Autômatos e Teoria da Computação

aula 21: Linguagens, gramáticas e autômatos de piha

1 Introdução

Na aula passada, nós apresentamos as gramáticas como uma ferramenta para descrever o padrão repetitivo recursivo de uma linguagem.

De fato, o termo *gramática* não foi escolhido por acaso.

E o termo *linguagem* também não!

Quer dizer, você já se deu conta de que uma linguagem como o português é uma coisa finita?

Qualquer bom dicionário contém todas as palavras do português — (*e um dicionário é um livro finito*)

Por outro lado, uma linguagem como o português também é uma coisa infinita.

Quer dizer, não existe nenhum livro que contenha todas as frases que podem ser formadas com as palavras do português.

E o mais interessante é que as frases do português não são formadas de maneira aleatória, juntando palavras do dicionário.

Quer dizer, as frases do português são as sequências de palavras que fazem sentido.

Por exemplo, essa frase faz sentido

João comeu o abacaxi que estava em cima da mesa

A seguinte frase é um pouco estranha, mas ela também faz algum sentido

O abacaxi comeu a mesa que estava em cima do João

Mas essa outra não faz nenhum sentido[1]

Abacaxi abacaxi a mesa mesa mesa estava abacaxi abacaxi

Agora, o mais interessante é que o português contém uma quantidade infinita de frases que fazem sentido — (*Você já se deu conta disso?*)

E, naturalmente, você está o tempo todo encontrando frases que fazem sentido e que você nunca tinha visto antes.[2]

Quer dizer, você consegue entender frases que você nunca ouviu antes, e consegue distinguir aquelas que fazem sentido daquelas que não fazem.

Como é que pode isso?

[1] As palavras reconhecidas pelos autômatos se parecem mais com essa última ...

[2] Você já tinha ouvido essa frase antes?

Bom, a resposta é que a infinidade de frases (com sentido) do português é gerada por meio de combinação, repetição e recursão, utilizando as suas finitas palavras.

Por exemplo,

- Combinação: Ivo viu a uva
- Repetição: João gosta de jaca e banana e goiaba e amendoim e ...
- Recursão: Bom, das duas uma; no primeiro caso isso ou aquilo, e no segundo caso das duas uma outra vez ...

A gramática de uma linguagem basicamente descreve os mecanismos de combinação, repetição e recursão que são usados nessa linguagem.

Na prática, quando a gente aprende a falar o português, a gente aprende a gramática do português ao mesmo tempo.

E daí, porque a gente já conhece os mecanismos de construção da linguagem, a medida que a gente vai lendo uma frase complexa (que a gente nunca viu), a gente vai entendendo o que está acontecendo.

— (*É como se a gente tivesse um autômato de pilha dentro da nossa cabeça ...*)

É assim que a coisa funciona.

2 Linguagens, gramáticas e autômatos de pilha

Na discussão acima, nós vimos que conhecer a gramática de uma linguagem nos permite distinguir as palavras (ou frases) que fazem parte da linguagem daquelas que não fazem.

Mas nós também dissemos que a gramática nos permite *entender* a palavra ou a frase.

O que significa isso?

Bom, entender uma coisa significa saber o que a gente pode fazer com ela.

Por exemplo, você entende o que são as expressões aritméticas se você sabe o que fazer quando você encontra uma delas.

E o que você deve fazer são as contas que ela indica.

Por exemplo, você mostra que você entende a expressão

$$2 \quad + \quad 5$$

quando você responde: 7.

E você mostra que você entende a expressão

$$(\,1 \;+\; 1\,) \;*\; 1$$

não apenas quando você responde 2, mas quando você explica que

- primeiro você realizou a operação $1 + 1$, que tem resultado 2
- e depois você realizou a operação $2 * 1$, que tem resultado 2

A ideia é que os autômatos de pilha podem não apenas distinguir entre as expressões que fazem e não fazem parte da linguagem, mas em certo sentido eles também entendem essas expressões.

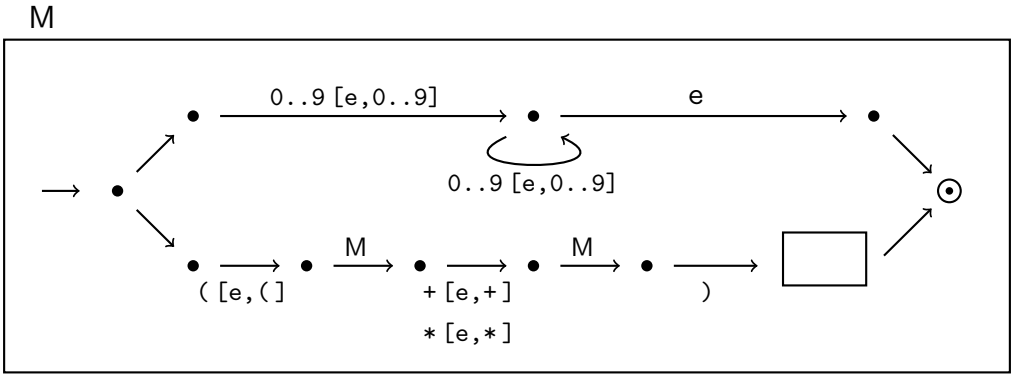
Quer dizer, eles podem guiar as manipulação que nós fazemos com essas expressões, o que corresponde a uma forma de entendimento.

Considere novamente a linguagem EXPR.

Nós já conhecemos o padrão recursivo dessa linguagem, que é descrito pela gramática abaixo

$$
\begin{aligned}
\mathsf{E} &\rightarrow (\mathsf{E} + \mathsf{E}) \mid (\mathsf{E} * \mathsf{E}) \\
\mathsf{E} &\rightarrow \mathsf{N} \\
\mathsf{N} &\rightarrow \mathsf{DN} \mid \mathsf{D} \\
\mathsf{D} &\rightarrow 0 \mid 1 \mid \ldots \mid 9
\end{aligned}
$$

E abaixo nós temos um autômato que reconhece essa linguagem



A primeira obervação é que esse é um autômato recursivo, mas nós já sabemos que é fácil trasformá-lo em um autômato de pilha, se quisermos.

A segunda observação é que dessa vez nós estamos colocando os números na pilha, e não apenas uma indicação de que ali aparece um operando.

A razão para isso é que dessa vez nós queremos fazer as contas, e não apenas distinguir entre expressões aritméticas bem formadas e mal formadas.

A terceira observação é que esse é um autômato não-determinístico.

Mas o não-determinismo desse autômato não é importante.

Quer dizer, dando uma espiadinha no próximo símbolo da expressão, nós sempre sabemos qual é caminho que nós devemos seguir.

Por exemplo, quando o autômato se encontra no estado inicial, basta verificar se o próximo símbolo é um dígito ou um abre-parênteses, para saber se ele deve seguir por cima ou por baixo.

A última observação diz respeito à caixinha que se encontra no final do caminho de baixo.

Nós chegamos ali sempre que nós lemos o fecha-parênteses.

A ideia é que nesse momento o topo da pilha sempre contém: um número, um operador e um número

E a caixinha indica que esse é o momento de fazer uma conta.

Quer dizer, lembre que o autômato de pilha vai apenas guiar as nossas manipulações.

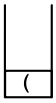
Quem vai realizar as contas somos nós.[3]

Por exemplo, considere essa expressão outra vez

```
( ( 1 + 1 ) * 1 )
```

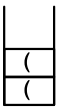
Quando o autômato começa a ler essa expressão, ele escolhe o caminho de baixo, lê (e empilha) o abre-parênteses, e realiza uma chamada recursiva

```
( 1 + 1 ) * 1 )
```



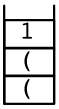
Daí, o autômato segue outra vez pelo caminho de baixo, lê (e empilha) o segundo abre-parênteses, e faz mais uma chamada recursiva

```
1 + 1 ) * 1 )
```



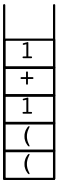
E daí, o primeiro operando é lido pelo caminho de cima e o autômato retorna, deixando a situação assim

```
+ 1 ) * 1 )
```



Bom, daí as coisas continuam da mesma maneira, até que nós chegamos ao primeiro fecha-parênteses

```
) * 1 )
```

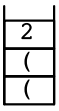


Aqui é chegada a hora de fazer a conta indicada no topo da pilha.

[3]Até porque o autômato de pilha não é capaz de realizar essas contas — (*porque?*)

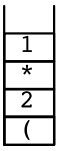
Esse trabalho é seu; você faz as contas, chega ao resultado 2, e coloca essa informação no topo da pilha

```
* 1 )
```



Daí, o autômato continua fazendo o trabalho dele, até que você é chamado novamente para fazer uma conta

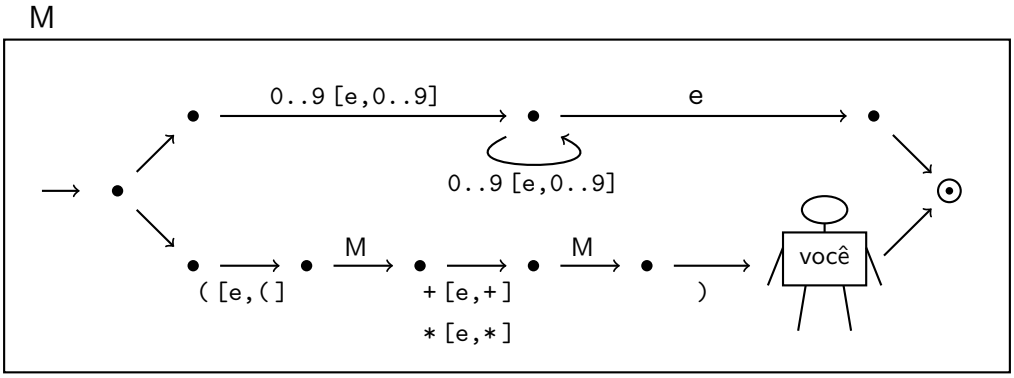
```
)
```



Esse exemplo mostra que, em certo sentido, o autômato de pilha entende o que é uma expressão aritmética.

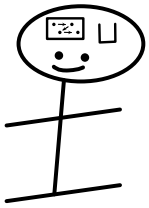
Ele só não sabe fazer as contas.

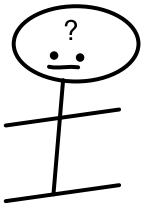
E é só para isso que ele precisa de você



Ou colocando as coisas de outro modo

→ *A diferença entre uma pessoa que entende expressões aritméticas*
e uma pessoa que sabe apenas fazer contas é só um autômato de pilha





3 Mais autômatos que entendem as coisas

A linguagem EXPR foi definida como o conjunto das expressões aritméticas completamente parentizadas (envolvendo as operações de + e *).

Mas, na prática, nós só usamos os parênteses quando necessário.

Quer dizer, nós assumimos que as operações de multiplicação têm precedência sobre as operações de soma, e nós só usamos os parênteses quando queremos mudar a ordem da realização das operações.

Mas isso signfica, então, que nós utilizamos uma outra linguagem com uma outra gramática.

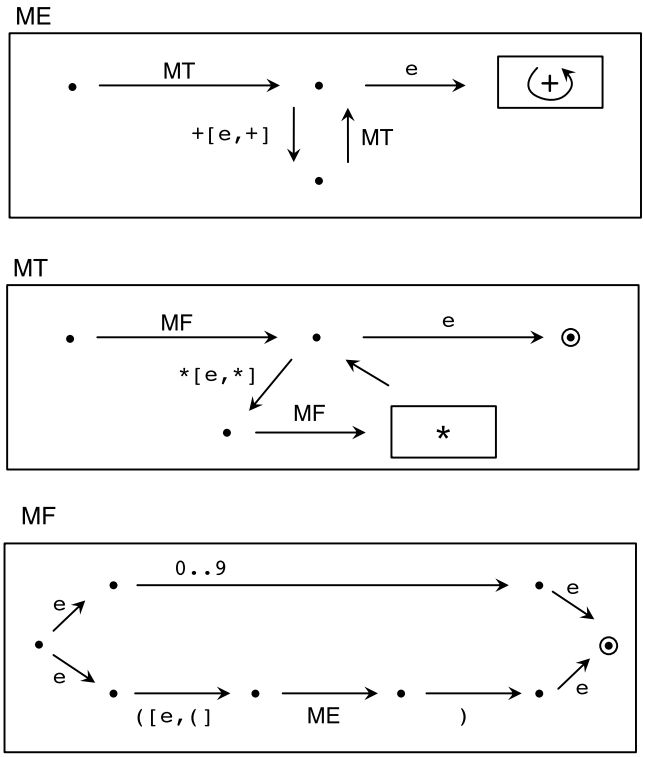
Para chegar a essa gramática, nós podemos pensar que

- uma expressão arimética é uma soma de termos (possivelmente apenas um)
- um termo é um produto de fatores (possivelmente apenas um)
- um fator é um número ou uma expressão aritmética entre parênteses

Abaixo nós temos a gramática que corresponde a esse esquema

$$\begin{aligned} E &\rightarrow T + E \mid T \\ T &\rightarrow F * T \mid F \\ F &\rightarrow N \mid (\,E\,) \end{aligned}$$

E agora, para construir um autômato recursivo que reconhece essa linguagem, basta definir um módulo para cada variável da gramática



Exemplos

a. BOOL

O autômato de pilha não consegue realizar as operações de soma e multiplicação com os operandos da pilha porque esses operandos podem ter uma quantidade arbitrária de dígitos, e o autômato só consegue acessar o primeiro dígito do operando que está no topo.

Mas a situação fica diferente quando os operandos possuem apenas um símbolo.

Esse é o caso das expressões booleanas, onde os operandos são sempre `0` ou `1`.

Defina a linguagem

BOOL = expressões booleanas completamente parentizadas (envolvendo as operações + e *)

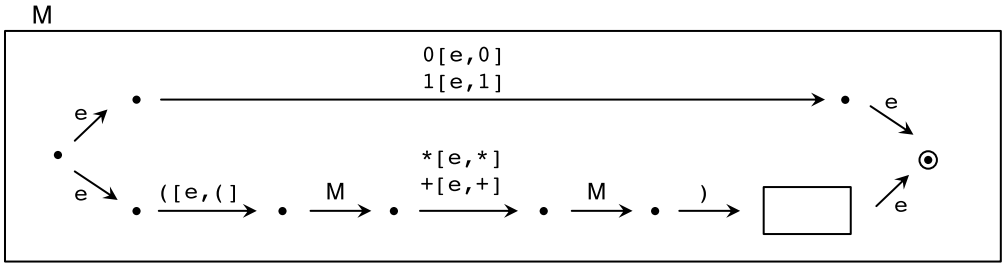
E lembre que a soma e a multiplicação booleanas correspondem ao E e OU lógicos

		+	*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

A gramática da linguagem BOOL é basicamente uma versão simplificada da gramática de EXPR

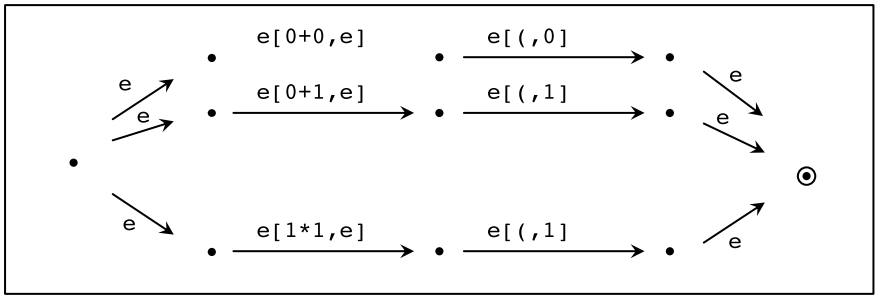
$$E \rightarrow (E + E) \mid (E * E) \mid B$$
$$B \rightarrow 0 \mid 1$$

E o autômato de pilha que reconhece (e calcula) as expressões de BOOL também é uma versão simplificada do autômato de EXPR



Mas a novidade é que esse autômato não precisa da sua ajuda para fazer as contas.

Quer dizer, como agora os operandos possuem apenas 1 bit, a seguinte lógica pode ser colocada na caixa



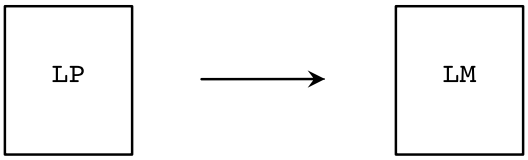
◇

b. LOOP

Por razões de eficiência e simplicidade, os computadores são construídos com a capacidade de reconhecer linguagens muito simples, as chamadas *linguagens de máquina*.

Mas, programar em linguagem de máquina é uma tarefa muito pouco conveniente para um ser humano.

A solução que foi encontrada para lidar com essa situação foi trabalhar com duas linguagens



- uma linguagem de máquina (LM) muito simples que o computador é capaz de entender
- uma linguagem de programação (LP) com uma gramática mais próxima da nossa linguagem natural

E o problema que aparece na prática, então, é que agora é preciso fazer a tradução da linguagem LP para a linguagem LM.

Mas, na medida em que a gramática da linguagem LP é uma gramática livre de contexto conhecida, é possível construir um autômato de pilha que *entende* o código escrito pelo programador e *explica* esse código para o computador.

Em outras palavras, nós podemos construir um autômato que traduz um programa escrito na linguagem LP para código escrito na linguagem LM.

Vejamos um exemplo concreto.

Lembre da linguagem LOOP que foi apresentada na aula 19, e que possui apenas dois tipos de instrução

```
X <-- <expressão>
```

```
LOOP X
   <cmd>
   ...
   <cmd>
ENDLOOP
```

A gramática da linguagem LOOP é a seguinte

P → C P | C

C → LOOP V P ENDLOOP

C → V := E

V → X | ... | Z

E → (...)

onde as regras da variável E são as regras de EXPR e foram omitidas.

E abaixo nós temos o autômato recursivo que reconhece programas escritos na linguagem LOOP

— Figura —