Teoria dos Autômatos e Linguagens Formais

aula 13: O autômato bidirecional

1 Introdução

Há duas aulas atrás, nós dissemos que os autômatos não podem implementar o esquema de *backtracking* porque eles não podem mover a sua cabeça de leitura para trás.

Mas, quem disse que eles não podem?

Quer dizer, os autômatos são uma máquina que a gente inventou.

E a gente pode modificar essa invenção na hora que a gente quiser.

A gente já fez isso quando adicionou as funcionalidades de não-determinismo e transição vazia.

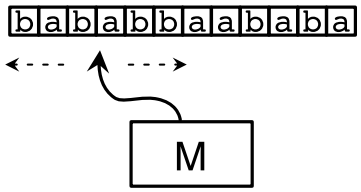
E viu que isso não aumentava o poder computacional dos autômatos.

Agora, a gente vai adicionar a capacidade de mover a cabeça de leitura para trás.

E o objetivo dessa aula é verificar se isso aumenta ou não o poder computacional dos autômatos.

2 O autômato bidirecional

O *autômato bidirecional* é um autômato que pode mover a cabeça de leitura para frente e para trás



Do ponto de vista do controle, a primeira novidade é que agora a movimentação da cabeça de leitura não é mais autômatica.

Quer dizer, como agora existem duas opções de movimento, é o próprio autômato quem deve determinar, a cada passo, se ele quer mover a cabeça de leitura para frente ou para trás.

Na prática, isso é feito adicionando um novo parâmetro às transições para indicar a direção do movimento.

Por exemplo,

$$q_i \xrightarrow{a,\,>} q_j \qquad\qquad q_i \xrightarrow{b,\,<} q_k$$

E do ponto de vista computacional, essa pequena mudança tem uma consequência muito significativa, porque

→ *os autômatos bidirecionais podem entrar em um* loop infinito*,*
e quando isso acontece a sua computação não acaba nunca

Por exemplo, o par de transições

$$q_i \xrightarrow{a,\,>} q_j \qquad\qquad q_j \xrightarrow{a,\,<} q_i$$

pode levar a um loop infinito, na situação em que a palavra de entrada tem 2 a's consecutivos.

Na prática, isso significa que é o autômato quem determina o momento em que a computação acaba.

Por conveniência, nós vamos adotar as seguintes convenções para os autômatos bidirecionais

1. Além da palavra de entrada, a fita também contém os símbolos delimitadores $\vdash$ e $\dashv$

$\vdash$	a	b	a	b	b	a	a	b	$\dashv$

2. A cabeça de leitura não se move quando nós tentamos andar para trás sobre o símbolo $\vdash$, ou quando nós tentamos andar para frente sobre o símbolo $\dashv$.
3. A computação sempre começa com a cabeça de leitura posicionada sobre sobre o símbolo inicial $\vdash$
4. O autômato sempre deve posicionar a cabeça de leitura sobre o símbolo final $\dashv$ antes de encerrar a sua computação.
5. O autômato possui apenas um estado final q_F, e apenas um estado de rejeição q_R, e a computação sempre deve terminar em um desses dois estados.

A seguir, nós vamos ver alguns exemplos de autômatos bidirecionais.

Exemplos

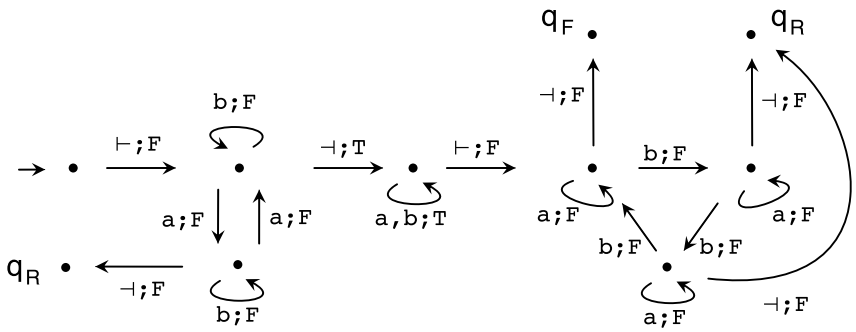
a. Duas tarefas

Considere a propriedade

→ palavras que possuem uma quantidade par de a's e uma quantidade de b's múltipla de 3

Reconhecer essa propriedade é uma tarefa fácil para os autômatos bidirecionais:

- primeiro, ele percorre a palavra verificando a condição associada aos a's
- depois, ele retorna a cabeça de leitura para o início da palavra
- finalmente, ele percorre a palavra mais uma vez, verificando a condição associada aos b's



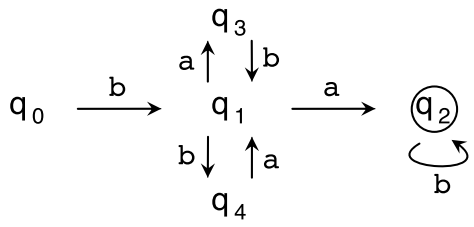
◇

b. Backtracking

Considere a expressão regular

$$\mathtt{b}\big(\mathtt{ab} \cup \mathtt{ba}\big)^*\mathtt{ab}^*$$

Abaixo nós temos um autômato não-determinístico que aceita as palavras descritas por essa expressão regular



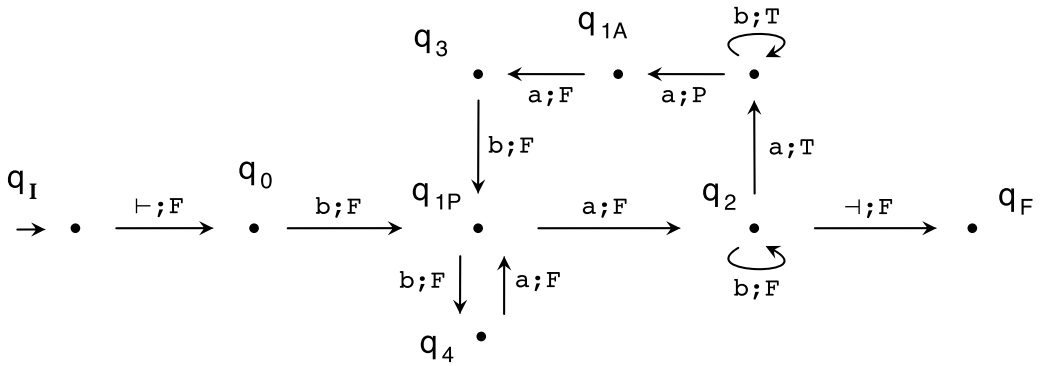
E agora, com os autômatos bidirecionais, nós podemos implementar a estratégia de backtracking para eliminar o não-determinismo.

A ideia é a seguinte

- Quando nós estamos no estado $\mathtt{q}_1$ e aparece um símbolo $\mathtt{a}$, nós fazemos a opção de seguir para o estado $\mathtt{q}_2$
- Mais adiante, caso apareça um outro símbolo $\mathtt{a}$, nós descobrimos que a decisão foi equivocada, e voltamos para o estado $\mathtt{q}_1$, retornando a cabeça de leitura para o símbolo $\mathtt{a}$ anterior

Para implementar essa ideia, nós vamos utilizar duas cópias do estado $\mathtt{q}_1$: uma associada ao comportamento padrão (i.e., seguir para $\mathtt{q}_2$ quando aparece um $\mathtt{a}$), e a outra associada ao comportamento alternativo (i.e., seguir para $\mathtt{q}_3$ quando aparece um $\mathtt{a}$).

Abaixo nós temos o diagrama que implementa essa ideia.



Nesse diagrama, nós utilizamos o parâmetro $\mathtt{P}$ (na transição chegando em $\mathtt{q}_{1\mathtt{A}}$) que mantém a cabeça de leitura parada no mesmo lugar, para deixar a solução mais elegante.

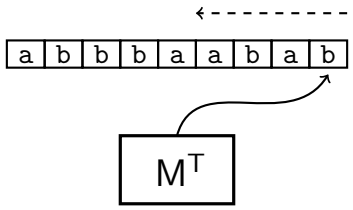
◇

3 Andando para trás

Nós vamos começar a nossa investigação dos autômatos bidirecionais com a seguinte observação

→ *andar para trás é a mesma coisa que andar para frente*

Quer dizer, imagine que em algum lugar do mundo, alguém teve a ideia de definir os autômatos com a cabeça de leitura cabeça de leitura andando para trás



Mas isso não faz a menor diferença.

Quer dizer, na prática o autômato só está lendo a palavra de trás para frente.

E na aula passada a gente viu que

$$\mathsf{M} \text{ reconhece } \mathsf{L} \quad \Rightarrow \quad \text{existe } \mathsf{M^R} \text{ reconhece } \mathsf{L^R} \text{ (andando p/ frente)}$$

Mas, reconhecer $\mathsf{L^R}$ andando para frente é a mesma coisa que reconhecer L andando para trás.

Logo, existe um autômato $\mathsf{M^T}$ (que anda p/ trás) que reconhece a linguagem L.

Essa observação nos permite concluir que

① *Tudo o que os autômatos que andam p/ frente fazem os autômatos que andam p/ trás também podem fazer*

Mas, o contrário também é verdade.

Quer dizer, imagine que você tem um autômato $\mathsf{M^T}$ que reconhece a linguagem L (andando p/ trás).

Ora, mas isso é a mesma coisa que reconhecer a linguagem $\mathsf{L^R}$ andando p/ frente.

Daí, usando o resultado da aula passada outra vez, nós obtemos

$$\underset{\text{(andando p/ trás)}}{\mathsf{M^T} \text{ reconhece } \mathsf{L^R}} \quad \Rightarrow \quad \underset{\text{(andando p/ frente)}}{\text{existe } \mathsf{M} \text{ reconhece } \left(\mathsf{L^R}\right)^{\mathsf{R}} = \mathsf{L}}$$

Logo, existe um autômato que reconhece a linguagem L (andando p/ frente).

Daí que,

② *Tudo o que os autômatos que andam p/ trás fazem os autômatos que andam p/ frente também podem fazer*

E agora, diante de ① e ②, nós podemos concluir que

→ *andar para trás é a mesma coisa que andar para frente*

Legal.

Mas, isso não é totalmente verdade.

Quer dizer, ser capaz de fazer a mesma coisa não é a mesma coisa que fazer a coisa da mesma maneira.

E agora a coisa ficou enrolada ...

Mas a ideia é simples.

Quer dizer, nós já sabemos que

$$\underset{\text{(andando p/ trás)}}{\mathsf{M}^{\mathsf{T}} \text{ reconhece } \mathsf{L}} \quad \Rightarrow \quad \underset{\text{(andando p/ frente)}}{\text{existe } \mathsf{M}^{\mathsf{F}} \text{ reconhece } \mathsf{L}}$$

Mas, M^{F} não é determinístico — (*porque* $\mathsf{M}^{\mathsf{F}} = \left(\mathsf{M}^{\mathsf{T}}\right)^{\mathsf{R}}$)

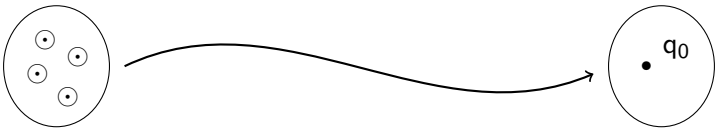
Daí que, M^{T} e M^{F} não funcionam da mesma maneira.

Mas, como é que o autômato M^{F} *funciona ?*

Bom, o estado inicial de M^{F} é a coleção de estados finais de M^{T}.

E o estado final de M^{F} contém o estado inicial de M^{T}.

Daí que, os caminhos de aceitação do autômato M^{F} são assim

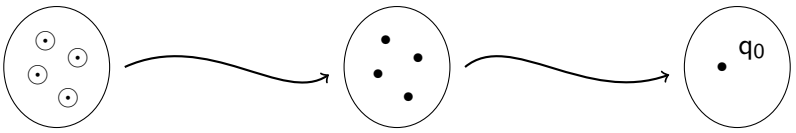


Quer dizer,

- M^{F} começa onde M^{T} deve terminar
- e M^{F} deve terminar aonde M^{T} começa

Mas o que acontece no meio do caminho ?

Bom, no meio do caminho M^{F} está em uma coleção de estados — (*porque ele é não-determinístico*)



E a observação chave é que esses são os estados onde M^{T} pode estar nesse ponto da palavra de entrada (andando p/ trás), para terminar a sua computação em um estado final.

Quer dizer, ao longo da computação, M^{F} vai calculando onde é que M^{T} poderia estar para que ele aceite a palavra de entrada.

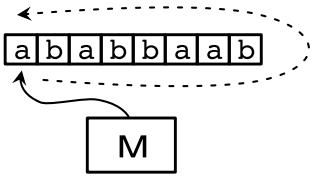
Daí que, no final da computação, se q_0 é um dos estados na coleção de M^{F}, então existe um caminho de aceitação de M^{T} que lê a palavra de entrada (de trás p/ frente).

4 O autômato vai-e-volta

A nossa próxima observação é a seguinte

→ *andar para a frente e depois para trás também é a mesma coisa que*
andar só uma vez para frente

Quer dizer, agora nós vamos examinar os autômatos *vai-e-volta*, que lêem a palavra de entrada uma vez andando para frente, e uma segunda vez andando para trás



Por exemplo, a gente pode reconhecer a seguinte linguagem dessa maneira

→ palavras que possuem uma quantidade par de a's
e uma quantidade de b's múltipla de 3

Quer dizer, basta verificar a condição associada aos a's na computação da ida, e verificar a condição associada aos b's na computação da volta.

Mas isso não faz a menor diferença.

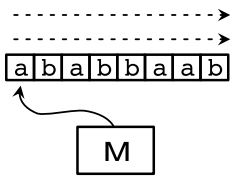
Quer dizer,

→ *Tudo o que os autômatos vai-e-volta fazem*
também pode ser feito por um autômato comum

A intuição é a seguinte.

Nós acabamos de ver que andar para trás é a mesma coisa que andar para frente.

Logo, nós podemos pensar no autômato vai-e-volta como um autômato *vai-vai*



Quer dizer, esse autômato anda duas vezes para frente: uma vez realizando a computação da ida, e a outra vez simulando a computação da volta.

Ora, mas andar duas vezes para frente é a mesma coisa que realizar duas computações sobre a mesma palavra de entrada.

E na aula passada, nós vimos que um autômato comum pode realizar duas computações simultâneas sobre a mesma palavra de entrada — (*por meio da técnica do autômato produto*)

Logo, nós sempre podemos construir um autômato comum que faz a mesma coisa que um autômato vai-e-volta.

Mas, essa ideia ainda não funciona ...

Quer dizer, no autômato vai-e-volta a computação da volta acontece depois da computação da ida.

Daí que, a computação da volta pode fazer coisas diferentes, dependendo do resultado da computação da ida.

E é por isso que a gente não pode fazer as duas computações ao mesmo tempo.

Ou será que pode?

Vamos raciocinar um pouquinho.

Quer dizer, de que maneira a computação da ida pode afetar a computação da volta?

Bom, o nosso autômato não escreve.

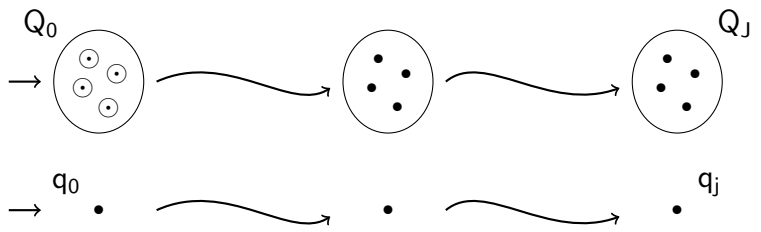
Logo, a palavra de entrada é sempre a mesma.

Mas, o estado em que a computação da ida termina é o estado em que a computação da volta começa.

E é assim que as duas se comunicam!

Certo.

Agora está na hora de visualizar como é que o autômato vai-vai funciona



Quer dizer, são duas linhas de execução

- a computação (determinística) da ida
- e o cálculo (não-determinístico) dos caminhos da volta que terminam em algum estado final

Quando nós chegamos ao final da palavra de entrada, nós descobrimos o estado q_j em que a computação da ida termina.

E esse também é o estado em que a computação da volta começa.

Por outro lado, Q_J é a coleção de estados onde a computação da volta deve começar para terminar em algum estado final.

Logo, o autômato vai-e-volta aceita a palavra de entrada quando $q_j \in Q_J$.

Essa é a verificação que o autômato vai-vai deve fazer.

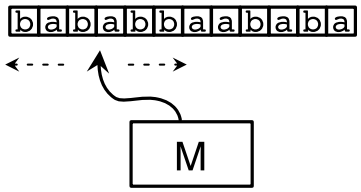
Mas se o autômato vai-vai pode fazer isso, então um autômato comum também pém fazer
— (*por meio da técnica do autômato produto*)

5 O autômato volta-e-vai

(. . .)

6 O autômato bidirecional (simulação)

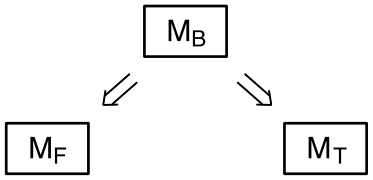
O autômato bidirecional, como nós já sabemos, é um autômato que pode mover a sua cabeça de leitura para frente e para trás



O controle da movimentação da cabeça de leitura é feito por meio de um parâmetro adicional associado às transições

$$q_i \xrightarrow{a,F} q_j \qquad\qquad q_i \xrightarrow{a,T} q_j$$

Em princípio, nós podemos imaginar a coleção de transições para frente e para trás como dois autômatos independentes: um autômato $\mathsf{M_F}$ que só anda para a frente, e outro autômato $\mathsf{M_T}$ que só anda para trás.

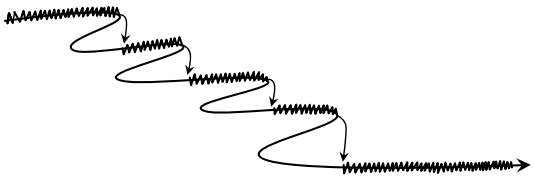


Desse ponto de vista, uma computação em $\mathsf{M_B}$ se desenrola como uma sequência alternada de computações em $\mathsf{M_F}$ e M_T.

A ideia da simulação consiste em realizar uma série de computações simultâneas em $\mathsf{M_F}$, e simular em paralelo uma série de computações em $\mathsf{M_T}$ (sempre andando com a cabeça de leitura para frente)

E de alguma maneira combinar esses segmentos de computação para determinar o resultado da computação bidirecional de $\mathsf{M_B}$

Mais especificamente, a simulação vai acontecer da seguinte maneira



Quer dizer, nós vamos ter uma linha de execução principal que realiza a trajetória indicada acima (com saltos).

E nós também vamos ter outras computações para frente e simulações de computações para trás acontecendo em paralelo, que vão fornecer as informações sobre como os saltos devem ocorrer.

Vejamos.

Os saltos na linha de execução principal acontecem no momento em que a cabeça de leitura começa a andar para trás — i.e., no momento em que uma transição do tipo $\mathsf{q}_\mathsf{i} \xrightarrow{\mathtt{s},\mathtt{T}} \mathsf{q}_\mathsf{j}$ está para ser realizada em M_B.

Quer dizer, os saltos acontecem no momento em que um segmento de computação para frente (em M_F) está se encerrando, o que corresponde também ao início de um segmento de computação para trás (em M_T).

A ideia é que esse segmento de computação para trás já vem sendo simulado (com a cabeça de leitura andando para frente), e que ele pode ser identificado pelo estado q_i onde ele começa.

Esse segmento de computação para trás se encerra no momento em que a cabeça de leitura começa a andar para frente novamente — i.e., no momento em que uma transição do tipo $\mathsf{q}_\mathsf{i} \xrightarrow{\mathtt{s},\mathtt{F}} \mathsf{q}_\mathsf{j}$ está para ser realizada em M_B.

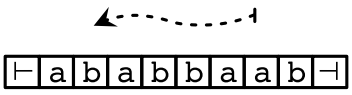
E, como no caso anterior, esse ponto corresponde também ao início de um segmento de computação para frente (em M_F).

De fato, é essa linha de execução para frente que vai indicar o destino do salto que acontece na linha de execução principal

Certo.

Vejamos agora os detalhes de como como a coisa vai funcionar.

A primeira observação importante é que, andando com a cabeça de leitura para frente, nós encontramos primeiro o ponto em que o segmento de computação para trás termina, e só mais adiante o ponto em que esse segmento de computação começa.



Nós já sabemos que um segmento de computação para trás termina no momento em que uma transição do tipo $q_i \xrightarrow{s,F} q_j$ está para ser realizada em M_B.

Mas, nós não temos como consultar a computação em M_B (ela não está acontecendo) para saber qual é a transição que será realizada em seguida.

Quer dizer, nós até sabemos qual é o símbolo da palavra de entrada nesse ponto, a ou b, mas nós não sabemos qual é o estado de M_B.

A ideia, então, é considerar todas as possibilidades.

Suponha que o símbolo da palavra de entrada nesse ponto é b (como na figura).

Daí, nós examinamos todos os estados de M_B que possuem uma transição

$$q_k \xrightarrow{b,F}$$

Cada um desses estados q_k é um lugar onde, nesse ponto da palavra de entrada, pode estar se encerrando um segmento de computação para trás.

Mas, se isso é o caso, então cada um desses estados q_k deve ser o lugar onde nós devemos estar iniciando a simulação de uma computação para trás.

Quer dizer, agora nós já sabemos o momento em que as simulações de computações para trás são iniciadas.

Mas, em princípio, não há como saber se essa simulação será útil mais adiante ou não — i.e., se ela vai se conectar com a linha de execução principal.

Em todo caso, nós vamos realizar todas elas em paralelo.

Nesse ponto, surge a questão sobre o número de simulações que estarão sendo realizadas em paralelo.

E aqui nós fazemos uma segunda observação importante:

- no momento em que nós estamos iniciando a simulação de uma computação para trás a partir de q_k, nenhuma outra simulação pode estar no estado q_k

A razão é fácil de entender.

Qualquer outra simulação de uma computação para trás teve início em um ponto anterior da palavra de entrada.

E esse ponto anterior onde a simulação começa é o lugar onde a computação para trás termina.

Mas, uma computação para trás que esteja em q_k no ponto atual não pode terminar mais atrás na palavra de entrada, pois q_k tem a transição

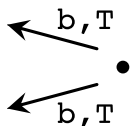
$$q_k \xrightarrow{b,F}$$

e logo não pode ter a transição

$$q_k \xrightarrow{b,T}$$

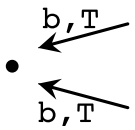
(pois M_B é determinístico).

Da mesma maneira, duas simulações de computações para trás não podem convergir para o mesmo estado, pois isso corresponde ao seguinte par de transições



o que não pode acontecer, pois M_B é determinístico.

Por outro lado, a simulação de uma computação para trás pode bifurcar em simulações que continuam em direções diferentes



o que corresponde a duas linhas de execução para trás diferentes que convergem para o mesmo estado.

Nesse ponto, a simulação se divide em duas simulações diferentes, uma para cada possível continuação.

Finalmente, pode acontecer da simulação de uma computação para trás não ter como continuar, porque não há uma transição para trás, associada ao símbolo atual da palavra de entrada, chegando no estado em que ela se encontra

Nesse ponto, a simulação é encerrada.

A figura abaixo ilustra a situação que nós descrevemos até o momento

— Figura —

Quer dizer

- nós temos uma linha de execução principal que foi interrompida em um certo ponto, e que está prestes a realizar um salto
- e nós temos uma série de simulações de computações para trás, que podem ter começado em pontos diferentes da palavra de entrada, e que se encontram em estados diferentes