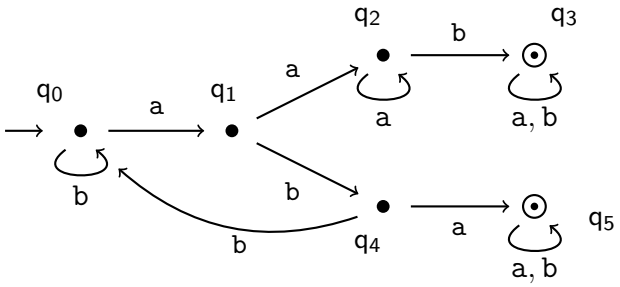


No diagrama acima, nós já identificamos o estado inicial `q0`, e já marcamos os estados `q3` e `q5` como estados finais (ou estados que respondem SIM), pois esses são os lugares onde os padrões são identificados.

Finalmente, as seguintes observações permitem concluir a construção

- a leitura de um `b` em q_0 não muda o estado do autômato, pois os dois padrões começam com `a`
- a leitura de um `a` em `q2` também não muda o estado do autômato, pois esse é o lugar onde ele se encontra quando acabou de ler `aa`
- a leitura de um `b` em `q3` significa que os últimos dois símbolos são `bb`; isso não é o início de nenhum dos dois padrões, logo o autômato volta para `q0`

Abaixo nós temos o diagrama completo



A seguir nós vamos ver mais exemplos.

Exemplos

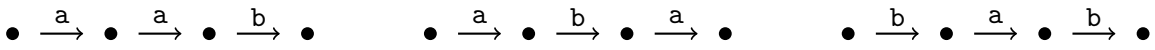
a. Disjunção tripla

Considere a seguinte tarefa

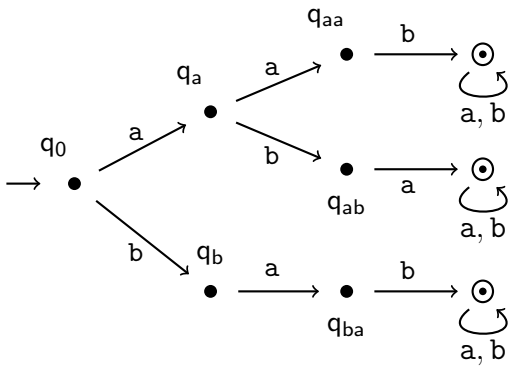
→ *Verificar se a palavra contém um dos seguintes padrões:* `aab`, `aba` *ou* `bab`

Essa tarefa é semelhante à anterior.

Mas dessa vez, nós precisamos combinar 3 caminhos



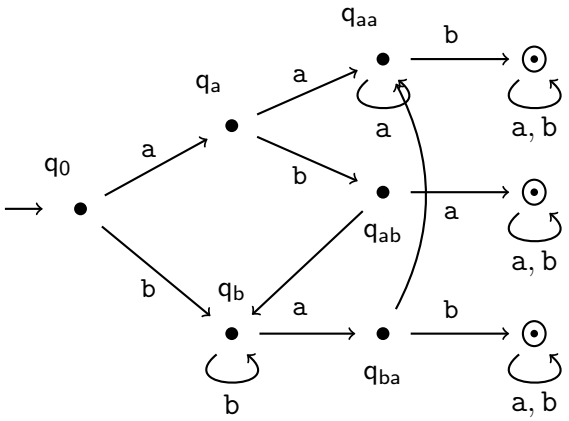
em um único diagrama



A ideia aqui foi nomear os estados com os símbolos que foram lidos até aquele ponto para facilitar a identificação das outras transições

- em q_{aa} a leitura de um a mantém o autômato em q_{aa}
- em q_{ab} a leitura de um b leva o autômato para q_b
- em q_b a leitura de um b mantém o autômato em q_b
- em q_{ba} a leitura de um a leva o autômato para q_{aa}

Abaixo nós temos o diagrama completo



Outra solução:

Esse autômato ficou um pouquinho mais complicado do que devia.

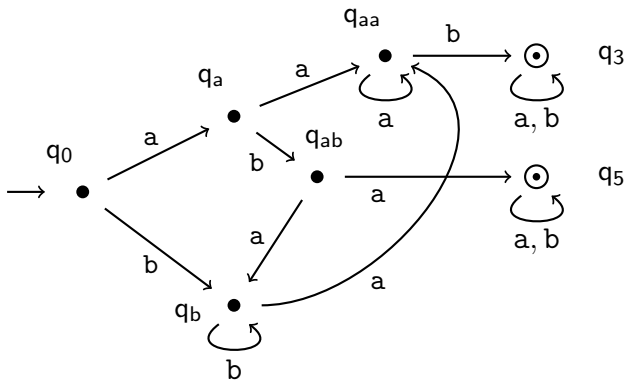
Mas isso são coisas que a gente só descobre no fim da construção.

Porque no fim da construção a gente pode ver o que está acontecendo.

Quer dizer, examinando o diagrama a gente vê que os estados q_{aa} e q_{ba} correspondem à mesma situação:

- só falta um b para que um dos padrões seja identificado
- a leitura de um a leva o autômato para q_{aa}

Daí que, a construção do autômato pode ser simplificada transformando q_{aa} e q_{ba} em um único estado



Note que os estados q_3 e q_5 também poderiam ser transformados em um único estado.

Porque eles também correspondem à mesma situação: a resposta SIM.

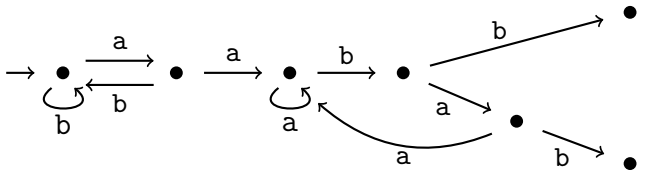
◇

b. Conjunção

Considere a seguinte tarefa

→ *Verificar se a palavra contém o padrão* `aabb` *e o padrão* `aabab`

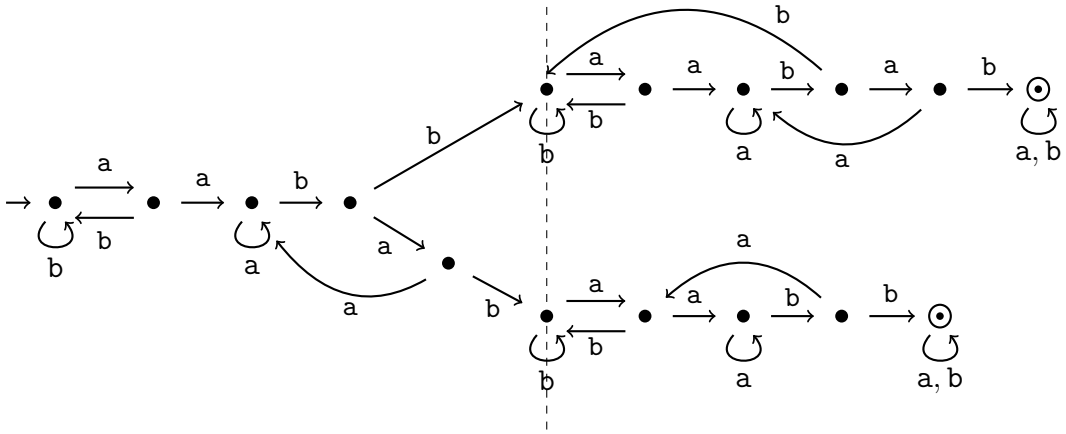
A ideia aqui é começar a construção com um autômato que reconhece `aabb` ou `aabab`



Daí,

- No ponto em que o autômato identifica o padrão `aabb`, ele passa a procurar pelo padrão `aabab`
- E no ponto em que o autômato identifica o padrão `aabab`, ele passa a procurar pelo padrão `aabb`

A coisa fica assim



Note que esse diagrama pode ser visto como a concatenação de 3 autômatos

- Na parte esquerda, nós temos o autômato que reconhece `aabb` ou `aabab`
- Na parte direita, nós temos o autômato que reconhece `aabab` (em cima) e o autômato que reconhece `aabb` (embaixo)

◇

c. Padrões com sobreposição

Agora considere a seguinte tarefa

→ *Verificar se a palavra contém o padrão* `aab` *e o padrão* `aba`

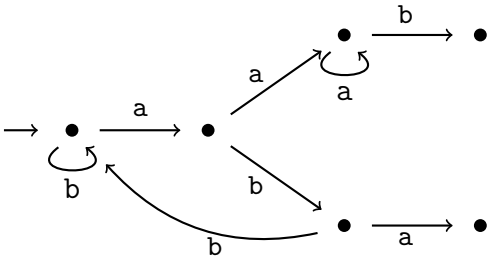
A novidade desse exemplo é que os dois padrões podem aparecer sobrepostos

Quer dizer, uma palavra que tem a sequência de símbolos `abaab` possui ocorrências dos padrões `aba` e `aab` que compartilham um símbolo.

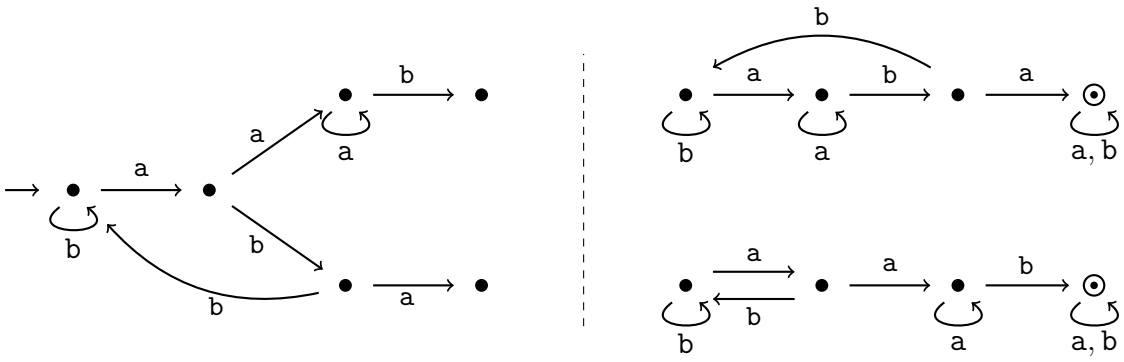
E uma palavra que tem a sequência de símbolos `aaba` possui ocorrências dos padrões `aab` e `aba` que compartilham dois símbolos.

Apesar disso, nós podemos utilizar a mesma ideia do exemplo anterior para construir o autômato.

Quer dizer, nós começamos com um autômato que reconhece `aab` ou `aba`



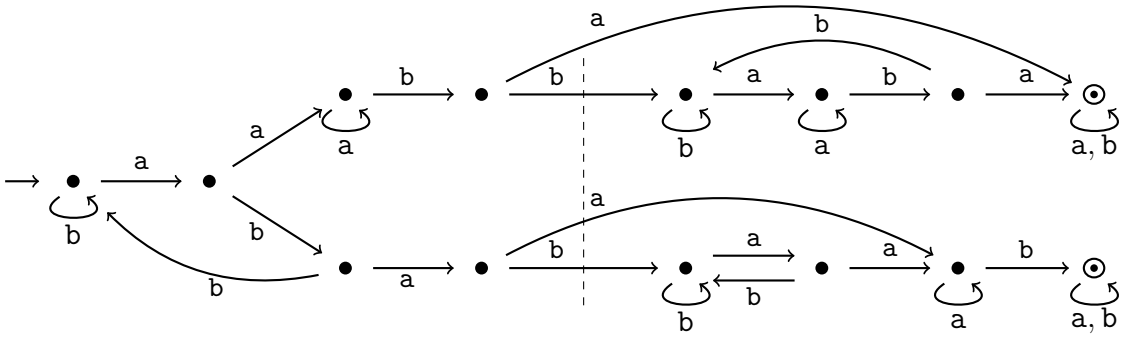
Daí, nós colocamos no lado direito o autômato que reconhecem `aba` (em cima), e o autômato que reconhece `aab` (embaixo)



E agora nós raciocinamos da seguinte maneira

- Logo após a identificação do padrão `aab`, um símbolo `a` completa uma ocorrência do padrão `aba`, mas um símbolo `b` nos obriga a começar a procura por `aba` a partir do zero.
- Logo após a identificação do padrão `aba`, um símbolo `a` já nos dá a porção inicial `aa` do padrão `aab`, mas um símbolo `b` nos obriga a começar a procura por `aab` a partir do zero.

Com base nessas observações, nós completamos o diagrama da seguinte maneira



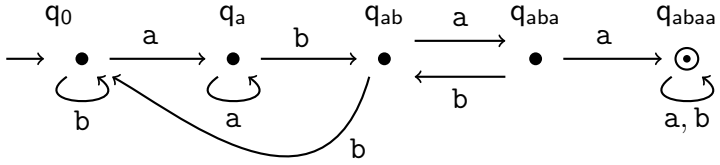
◇

d. Negação

Agora considere a tarefa

→ *Aceitar as palavras que <u>não</u> possuem o padrão* `abaa`

Na aula passada nós vimos que o autômato abaixo aceita as palavras que possuem o padrão `abaa`



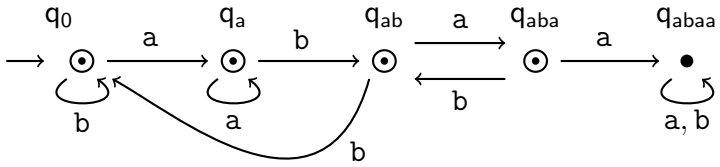
A ideia aqui é que no momento em que o padrão é encontrado, o autômato alcança o estado q_{abaa} e não sai mais de lá.

E como q_{abaa} é um estado final, o autômato responde SIM no final da computação.

Por outro lado, se o padrão `abaa` jamais é encontrado, o autômato permanece nos outros estados.

E como esses estados são todos não finais o autômato responde NÃO no final da computação.

Ora, agora é bem fácil ver que basta inverter a marcação de estados finais e não finais



◇

3 A operação de concatenação

Agora nós vamos considerar uma maneira diferente de definir as palavras que devem ser reconhecidas pelo autômato.

A ideia é que a palavra será formada pela concatenação sucessiva de um ou mais padrões.

Começando com o caso mais simples de todos, considere o padrão `aba`.

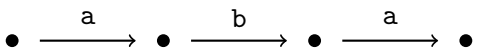
Daí, nós podemos considerar as palavras que são formadas por concatenações do bloco `aba`

`aba`, `abaaba`, `abaabaaba`, `abaabaabaaba`, ...

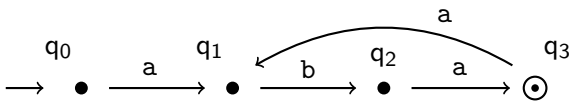
O próximo passo é construir um autômato que reconhece essas palavras.

Mas isso é bem fácil.

Quer dizer, nós começamos com um caminho que tem os símbolos do padrão

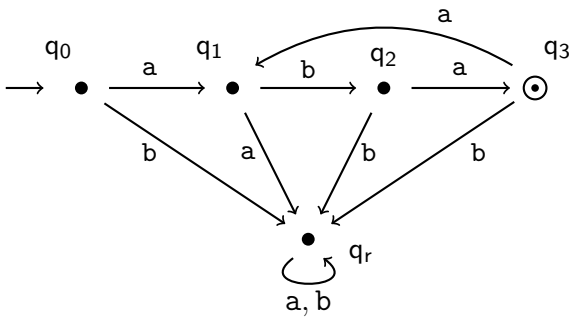


E daí, para permitir a leitura de múltiplas cópias do bloco `aba`, nós formamos um laço



Note que a leitura de qualquer quantidade de blocos `aa` deixa o autômato no estado q_3 onde ele dá a resposta SIM.

Finalmente, para completar a construção, nós colocamos as transições que correspondem à leitura de qualquer coisa diferente de `aba`



A seguir nós vamos ver outros exemplos.

Exemplos

a. Concatenação vazia

Quando nós trabalhamos com a operação de concatenação, faz sentido considerar o caso da concatenação vazia onde a palavra é formada por zero blocos de um determinado tipo.

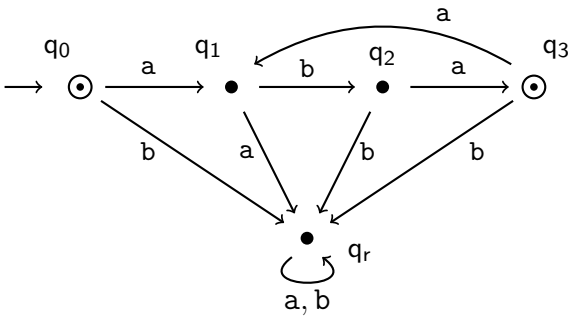
Nesse caso, o que nós obtemos é a palavra vazia, que não contém nenhum símbolo, e é denotada por `e` — (de *empty*, em inglês)

Examinando o autômato que foi construído acima, nós observamos que ele não aceita a palavra vazia.

Porque ao receber a palavra vazia o autômato não sai do estado inicial q_0, e o estado q_0 não é um estado final.

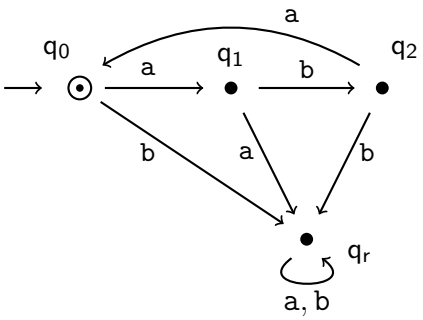
Mas isso é um problema bem fácil de resolver.

Quer dizer, basta marcar q_0 como um estado final



Por outro lado, quando a gente olha para esse diagrama, a gente percebe que é possível fazer uma pequena esperteza.

Quer dizer, nós podemos transformar os estados q_0 e q_3 em um único estado, para obter um autômato um pouco menor



Legal, não é?

◇

b. Concatenação com dois blocos

Considere a seguinte tarefa

→ *Aceitar as palavras formadas pela concatenação de blocos* `baa` *e* `bab`

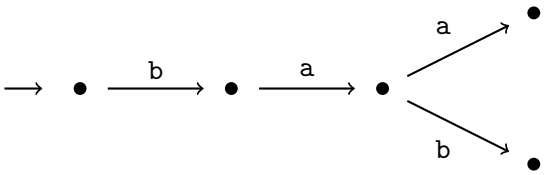
Quer dizer, agora cada bloco da palavra pode ser do tipo `baa` ou do tipo `bab`

Por exemplo

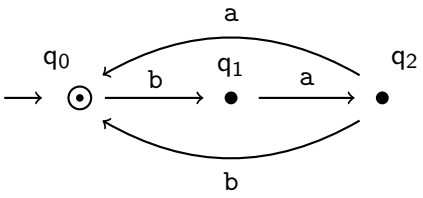
`baa`, `baabab`, `babbabbab`, ...

Para reconhecer essas palavras o autômato precisa ter um caminho correspondendo a cada padrão

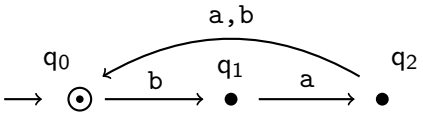
Como os padrões tem os dois primeiros símbolos iguais, os caminhos podem compartilhar as duas primeiras transições



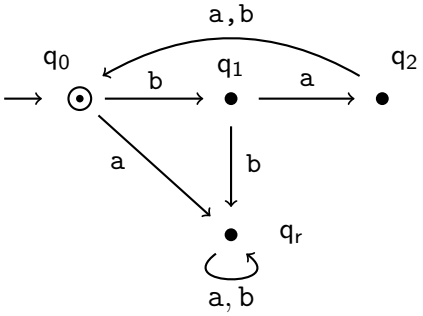
Daí, para permitir a leitura de múltiplos blocos, nós fazemos com que a leitura do último símbolo do padrão traga o autômato de volta para o estado inicial



De fato, as duas transições de q_2 para q_0 podem ser representadas de maneira mais sucinta assim



Finalmente, basta acrescentar o estado de rejeição para completar a construção



◇

c. Blocos de tamanhos diferentes

Agora considere a tarefa

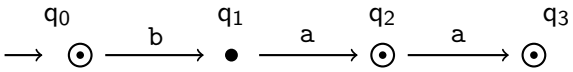
→ *Aceitar as palavras formadas pela concatenação de blocos* `ba` *e* `baa`

A novidade aqui são os blocos de tamanho diferente.

Mas isso não trás nenhuma dificuldade nova para a construção.

Quer dizer, nós começamos com caminhos que correspondem a esses dois padrões

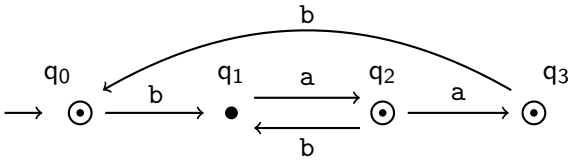
Daí, nós combinamos os dois caminhos em um único caminho



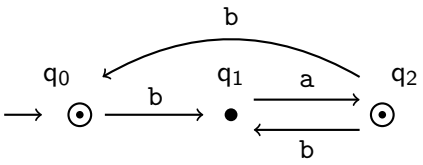
Nesse ponto, nós observamos que

- a leitura de um `a` em q_2 indica que o bloco é do tipo `baa` — (*e isso nos leva para* q_3)
- a leitura de um `b` em q_2 indica que um bloco `ba` chegou ao fim, e nós já começamos a leitura do próximo bloco — (*o que nos leva para* q_1)

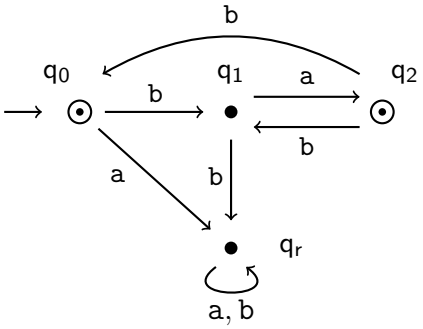
Com base nessas observações, nós aumentamos o diagrama para



O que pode ser simplificado para isso aqui



Finalmente, basta acrescentar o estado de rejeição para completar a construção



◇

d. Indeterminação

Agora considere a tarefa

→ *Aceitar as palavras formadas pela concatenação de blocos* `ba` *e* `bab`

A novidade desse exemplo é que após a leitura do par de símbolos `ba`, nós sempre encontramos o

símbolo `b` — (*se a palavra tem a propriedade*)

Só que esse `b` pode corresponder a

- o início do próximo bloco
- ou o fim de um bloco `bab`

O que fazer nesse caso?

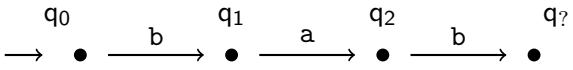
Bom, a ideia é olhar o símbolo que vem depois desse `b`.

Daí,

→ Se esse símbolo for um `a`, então o `b` era o início de um novo bloco

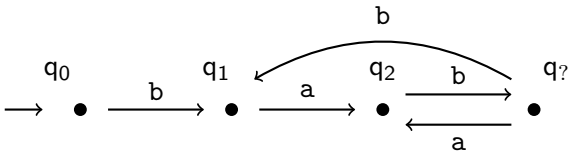
→ Se esse símbolo for um `b`, então o `b` (anterior) era o fim de um bloco `bab`

Para implementar essa lógica, nós podemos utilizar um estado de dúvida

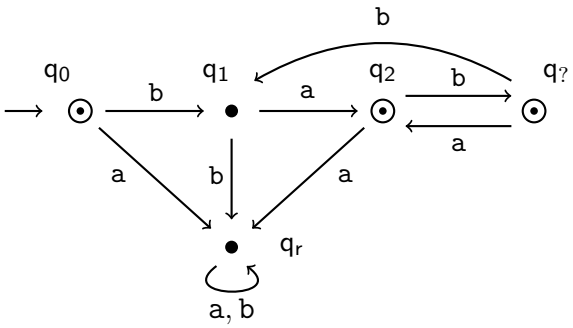


Quer dizer, quando nós chegamos em $q_?$, nós examinamos o próximo símbolo para saber o que fazer.

Daí, com base nas observações acima, nós aumentamos o diagrama da seguinte maneira



Finalmente, basta acrescentar o estado de rejeição para completar a construção



◇

e. Blocos de tamanho variável

Agora considere a tarefa

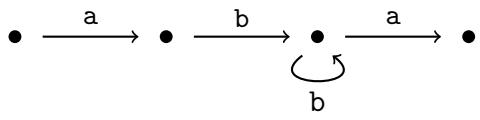
- *Aceitar as palavras formadas pela concatenação de blocos da forma* $\texttt{a}\underbrace{\texttt{b}..\texttt{b}}_{>0}\texttt{a}$

A novidade desse exemplo é que dessa vez existe uma inifinidade de blocos diferentes

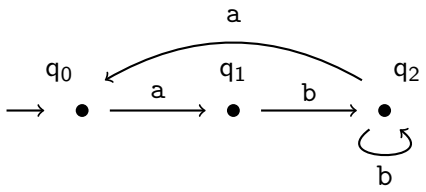
$$\texttt{aba}, \quad \texttt{abba}, \quad \texttt{abbba}, \quad \ldots$$

Mas os blocos tem todos a mesma forma.

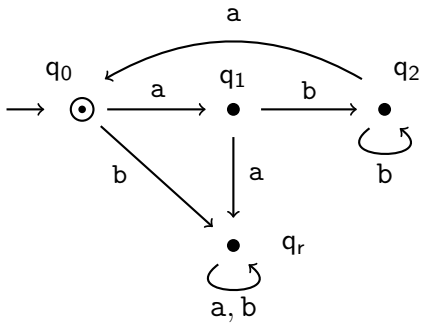
Daí que nós podemos usar um laço para ler qualquer um deles



O próximo passo é colocar um laço mais externo para ler vários blocos em sequência



Finalmente, basta acrescentar o estado de rejeição para completar a construção



◇