Notas de Aula de Teoria da Computação

Prof. Dr. Rodrigo Geraldo Ribeiro

23 de setembro de 2023

Conteúdo

Parte I

Linguagens regulares

1

Aula 1 - Introdução às linguagens formais

Objetivos

- Apresentar os conceitos centrais da teoria de linguagens: alfabeto, palavra, linguagem e problemas de decisão.
- Especificação de linguagens usando conjuntos.

1.1 Conceitos básicos

Definição 1 (Alfabeto)**.** Um alfabeto é qualquer conjunto finito não vazio. Usaremos a letra grega sigma, Σ, para representar um alfabeto qualquer. ■

Exemplo 1. Dois alfabetos importantes são $\Sigma_1 = \{1\}$ e $\Sigma_2 = \{0, 1\}$. Caso desejamos lidar com texto, um possível alfabeto é o UNICODE. ■

Definição 2 (Palavra)**.** Uma palavra sobre um alfabeto Σ é uma sequência finita formada por elementos de Σ. ■

Definição 3 (Tamanho de uma palavra, palavra vazia)**.** Seja w uma palavra qualquer sobre um alfabeto Σ. Denomina-se por tamanho de w, $|w|$, o número de símbolos de w. Em especial, existe apenas uma palavra de tamanho 0, e esta é representada pela letra grega λ. ■

Exemplo 2. São exemplos de palavras sobre $\Sigma = \{0, 1\}$: $0, 1, 00, 1101, \lambda$. São exemplos de palavras sobre $\Sigma = \{1\}$: $\lambda, 1, 11, ...$ Note que λ é uma palavra sobre qualquer alfabeto. ■

Definição 4 (Repetição)**.** Denotamos por a^n, em que $a \in \Sigma$, a palavra formada por n ocorrências do símbolo a. Formalmente:

$$\begin{array}{lcl} a^0 & = & \lambda \\ a^{n+1} & = & a^n a \end{array}$$

■

Exemplo 3. Considere $\Sigma = \{0, 1\}$. Temos que $0^3 = 000$, $1^4 = 1111$ e $1^0 = \lambda$. ■

Definição 5 (Concatenação)**.** Sejam x e y duas palavras sobre um alfabeto Σ. A concatenação de x e y é a simples justaposição de y ao fim da palavra x e é representada por xy. ■

Exemplo 4. Considere $\Sigma = \{0, 1\}$, $x = 001$, $y = 01$ e $z = 11$. Temos que:

- $xy = 00101$.
- $yx = 01001$.

- $\lambda x = x\lambda = x$.
- $(xy)z = 0010111$ e $x(yz) = 0010111$.

Note que a concatenação é uma operação associativa, isto é, para quaisquer palavras x, y e z temos que $x(yz) = (xy)z$. Além disso, λ é o elemento neutro da concatenação, isto é, para qualquer palavra x, $x\lambda = \lambda x = x$. ■

Definição 6 (Concatenação repetida)**.** Seja w uma palavra qualquer sobre um alfabeto Σ. A concatenação repetida de w, w^n, é definida por recursão sobre n como:

$$\begin{array}{lcl} w^0 & = & \lambda \\ w^{n+1} & = & w^n w \end{array}$$

■

Exemplo 5. Alguns exemplos da operação de concatenação repetida:

- $(01)^3 = 010101$; $(110)^0 = \lambda$ e $(110)^3 = 110110110$.

■

1.2 Linguagens e operações

Definição 7 (Linguagem)**.** Seja Σ um alfabeto. Denominamos por linguagem um conjunto de palavras sobre Σ. ■

Exemplo 6. Como linguagens são conjuntos de palavras, existem linguagens finitas e infinitas. Além disso, podemos nos valer de diversas operações sobre conjuntos para definir linguagens.

- $\emptyset$, é uma linguagem sobre qualquer alfabeto.
- $\{0, \lambda\}$ é uma linguagem sobre $\Sigma = \{0, 1\}$.
- $\{(01)^n \,|\, n \geq 0\}$ é linguagem de palavras formadas por sequencias sobre 01.
- $\{1^n | n \geq 1\}\,\{0^n | 1 \leq n \leq 10\}$ é a linguagem formada por um prefixo de $n \geq 0$ 1's seguido de um sufixo $1 \leq p \leq 10$ 1's.

■

Definição 8 (Outras operações sobre palavras)**.** Seja $w = a_1a_2...a_n$ uma palavra sobre um alfabeto Σ. Denomina-se *reverso* de w, w^R, a palavra $w^R = a_na_{n-1}...a_2a_1$. Em especial, diz-se que uma palavra w é um *palíndromo* se $w = w^R$. Seja $w = xyz$, em que tanto x, y e z podem ser λ ou não. Dizemos que y é uma *subpalavra* de w, x é um *prefixo* de w e z é um *sufixo* de w. Em particular, tanto λ quanto w são prefixo, sufixo e subpalavra de w. ■

Exemplo 7. Seja $\Sigma = \{0, 1\}$. Temos que $(100)^R = 001$ e a palavra 101 é um palíndromo, visto que $101 = (101)^R$. Seja $w = 010$. Temos que $\lambda, 0, 01$ e 010 são prefixos de w. Por sua vez, $\lambda, 0, 10$ e 010 são sufixos de w. As subpalavras de w são $\lambda, 0, 1, 01, 10, 010$. ■

Definição 9 (Concatenação de linguagens)**.** Sejam L_1 e L_2 duas linguagem sobre um alfabeto Σ. Denotamos a concatenação de L_1 com L_2, L_1L_2, como o seguinte conjunto:

$$L_1L_2 = \{w_1w_2 \,|\, w_1 \in L_1 \wedge w_2 \in L_2\}$$

■

Exemplo 8. Seja $\Sigma = \{0, 1\}$, $L_1 = \{0, 01, 11\}$ e $L_2 = 1, \lambda$. Temos que L_1L_2 é o conjunto

$$L_1L_2 = \{01, 0, 011, 111, 11\}$$

Um outro exemplo: Considere $\{0, 1\}\{0, 1\}$ é igual a:

$$\{0, 1\}\{0, 1\} = \{00, 01, 10, 11\}$$

■

Definição 10 (Concatenação repetida de linguagens)**.** Seja L uma linguagem sobre um alfabeto Σ. A concatenação repetida de L, L^n, é definida por recursão sobre n como:

$$\begin{aligned} L^0 &= \{\lambda\} \\ L^{n+1} &= L^n\, L \end{aligned}$$

■

Exemplo 9. Seja $L = \{01, 0\}$ uma linguagem sobre $\Sigma = \{0, 1\}$. Temos que:

$$\begin{aligned} L^2 &= L^{2-1}L \\ &= L^1 L \\ &= (L^0 L)L \\ &= (\{\lambda\}L)L \\ &= (\{\lambda\}\{01, 0\})\{01, 0\} \\ &= \{01, 0\}\{01, 0\} \\ &= \{0101, 010, 001, 00\} \end{aligned}$$

Ainda considerando $\Sigma = \{0, 1\}$, temos que Σ^3 é definido como:

$$\begin{aligned} \Sigma^3 &= \Sigma^2\Sigma \\ &= (\Sigma^1\Sigma)\Sigma \\ &= ((\Sigma^0\Sigma)\Sigma)\Sigma \\ &= ((\{\lambda\}\Sigma)\Sigma)\Sigma \\ &= (\Sigma\Sigma)\Sigma \\ &= (\{0, 1\}\{0, 1\})\{0, 1\} \\ &= (\{00, 01, 10, 11\})\{0, 1\} \\ &= \{000, 001, 010, 011, 100, 101, 110, 111\} \end{aligned}$$

Observe que Σ^n produz o conjunto de todas as palavras de tamanho n para um alfabeto Σ. ■

Definição 11. Fecho de Kleene Seja L uma linguagem sobre um alfabeto Σ. Definimos o fecho de Kleene de L, L^*, como:

$$L^* = \bigcup_{n \in \mathbb{N}} L^n$$

Isto é, L^* é a união de todos os L^n para $n \in \mathbb{N}$. Outra forma de definir L^* é usando uma definição recursiva como a seguinte:

- $\lambda \in L^*$.
- Se $x \in L^*$ e $y \in L$ então $xy \in L^*$.

Em particular, define-se o fecho positivo de Kleene de L, L^+, como

$$L^+ = \bigcup_{n \in \mathbb{N} - \{0\}} L^n$$

Finalmente, note que $L^* = L^+ \cup \{\lambda\}$. ■

Exemplo 10. Segue alguns exemplos do fecho de Kleene de algumas linguagens.

- $\emptyset^* = \{\lambda\}$ e $\emptyset^+ = \emptyset$.
- $\{\lambda\}^* = \{\lambda\}^+ = \{\lambda\}$.
- $\{1\}^* = \{1^n \mid n \in \mathbb{N}\}$.
- $\{\lambda, 00, 11\}^* = \{\lambda\} \cup \{00, 11\}^*$.

Em particular, Σ^* denota o conjunto de todas as palavras, de todos os tamanhos para um certo alfabeto Σ. Além disso, podemos dizer que L é uma linguagem sobre um alfabeto Σ simplesmente especificando que $L \subseteq \Sigma^*$. ■

1.3 Especificando linguagens usando conjuntos

A seguir apresentamos alguns exemplos ilustrando o uso de operações e notações da teoria de conjuntos e operações recém definidas sobre linguagens para especifar novas linguagens.

Exemplo 11. A seguir apresentamos uma descrição textual de uma linguagem e em seguida apresentamos sua descrição como conjunto.

- Conjunto de palavras que começam com 0: $\{0\}\{0,1\}^*$. Observe que $\{0,1\}^* = \{\lambda, 0, 1, 00, 01...\}$. Logo, Concatenando a linguagem $\{0\}$ a $\{0,1\}^*$ irá produzir $\{0\}\{0,1\}^* = \{0, 00, 01, 000, ...\}$, como requerido.
- Conjunto de palavras que terminam em 00: $\{0,1\}^*\{00\}$.
- Conjunto de palavras que contém 00 ou 11: $\{0,1\}^*\{00,11\}\{0,1\}^*$.
- Conjunto de palavras de tamanho par: $(\{0,1\}^2)^*$.
- Conjunto de palavras de tamanho par que começam com 0: $(\{0,1\}^2)^* \cap \{0\}\{0,1\}^*$.

■

1.4 Problemas de decisão

Definição 12 (Problema de decisão)**.** Um problema de decisão é uma questão que faz referência a um conjunto finito de parâmetros, e que, para valores específicos dos parâmetros, tem como resposta sim ou não. ■

Exemplo 12. A seguir apresentamos alguns exemplos de problemas de decisão.

- Determinar se $n \in \mathbb{N}$ é um número primo.
- Determinar se um grafo G possui um ciclo.
- Determinar se um grafo possui um ciclo hamiltoniano.
- Determinar se um programa pára com uma determinada entrada.

■

1.5 Exercícios

1.5.1 Exercícios de fixação

1. Apresente uma definição recursiva para o fecho positivo de Kleene para uma linguagem L.
2. Apresente definições, usando operações de conjuntos, para as seguintes linguagens. Considere $\Sigma = \{0,1\}$.
   (a) Palavras que contém pelo menos um 0.
   (b) Palavras de tamanho ímpar.
   (c) Palavras que não possuem nenhuma ocorrência de 00.
   (d) Palavras em que todo 0 é seguido por pelo menos dois 1's. Exemplo: $\lambda, 1, 1111, 011, 11101101111, ...$
3. Apresente definições recursivas para as seguintes linguagens.
   (a) $\{0\}^*\{1\}^*$.
   (b) $\{0^n 1^n \mid n \in \mathbb{N}\}$.

1.5.2 Exercícios para tutoria

1. Sejam $A = \{00, 11\}$, $B = \{01, 10\}$ e $C = \{\lambda, 1\}$. Calcule as seguintes linguagens.
   (a) AA
   (b) CCC
   (c) AC
   (d) ABC
   (e) $A(B \cup C)$
2. Sejam $\Sigma = \{0, 1\}$, $A = \Sigma^*\{0\}$ e $B = \{1\}\Sigma^*$. Descreva, em português, as linguagens a seguir.
   (a) $A \cup B$
   (b) $A \cap B$
   (c) $A - B$
   (d) AB
3. Descreva mais formalmente as seguintes linguagens a seguir sobre $\{0, 1\}$:
   (a) Palavras com, no mínimo, um 0.
   (b) Palavras com, no máximo, um 0.
   (c) Palavras de tamanho ímpar.
   (d) Palavras com um prefixo de um ou mais 0s seguidos de um sufixo de zero ou mais 1s.
   (e) Palavras com uma quantidade par de 0s.
   (f) Palavras com uma quantidade ímpar de 1s.
4. Descreva, em português, as seguintes linguagens sobre $\{0, 1\}$:
   (a) $\{0, 1\}^*\{1\}\{0, 1\}$.
   (b) $\{0\}\{0, 1\}^* \cup \{0, 1\}^*\{1\}$.
   (c) $\{0, 1\}^*\{01\}\{11\}^*$.
   (d) $\{01, 1\}^*$.
   (e) $\{\lambda, 1\}(\{0\}\{0\}^*\{1\})^*\{0\}^*$.

2

Aula 2 - Autômatos finitos determinísticos

Objetivos

- Apresentar o conceito de autômato finito determínistico e exemplos dessa definição.
- Apresentar a definição de linguagem aceita por um autômato e equivalência entre autômatos finitos determínisticos.

2.1 Um exemplo introdutório

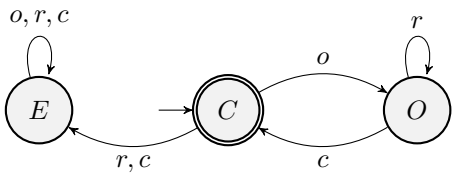
Exemplo 13 (Modelando manipulação de arquivos)**.** Considere a tarefa de modelar o comportamento de uma biblioteca de leitura de arquivos em sua linguagem de programação predileta. Idealmente, essa deve conter funções para abrir um arquivo, ler conteúdo de um arquivo e fechar um arquivo.

Uma maneira de modelarmos tal biblioteca seria usando um grafo, em que os vértices representam as possíveis configurações de um arquivo. Em um dado instante de tempo, um arquivo pode estar em uma de duas possíveis configurações: aberto ou fechado. A transição entre as possíveis configurações é feita por funções da biblioteca, a saber:

- `open(f)`: Abre o arquivo `f`.
- `read(f,v)`: Lê uma parte do conteúdo do arquivo `f` e o armazena em `v`.
- `close(f)`: fecha o arquivo `f`.

Evidentemente, algumas configurações não são válidas. Por exemplo, chamar a operação `read(f,v)` sobre um arquivo não aberto resulta em um erro em tempo de execução. Além disso, não fechar um arquivo após a execução pode ocasionar problemas de desempenho. Logo, além de nós para representar as configurações relativas ao arquivo estar aberto ou fechado devemos ter um nó para representar todas as configurações inválidas. Dessa forma, nosso modelo cobrirá todas as possibilidades de uso dessa biblioteca. Representaremos o estado fechado pela letra C, aberto por O e o estado de configuração inválida por E.

Mas o que causará a mudança dessas configurações? Observe que a função `open(f)` faz com que um arquivo fechado passe para o estado aberto, a função `read(f,v)` não altera a configuração de um arquivo e a função `close(f)` possui significado óbvio. Visando simplificar a escrita de nosso modelo, utilizaremos a letra o para representar a função `open(f)`, r para `read(f,v)` e c para `close(f)`. Abaixo apresentamos o diagrama que modela o comportamento descrito. Note que o diagrama acima modela o comportamento esperado da biblioteca. Antes de lermos informações presentes em um arquivo, este deve estar aberto e todo uso da biblioteca deve ser finalizado com o fechamento de arquivo. Dessa forma, o comportamento correto da biblioteca pode ser especificado pelo seguinte conjunto de palavras sobre o alfabeto $\Sigma = \{o, c, r\}$: $\{o\}\{r\}^*\{c\}$.



Existem tipos distintos de vértices nesse grafo. O primeiro é dito ser inicial e possui uma seta à sua esquerda. Intuitivamente, o processamento de uma palavra deve iniciar a partir desse vértice. Outro tipo de vértice é o final, representado com uma borda adicional (no exemplo, o mesmo vértice C é inicial e final). Dizemos que uma palavra representa uma computação válida nesse modelo de biblioteca se iniciando o processamento no vértice inicial, consumindo um símbolo ao passar por cada aresta, a computação da palavra termina em um estado final. Formalizaremos essa idéia a seguir.

O par $[e, w]$, em que e é um dos vértices do grafo anterior e $w \in \{o, c, r\}^*$ é chamado de *configuração instantânea* pois, representa um momento do processamento de uma palavra (ou estado da biblioteca de manipulação de arquivos). A palavra $orrc$ repreesenta uma computação válida, visto que seu processamento termina em um nó final.

$$[C, orrc] \vdash [O, rrc] \vdash [O, rc] \vdash [O, c] \vdash [C, \lambda]$$

Por sua vez, a palavra $orcrr$ termina em um estado não final (E), conforme a sequência de configurações abaixo.

$$[C, orcrr] \vdash [O, rcrr] \vdash [O, crr] \vdash [C, rr] \vdash [E, r] \vdash [E, \lambda]$$

A próxima seção apresentará definições precisas deste tipo de modelo e de sua respectiva computação. ■

2.2 Autômatos finitos determinísticos

Definição 13 (Autômato finito determinístico)**.** Um autômato finito determinístico (AFD) M é uma quíntupla $M = (E, \Sigma, \delta, i, F)$, em que:

- E: conjunto finito de estados.
- Σ: alfabeto
- $\delta : E \times \Sigma \to E$: função de transição, uma função total.
- $i \in E$: estado inicial
- $F \subseteq E$: conjunto de estados finais.

■

Exemplo 14. Considere o AFD apresentado anteriormente que modela o comportamento de uma biblioteca de manipulação de arquivos. Sua representação de acordo com a definição formal de um AFD é: $M = (S, \{o, c, r\}, \delta, C, \{C\})$, em que: $S = \{E, C, O\}$ e δ é tal que:

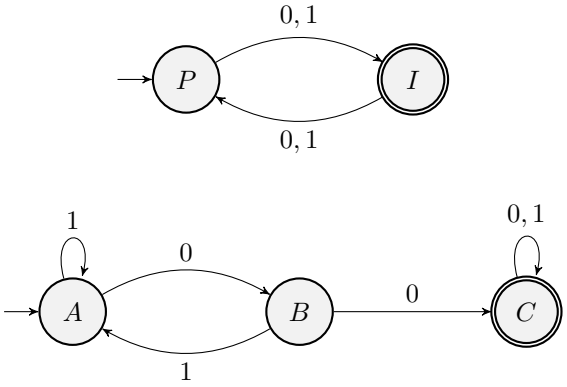
$$\begin{array}{ll} \delta(E, r) = E & \delta(E, c) = E \\ \delta(E, o) = E & \delta(C, r) = E \\ \delta(C, c) = E & \delta(C, o) = O \\ \delta(O, r) = O & \delta(O, c) = C \end{array}$$

Observe que cada aresta rotulada representa uma entrada na função de transição. ■

Exemplo 15. Considere a seguinte linguagem $L \subseteq \{0, 1\}^*$:

$$L = \{w \in \{0, 1\}^* \mid \exists k. |w| = 2k + 1\}$$

isto é, palavras de tamanho ímpar. O seguinte diagrama de estados (grafo dirigido para representar um AFD) aceita palavras pertencentes a essa linguagem. ■



Exemplo 16. Considere a seguinte linguagem sobre $\Sigma = \{0, 1\}$:

$$L = \{0, 1\}^*\{00\}\{0, 1\}^*$$

O seguinte AFD aceita as palavras pertencentes a esta linguagem.

■

2.2.1 Linguagem aceita por um AFD

Definição 14 (Função de transição estendida)**.** Seja $M = (E, \Sigma, \delta, i, F)$ um AFD qualquer. A função de transição estendida para M, $\widehat{\delta} : E \times \Sigma^* \to E$, é definida da seguinte maneira:

$$\begin{array}{lcl} \widehat{\delta}(e, \lambda) & = & e \\ \widehat{\delta}(e, ay) & = & \widehat{\delta}(\delta(e, a), y) \end{array}$$

em que $a \in \Sigma$ e $y \in \Sigma^*$. Informalmente, dado um estado $e \in E$ e $w \in \Sigma^*$, $\widehat{\delta}(e, w)$ retorna o estado em que o processamento da palavra w termina ao ser iniciado no estado e. ■

Exemplo 17. Considere o AFD do Exemplo 24. Temos que iniciando o processamento da palavra 010011 no estado inicial, este termina no estado final. Conforme apresentado a seguir pelo cálculo de $\widehat{\delta}(A, 010011)$.

$$\begin{array}{lcllcl} \widehat{\delta}(010011) & = & \widehat{\delta}(\delta(A, 0), 10011) & = & \widehat{\delta}(B, 10011) \\ \widehat{\delta}(\delta(B, 1), 0011) & = & \widehat{\delta}(A, 0011) & = & \widehat{\delta}(\delta(A, 0), 011) \\ \widehat{\delta}(B, 011) & = & \widehat{\delta}(\delta(B, 0), 11) & = & \widehat{\delta}(C, 11) \\ \widehat{\delta}(\delta(C, 1), 1) & = & \widehat{\delta}(C, 1) & = & \widehat{\delta}(\delta(C, 1), \lambda) \\ \widehat{\delta}(C, \lambda) & = & C & & \end{array}$$

De maneira similar, podemos realizar o cálculo de $\widehat{\delta}(A, 101)$.

$$\begin{array}{lcllcl} \widehat{\delta}(A, 101) & = & \widehat{\delta}(A, 01) & = & \widehat{\delta}(B, 1) \\ \widehat{\delta}(A, \lambda) & = & A & & \end{array}$$

A partir desses exemplos, podemos perceber que se i é o estado inicial de um AFD M e w é uma palavra sobre o alfabeto Σ de M, então w é aceita por M se $\widehat{\delta}(i, w)$ é um estado final e recusada, caso contrário. A definição seguinte formaliza essa noção. ■

Definição 15. Linguagem aceita por um AFD Seja $M = (E, \Sigma, \delta, i, F)$ um AFD qualquer. A linguagem aceita por M, $L(M)$, é definida como:

$$L(M) = \{w \in \Sigma^* \mid \widehat{\delta}(i, w) \in F\}$$

■

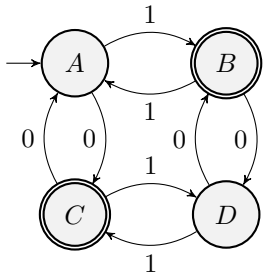
2.2.2 Relação entre configurações instantâneas de um AFD

Definição 16 (Configuração instantânea)**.** Seja $M = (E, \Sigma, \delta, i, F)$ um AFD qualquer. Denomina-se por configuração instantânea de M o par $[e, w]$ em que $e \in E$ e $w \in \Sigma^*$. ■

Definição 17 (Transição entre configurações instantâneas)**.** Seja $M = (E, \Sigma, \delta, i, F)$ um AFD qualquer, $[e, w]$ e $[e', w']$ duas configurações instantâneas de M. A relação $\vdash \subseteq (E \times \Sigma^*)^2$ representa a transição entre duas configurações instantâneas de um AFD e é definida formalmente como: $[e, ay] \vdash [e', y]$ sempre que $\delta(e, a) = e'$. A notação $[e, w] \vdash^* [e', w']$ denotará o fecho transitivo e reflexivo de $\vdash$. ■

2.2.3 Equivalência entre AFDs

Exemplo 18. Considere o seguinte AFD que aceita a linguagem de palavras de tamanho ímpar:



Apesar de possuir mais estados, o AFD apresentado aceita a mesma linguagem do AFD apresentado no Exemplo 15. A diferença entre os dois é o número de estados que cada um possui (note que o número de transições depende apenas do número de estados, visto que o alfabeto é o mesmo e a função de transição é total). Logo, isso nos leva as seguintes perguntas: Quando dois AFDs são equivalentes entre si? Existe um AFD com o menor número de estados possível para uma certa linguagem? A segunda pergunta será o assunto da próxima aula. A primeira, é respondida pela definição seguinte. ■

Definição 18. Equivalência de AFDs Sejam M_1 e M_2 dois AFDs que reconhecem linguagens sobre um alfabeto Σ. Dizemos que M_1 é equivalente a M_2, $M_1 \equiv M_2$, se $L(M_1) = L(M_2)$. ■

2.3 Exercícios

2.3.1 Exercícios de fixação

1. Construa AFDs para as seguintes linguagens sobre $\Sigma = \{0, 1\}$
   (a) Palavras cujo tamanho é múltiplo de 4.
   (b) Palavras que terminam em 11.
   (c) Palavras que não possuem nenhuma ocorrência de 000.
   (d) Palavras que possuem um número par de 0's.
   (e) Palavras que possuem um número par de 0's e ímpar de 1's.
2. Apresente a definição formal dos AFDs apresentados nos exemplos dessas notas de aula.

2.3.2 Exercícios de tutoria

1. Construa AFDs para as linguagens a seguir.
   (a) $\{w \in \{0, 1\}^* \mid |w| \geq 2$ e o primeiro e penúltimo símbolos de w são $1\}$.
   (b) $\{w \in \{0, 1\}^* \mid$ o primeiro e último símbolo de w são diferentes$\}$.
   (c) $\{x10^n \mid n \geq 0$, $x \in \{0, 1\}^*$ e x tem um número par de 0s $\}$.
   (d) $\{01^i0 \mid i$ é ímpar$\}$.

(e) $\{w \in \{0,1\}^* \mid$ os quatro últimos símbolos de w possuem, no mínimo, dois 1s$\}$.

(f) $\{w \in \{0,1\}^* \mid$ a subpalavra 01 ocorre um número par de vezes em $w\}$.

2. Considere o seguinte problema:

   Existem duas jarras, uma com capacidade de 4l e outra de 3l, ambas sem nenhuma marcação de medida. Há uma torneira que fornece água à vontade. Como se deve agir para colocar 2 litros de àgua em uma das jarras, começando com ambas vazias?

   Modele esse problema como um AFD de maneira que uma palavra somente será aceita se essa representar uma solução.

3. Apresente a definição formal dos AFDs construídos por você nos exercícios 1-a) e 1-b) dessa seção.

2.3.3 Exercícios suplementares

1. Apresente a definição da linguagem aceita por um AFD usando a relação de transição entre configurações intantâneas.

2. Seja $M = (E, \Sigma, \delta, i, F)$ um AFD qualquer. Prove que, se $\widehat{\delta}(e, w) = e'$ então $[e, w] \vdash^* [e', \lambda]$, para todo $e, e' \in E$ e $w \in \Sigma^*$.

3

Aula 3 - Minimização de AFDs

Objetivos

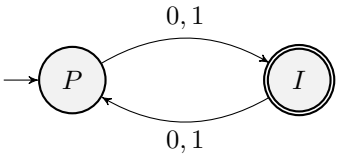
- Apresentar o conceito de equivalência entre estados de um AFD.
- Apresentar a construção do AFD mínimo equivalente.
- Demonstrar que o AFD produzido pela minimização é equivalente ao AFD original.

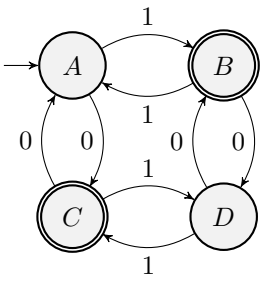
3.1 Introdução

Exemplo 19 (Dois AFDs equivalentes)**.** Considere a seguinte linguagem $L \subseteq \{0,1\}^*$:

$$L = \{w \in \{0,1\}^* \mid \exists k.|w| = 2k+1\}$$

isto é, palavras de tamanho ímpar. Abaixo, apresentamos dois AFDs que aceitam essa linguagem.





O objetivo da aula de hoje é mostrar que dados dois ou mais AFDs equivalentes é possível construir um AFD mínimo (isto é, com o menor número possível de estados) equivalente. Além disso, esse AFD mínimo é único, a menos de isomorfismo de grafos. ■

3.2 Construção do AFD mínimo

Definição 19 (Equivalência de estados)**.** Seja $M = (E, \Sigma, \delta, i, F)$ um AFD. Dizemos que dois estados $e_1, e_2 \in E$ são equivalentes, $e_1 \approx e_2$, se a seguinte fórmula é verdadeira:

$$\forall w.w \in \Sigma^* \rightarrow \widehat{\delta}(e_1, w) \in F \leftrightarrow \widehat{\delta}(e_2, w) \in F$$

A equivalência de estados é uma relação de equivalência: ela é reflexiva, transitiva e simétrica. Logo, essa introduz uma partição sobre o conjunto de estados. O AFD mínimo equivalente consiste em considerar como novo conjunto estados o conjunto de classes de equivalência induzida pela relação $\approx$. ■

Definição 20 (Autômato finito mínimo equivalente)**.** . Seja $M = (E, \Sigma, \delta, i, F)$ um AFD. O AFD mínimo equivalente a M é $M' = (E', \Sigma, \delta', i', F')$, em que:

- $E' = \{[e]_{\approx} \mid e \in E\}$, isto é, o conjunto de classes de equivalência induzidas por $\approx$ em E.
- $\delta'([e]_{\approx}, a) = [\delta(e, a)]_{\approx}$.
- $i' = [i]_{\approx}$.
- $F' = \{[e]_{\approx} \mid e \in F\}$.

Visto a definição acima, temos que o problema central em minimizar um AFD é encontrar a partição de seu conjunto de estados. ■

3.2.1 Correção da construção do AFD mínimo

Lema 1. *Se $e_1 \approx e_2$, então $\delta(e_1, a) \approx \delta(e_2, a)$.*

Demonstração. Suponha que $e_1 \approx e_2$.Suponha $a \in \Sigma$, $y \in \Sigma^*$ arbitrários. Temos que:

$$\begin{array}{lll} \widehat{\delta}(\delta(e_1, a), y) \in F & \leftrightarrow & \\ \widehat{\delta}(e_1, ay) \in F & \leftrightarrow & \{e_1 \equiv e_2\} \\ \widehat{\delta}(e_2, ay) \in F & \leftrightarrow & \\ \widehat{\delta}(\delta(e_2, a), y) \in F & & \end{array}$$

Como $y \in \Sigma^*$ é arbitrário, temos que $[\delta(e_1, a)] \approx [\delta(e_2, a)]$. □

Lema 2. *$e \in F$ se e somente se $[e]_{\approx} \in F'$.*

Demonstração.
($\rightarrow$) Imediato pela definição de F'.
($\leftarrow$) Suponha que $[e]_{\approx} \in F'$. Como $[e]_{\approx} \in F'$, pela definição de F', existe $e' \in F$ tal que $e \approx e'$. Uma vez que $e \approx e'$ e $e' \in F$, temos que $e \in F$. □

Lema 3. *Para todo $w \in \Sigma^*$, $e \in E$, temos que $\widehat{\delta'}([e], w) = [\widehat{\delta}(e, w)]$*

Demonstração. Suponha $w \in \Sigma^*$ arbitrário. Procederemos por indução sobre w.

- Caso base: Para $w = \lambda$, temos:

$$\begin{array}{ll} \widehat{\delta'}([e], w) & = \\ [e] & = \\ [\widehat{\delta}(e, \lambda)] & \end{array}$$

- Passo indutivo. Suponha que para todo $e \in E$, $\widehat{\delta'}([e], w) = [\widehat{\delta}(e, w)]$. Suponha $a \in \Sigma$ arbitrário. Temos:

$$\begin{array}{lll} \widehat{\delta'}([e], aw) & = & \{\textit{def. de } \widehat{\delta}.\} \\ \widehat{\delta'}(\delta'([e], a), w) & = & \{\textit{def. de } \delta'\} \\ [\widehat{\delta'}([\delta(e, a)], w)] & = & \{H.I.\} \\ [\widehat{\delta}(\delta(e, a), w)] & = & \{\textit{def. de } \widehat{\delta}.\} \\ [\widehat{\delta}(e, aw)] & & \end{array}$$

□

Teorema 1 (Correção da construção de AFD mínimo)**.** *Seja M um AFD qualquer e M' o seu AFD mínimo construído usando a Definição 20. Temos que $L(M') = L(M)$.*

Demonstração. Suponha $w \in \Sigma^*$ arbitrário. Temos:

$$\begin{array}{lll} w \in L(M') & \leftrightarrow & \{\textit{definição de L(M')}\} \\ \widehat{\delta}(i', w) \in F' & \leftrightarrow & \{\textit{definição de i'}\} \\ \widehat{\delta}([i], w) \in F' & \leftrightarrow & \{\textit{Lema 3}\} \\ [\widehat{\delta}(i, w)] \in F' & \leftrightarrow & \{\textit{Lema 2}\} \\ [\widehat{\delta}(i, w)] \in F & \leftrightarrow & \{\textit{definição de L(M)}\} \\ w \in L(M) & & \end{array}$$

□

3.2.2 Construção da partição de estados

Definição 21 (Estados i-equivalentes)**.** Seja $M = (E, \Sigma, \delta, i, F)$ um AFD. A relação $\approx_i$ (i-equivalência) é definida como:

- $e \approx_0 e'$ se, e somente se, $e, e' \in F \vee e, e' \in (E - F)$.
- $e \approx_{n+1} e'$ se, e somente se, $e \approx_n e'$ e, para todo $a \in \Sigma$ temos que $\delta(e, a) \approx_n \delta(e', a)$.

■

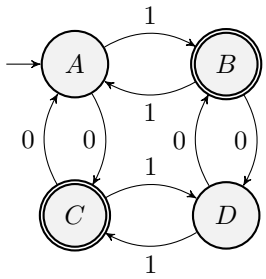
Definição 22. Partições sucessivas Seja $M = (E, \Sigma, \delta, i, F)$ um AFD. Definimos as partições sucessivas de E com respeito a $\approx_i$ como:

$$[e]_0 = \begin{cases} F & \textit{se } e \in F \\ E - F & \textit{caso contrário} \end{cases}$$

$$[e]_{n+1} = \{e' \in [e]_n \mid \forall a.a \in \Sigma \rightarrow [\delta(e', a)] = [\delta(e, a)]\}$$

A construção iterativa da partição continua até que $[e]_n = [e]_{n+1}$. ■

Exemplo 20. Para exemplificar as diversas definições apresentadas, vamos considerar o AFD apresentado no início destas notas.



Para encontrar o AFD mínimo equivalente ao acima apresentado, devemos construir a partição de seu conjunto de estados. Para isso, utilizaremos uma tabela para armazenar a configuração intermediária da partição. No primeiro momento, apresentamos a partição para $n = 0$.

n	Partição
0	$\{A, D\}\{B, C\}$

Note que na primeira iteração, dividimos o conjunto de estados em dois subconjuntos contendo os estados finais e não finais. Nessa etapa, podemos afirmar que $A \approx_0 D$ e que $B \approx_0 C$.

Para a iteração 1, devemos verificar se $A \approx_1 D$. Para isso, vamos recorrer a Definição 21, que especifica:

$$A \approx_1 D \leftrightarrow A \approx_0 D \wedge \forall a \in \Sigma.\delta(A, a) \approx_0 \delta(D, a).$$

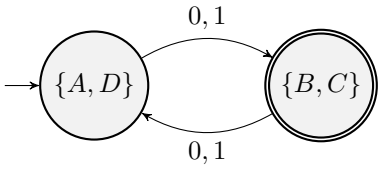
Isto é, para A e D serem 1-equivalentes eles devem: 1) ser 0-equivalentes e; 2) o resultado de toda transição saindo desses estados também deve ser 0-equivalente. Uma vez que $A \approx_0 B$, basta verificar se o resultado de cada uma das transições também é 0-equivalente.

Para $a = 0$, temos que $\delta(A, 0) = C \approx_0 B = \delta(D, 0)$. Por sua vez, para $a = 1$, temos que $\delta(A, 1) = B \approx_0 C = \delta(D, 1)$. Dessa forma, temos que $A \approx_1 D$.

Seguindo um raciocínio similar, podemos constatar que $B \approx_1 C$. Logo, chegamos no seguinte resultado.

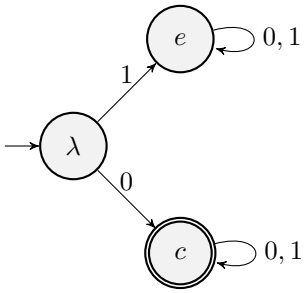
n	Partição
0	$\{A, D\}\{B, C\}$
1	$\{A, D\}\{B, C\}$

Como $[e]_n = [e]_{n+1}$, temos que a partição de estados para o AFD é $\{\{A, D\}, \{B, C\}\}$.
Finalmente, aplicando a construção da Definição 20, obtemos o seguinte AFD:



■

Exemplo 21. Nesse exemplo vamos construir o AFD mínimo equivalente ao seguinte AFD:



Construindo a partição para $n = 0$.

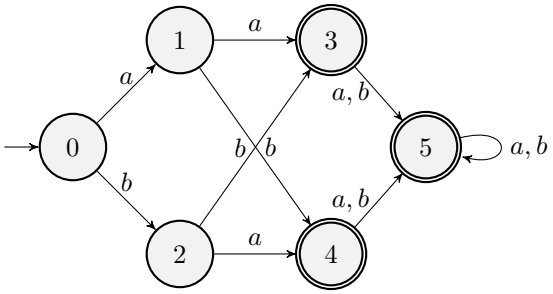
n	Partição
0	$\{c\}\{\lambda, e\}$

Note que a única possibilidade de evolução dessa partição é pela verificação de $\lambda \approx_1 e$. Porém, veja que $\delta(\lambda, 0) = C \not\approx_0 e = \delta(e, 0)$. Dessa forma, temos que $\lambda \not\approx_1 e$ e, portanto, esses estados devem estar em conjuntos diferentes na partição.

n	Partição
0	$\{c\}\{\lambda, e\}$
1	$\{c\}\{\lambda\}\{e\}$

Observe que na iteração 1 obtivemos uma partição que é formada apenas por conjuntos unitários. O que esse fato quer dizer? Observe que cada conjunto de estados de uma partição contém todos os estados equivalentes entre si. Se a partição é formada apenas por conjuntos unitários, isso evidencia que o AFD em questão já é o mínimo. ■

Exemplo 22. Construa o AFD mínimo equivalente ao AFD apresentado a seguir.



Temos a seguinte partição inicial dos estados:

n	Partição
0	$\{0,1,2\}\{3,4,5\}$

Como temos mais estados nos conjuntos, temos mais possibilidades para o mesmo conjunto. Por exemplo, temos que verificar, no primeiro conjunto, se $0 \approx_1 1$ e $1 \approx_1 2$.

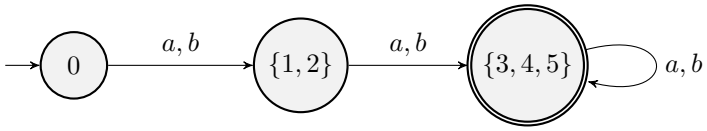
Primeiramente, para $0 \approx_1 1$ temos que $\delta(0,a) = 1 \not\approx_0 3 = \delta(1,a)$. Logo, $0 \not\approx_1 1$ e, portanto, esses dois estados não devem ficar no mesmo conjunto. Por sua vez, temos que $1 \approx_1 2$, uma vez que $\delta(1,a) = 3 \approx_0 4 = \delta(2,a)$ e $\delta(1,b) = 4 \approx_0 3 = \delta(2,b)$. Dessa forma, temos que o estado 0 deve ser removido desse conjunto. Com isso, temos a seguinte partição intermediária. Omitiremos a verificação de que os estados $3, 4$ e 5 são equivalentes, pois essa será repetida no próximo passo.

n	Partição
0	$\{0,1,2\}\{3,4,5\}$
1	$\{0\},\{1,2\},\{3,4,5\}$

Para construir a partição da iteração 2, pode-se ver que os estados 1 e 2 são equivalantes. Logo, para finalizar a partição basta verificar se os estados $3, 4$ e 5 são equivalentes. Note que os estados 3 e 4 são equivalentes pois ambos tem transições apenas para o estado 5. Logo, $3 \approx_2 4$. Da mesma forma, temos que $3 \approx_2 5$ pois o 5 tem apenas um ciclo. Logo os estados 3, 4 e 5 são equivalentes. Com isso, temos a seguinte partição final.

n	Partição
0	$\{0,1,2\}\{3,4,5\}$
1	$\{0\},\{1,2\},\{3,4,5\}$
2	$\{0\},\{1,2\},\{3,4,5\}$

A seguir, apresentamos a construção do AFD mínimo usando a partição obtida.



■

3.3 Exercícios

3.3.1 Exercícios de fixação

1. Apresente AFDs para as seguintes linguagens sobre $\Sigma = \{0,1\}$ e o respectivos AFDs mínimos.
   (a) Palavras cujo tamanho é múltiplo de 4.
   (b) Palavras que terminam em 11.
   (c) Palavras que não possuem nenhuma ocorrência de 000.
   (d) Palavras que possuem um número par de 0's.
   (e) Palavras que possuem um número par de 0's e ímpar de 1's.

3.3.2 Exercícios de tutoria

1. Apresente AFDs mínimos para as linguagens a seguir.
   (a) $\{w \in \{0,1\}^* \,|\, |w| \geq 2$ e o primeiro e penúltimo símbolos de w são $1\}$.
   (b) $\{w \in \{0,1\}^* \,|\,$ o primeiro e último símbolo de w são diferentes$\}$.
   (c) $\{x10^n \,|\, n \geq 0$, $x \in \{0,1\}^*$ e x tem um número par de 0s $\}$.

(d) $\{01^i0 \mid i \text{ é ímpar}\}$.

(e) $\{w \in \{0,1\}^* \mid$ os quatro últimos símbolos de w possuem, no mínimo, dois 1s$\}$.

(f) $\{w \in \{0,1\}^* \mid$ a subpalavra 01 ocorre um número par de vezes em $w\}$.

3.3.3 Exercícios suplementares

1. Prove que $\approx$ é uma relação de equivalência.
2. O Prof. Mipha Laram argumenta que pode-se obter AFDs mais eficientes por executar o processo de minimização de AFDs uma segunda vez. Mostre, utilizando as definições apresentadas nesta aula, que o argumento do Prof. Mipha Laram é incorreto.

4

Aula 4 - Produto, complementação de AFs e introdução aos autômatos finitos não determínisticos

Objetivos

- Apresentar as construções de produto (união e interseção) e complementação de AFDs.
- Apresentar a correção da construção de produto.
- Apresentar o conceito de autômato finito não-determínistico (AFN).

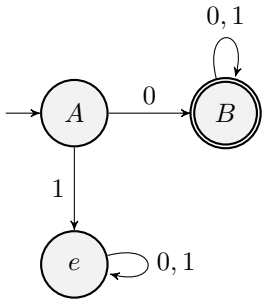
4.1 A construção do produto

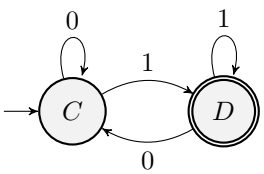
Definição 23 (Produto de AFDs)**.** Sejam $M_1 = (E_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$ dois AFDs quaisquer. O AFD $M_3 = (E_3, \Sigma, \delta_3, i_3, F_3)$ é o produto de M_1 e M_2, em que:

- $E_3 = E_1 \times E_2$
- $\delta_3((e_1, e_2), a) = (\delta_1(e_1, a), \delta_2(e_2, a))$.
- Para F_3, temos duas possibilidades:
  - Para interseção: $F_3 = F_1 \times F_2$.
  - Para união: $F_3 = (F_1 \times E_2) \cup (E_1 \times F_2)$.

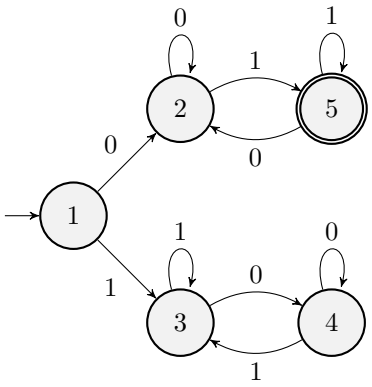
■

Exemplo 23. Considere os seguintes AFDs, um que reconhece $L_1 = \{0\}\{0,1\}^*$ e o segundo $L_2 = \{0,1\}^*\{1\}$:

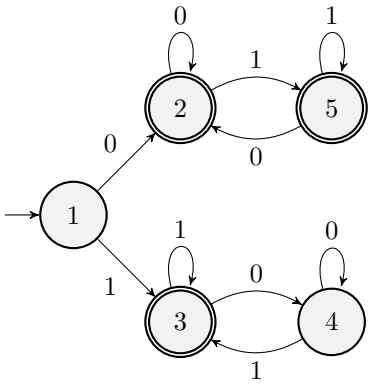




Aplicando a construção de produto, podemos criar AFDs que $L_1 \cap L_2$ e $L_1 \cup L_2$ que são apresentados a seguir. Vamos usar as seguintes abreviações para os conjuntos de estados: 1 para o conjunto $\{A, C\}$, 2 para $\{B, C\}$, 3 para $\{e, D\}$, 4 para $\{e, C\}$ e 5 para $\{B, D\}$. Primeiro, apresentamos o AFD para $L_1 \cap L_2$.



A seguir apresentamos o AFD para $L_1 \cup L_2$.



■

4.1.1 Correção da construção de produto

Apresentaremos a correção para a construção da interseção. O mesmo raciocínio aplica-se para a união.

Lema 4. *Para todo $e_1 \in E_1, e_2 \in E_2$ e $w \in \Sigma^*$, temos que $\widehat{\delta}_3((e_1, e_2), w) = (\widehat{\delta_1}(e_1, w), \widehat{\delta}_2(e_2, w))$.*

Demonstração. Indução sobre w.

- Caso base ($w = \lambda$). Temos:

$$\widehat{\delta}_3((e_1, e_2), \lambda) = (e_1, e_2) = (\widehat{\delta_1}(e_1, \lambda), \widehat{\delta}(e_2, \lambda))$$

- Passo indutivo. Suponha $a \in \Sigma$ arbitrario e que para todo e_1, e_2; $\widehat{\delta}_3((e_1, e_2), w) = (\widehat{\delta_1}(e_1, w), \widehat{\delta}_2(e_2, w))$. Temos:

$$\begin{array}{lcl}
\widehat{\delta}_3((e_1, e_2), aw) & = & \{\textit{def. de } \widehat{\delta}.\} \\
\widehat{\delta}_3(\delta_3((e_1, e_2), a), w) & = & \{\textit{def. de } \delta_3\} \\
\widehat{\delta}_3((\delta_1(e_1, a), \delta_2(e_2, a)), w) & = & \{\textit{I.H.}\} \\
(\widehat{\delta_1}(\delta_1(e_1, a), w), \widehat{\delta}_2(\delta_2(e_2, a), w)) & = & \{\textit{def. de } \widehat{\delta}\} \\
(\widehat{\delta_1}(e_1, aw), \widehat{\delta}(e_2, aw)) & &
\end{array}$$

□

Teorema 2. *Correção da construção de produto para interseção Sejam M_1 e M_2 dois AFDs quaisquer e M_3 o AFD construído para a interseção usando M_1 e M_2. Temos que $L(M_3) = L(M_1) \cap L(M_2)$.*

Demonstração. Suponha $w \in \Sigma^*$. Temos:

$$\begin{array}{lcl} w \in L(M_3) & \leftrightarrow & \{\textit{def. de } L(M_3)\} \\ \widehat{\delta}_3(i_3, w) \in F_3 & \leftrightarrow & \{\textit{def. de } i_3, F_3\} \\ \widehat{\delta}_3((i_1, i_2), w) \in F_1 \times F_2 & \leftrightarrow & \{\textit{Lema } 5\} \\ (\widehat{\delta}_1(e_1, w), \widehat{\delta}_2(e_2, w)) \in F_1 \times F_2 & \leftrightarrow & \{\textit{Prod. Cart.}\} \\ \widehat{\delta}_1(e_1, w) \in F_1 \wedge \widehat{\delta}_2(e_2, w) \in F_2 & \leftrightarrow & \{\textit{def. de } L(M)\} \\ w \in L(M_1) \wedge w \in L(M_2) & \leftrightarrow & \{\textit{def. de } \cap\} \\ w \in L(M_1) \cap L(M_2) & & \end{array}$$

Como w é arbitrário, temos que $L(M_3) = L(M_1) \cap L(M_2)$. □

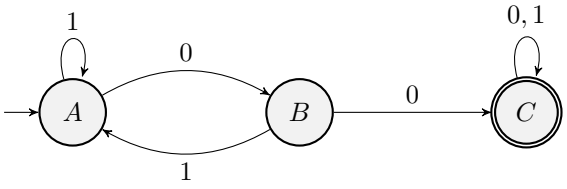
4.2 A construção da complementação

Definição 24. Seja $M = (E, \Sigma, \delta, i, F)$ um AFD que reconhece uma linguagem L. O AFD $\overline{M} = (E, \Sigma, \delta, i, E - F)$ aceita $\overline{L}$. ■

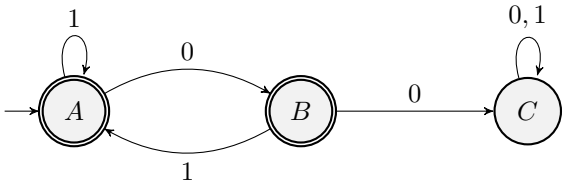
Exemplo 24. Considere a seguinte linguagem sobre $\Sigma = \{0, 1\}$:

$$L = \{0, 1\}^*\{00\}\{0, 1\}^*$$

O seguinte AFD aceita as palavras pertencentes a esta linguagem.

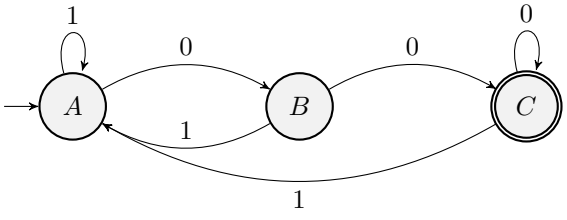


Podemos usar a construção de complementação para construir um AFD para $\overline{L}$. Para isso, basta marcar como finais todos os estados não finais e como não finais os antigos estados finais. Abaixo apresentamos o resultado para o AFD acima apresentado. ■

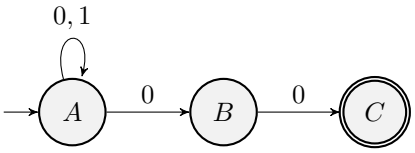


4.3 Introdução aos autômatos finitos não determínisticos

Exemplo 25 (Um exemplo informal)**.** Considere a tarefa de construir um AFD para a seguinte linguagem $\{0, 1\}^*\{00\}$.



Apesar de correto, o AFD acima é "poluído" por transições necessárias para garantir que a palavra aceita realmente termina em 00. Uma alternativa, mais compacta, e de mais fácil entendimento seria o seguinte autômato:



Após uma rápida análise, podemos perceber que o autômato anterior realmente aceita apenas palavras da linguagem $\{0,1\}^*\{00\}$. Porém, o autômato anterior não é um AFD, pois a partir do estado A processando 0 na palavra de entrada, temos transições tanto para o próprio estado A quanto para o estado B. O autômato acima apresentado é um exemplo de um autômato finito não determínistico. ■

Definição 25 (Autômato finito não determinístico - AFN)**.** Um AFN é uma quíntupla $M = (E, \Sigma, \delta, I, F)$, em que:

- E: conjunto finito de estados.
- Σ: alfabeto.
- $\delta : E \times \Sigma \to \mathcal{P}(E)$: função total de transição.
- $I \subseteq E$: conjunto de estados iniciais.
- $F \subseteq E$: conjunto de estados finais.

Observe que a função de transição produz, como resultado, um conjunto de estados e, por isso, seu contra-domínio é $\mathcal{P}(E)$, o conjunto de todos subconjuntos de estados. ■

Exemplo 26. Vamos considerar o AFN apresentado anteriormente sob um ponto de vista de sua definição matemática. Primeiramente, temos que o seu conjunto de estados e seu alfabeto são, respectivamente, $E = \{A, B, C\}$ e $\Sigma - \{0, 1\}$. O conjunto de estados iniciais é $I = \{A\}$ e o de estados finais $F = \{C\}$. Finalmente, a função de transição para esse AFN é dada pela seguinte tabela.

δ	0	1
A	$\{A, B\}$	$\{A\}$
B	$\{C\}$	$\emptyset$
C	$\emptyset$	$\emptyset$

■

4.3.1 Linguagem aceita por um AFN

Intuitivamente, dizemos que uma palavra $w \in \Sigma^*$ é aceita por um AFN M se **existe** uma computação que termina em algum estado final de M quando essa é iniciada em um estado inicial para a palavra w. O objetivo dessa seção é apresentar formalmente essa noção.

Definição 26 (Função de transição estendida)**.** Seja $M = (E, \Sigma, \delta, I, F)$ um AFN qualquer. A função de transição estendida $\widehat{\delta}^* : \mathcal{P}(E) \times \Sigma^* \to \mathcal{P}(E)$ é definida como:

- $\widehat{\delta}^*(\emptyset, w) = \emptyset$, para todo $w \in \Sigma^*$.
- $\widehat{\delta}^*(A, \lambda) = A$, para todo $A \subseteq E$.
- $\widehat{\delta}^*(A, ay) = \widehat{\delta}^*(\bigcup_{e \in A} \delta(e, a), y)$, para todo $a \in \Sigma, y \in \Sigma^*$ e $A \subseteq E$.

■

Definição 27 (Linguagem aceita por um AFN)**.** Seja $M = (E, \Sigma, \delta, I, F)$ um AFN qualquer. A linguagem aceita por M, $L(M)$, é definida como:

$$L(M) = \{w \in \Sigma^* \mid \widehat{\delta^*}(I, w) \cap F \neq \emptyset\}$$

Isto é, o conjunto de palavras w cujo processamento termina em um conjunto de estados com pelo menos um estado final. ■

4.4 Exercícios

4.4.1 Exercícios de fixação

1. Apresente AFDs para as seguintes linguagens sobre $\Sigma = \{0, 1\}$. Se possível, tente expressá-las como linguagens "menores" e use as construções apresentadas para obter o AFD final.
   (a) Palavras cujo tamanho é menor que 3.
   (b) Palavras cujo tamanho é maior que 3.
   (c) Palavras com no máximo 3 ocorrências de 1's.
   (d) Palavras que contém um ou dois 1's e cujo tamanhé múltiplo de 3.
2. Construa AFNs para as seguintes linguagens. Considere o alfabeto $\Sigma = \{a, b, c\}$.
   (a) Palavras em que o último símbolo seja igual ao primeiro.
   (b) Palavras em que o útlimo símbolo seja diferente do primeiro.
   (c) Palavras terminadas em abab.
   (d) Palavras em que o último símbolo não tenha ocorrido antes.

4.4.2 Exercícios de tutoria

1. Sejam M_1 e M_2 dois AFDs quaisquer. Mostre como construir um AFD M_3 tal que $L(M_3) = L(M_1) - L(M_2)$.
2. Sejam $L_1 = \{0\}\{0, 1\}^*$ e $L_2 = \{0, 1\}^*\{1\}$. Apresente AFDs para as seguintes linguagens usando as construções de produto e complementação:
   (a) $L_1 \cap L_2$.
   (b) $L_1 \cup L_2$.
   (c) $L_1 - L_2$.
   (d) $(L_1 - L_2) \cup (L_2 - L_1)$.

4.4.3 Exercícios suplementares

1. Mostre como a partir de um AFN qualquer é possível construir um AFN equivalente contendo apenas um estado inicial.

5

Aula 5 - Equivalência entre AFNs e AFDs e introdução aos AFNs com transições λ

Objetivos

- Apresentar a construção de subconjunto (equivalência entre AFNs e AFDs).
- Apresentar o conceito de AFN com transições λ e sua equivalência com AFNs.

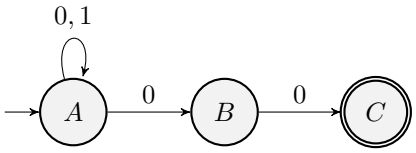
5.1 Equivalência entre AFNs e AFDs

Definição 28 (A construção de subconjunto)**.** Seja $M = (E, \Sigma, \delta, I, F)$ um AFN. O AFD $M' = (E', \Sigma, \delta', i', F')$ é equivalente a AFN, em que:

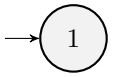
- $E' = \mathcal{P}(E)$
- $\delta'(X, a) = \bigcup_{e \in X} \delta(e, a)$, $\delta'(\emptyset, a) = \emptyset$, para $a \in \Sigma$.
- $i' = I$.
- $F' = \{X \subseteq E \mid X \cap F \neq \emptyset\}$.

■

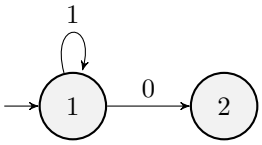
Exemplo 27. Considere o seguinte AFN para a linguagem $\{0,1\}^*\{00\}$.



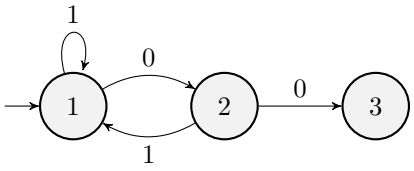
A partir da Definição 28, podemos construir o AFD equivalente a este AFN. Porém, como o conjunto de estados do AFD é $\mathcal{P}(E)$, este pode possui um número exponencial de estados. Em algumas situações, muitos destes estados são inúteis (um estado é inútil se ele não é alcançável a partir do estado inicial). Dessa forma, uma maneira de evitar tais estados inúteis é adicionar estados sob demanda, começando pelo estado que representa o conjunto de estados iniciais. Vamos adotar o rótulo 1 para denotar o conjunto $\{A\}$.



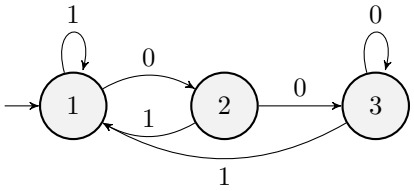
Note que no AFN, $\delta(A, 0) = \{A, B\}$ e $\delta(A, 1) = \{A\}$. Logo, devemos incluir um estado relativo ao conjunto $\{A, B\}$, que possuirá o rótulo 2.



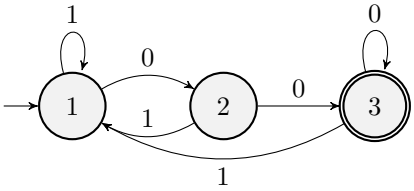
Com isso, encerramos as transições a partir do estado 1, que representa o conjunto $\{A\}$. Agora, para o estado 2, denotando $\{A, B\}$, temos que: $\delta(\{A, B\}, 0) = \{A, B, C\}$ e $\delta(\{A, B\}, 1) = \{A\}$. Logo, temos que incluir um novo estado que corresponde ao conjunto $\{A, B, C\}$. Chamaremos esse novo estado de 3.



Finalmente, obtemos o AFD equivalente ao incluir as transições para o estado 3, equivalente a $\{A, B, C\}$ que são: $\delta(\{A, B, C\}, 0) = \{A, B, C\}$ e $\delta(\{A, B, C\}, 1) = \{A\}$.



Como não há estados sem transições a serem incluídas, terminamos a construção. Agora, falta apenas indicar os estados finais, conforme abaixo.



■

5.1.1 Correção da construção de subconjunto

Lema 5. *Seja M um AFN e M' o seu AFD obtido pela construção de subconjunto. Então, para todo $X \subseteq E$ e $w \in \Sigma^*$, $\widehat{\delta}(X, w) = \widehat{\delta}'(X, w)$.*

Demonstração. Por indução sobre w.

- Caso base ($w = \lambda$). Temos:

$$\widehat{\delta}(X, \lambda) = X = \widehat{\delta}'(X, w)$$

- Passo indutivo ($w = ay$). Suponha que $\widehat{\delta}(X, y) = \widehat{\delta}'(X, y)$. Considere os seguintes casos:

– $X = \emptyset$. Temos:

$$\begin{array}{lcl}
\widehat{\delta'}(\emptyset, ay) & = & \{def.\ de\ \widehat{\delta}\} \\
\widehat{\delta'}(\delta'(\emptyset, a), y) & = & \{def.\ de\ \delta'\} \\
\widehat{\delta'}(\emptyset, y) & = & \{H.I.\} \\
\widehat{\delta}(\emptyset, y) & = & \{def.\ de\ \widehat{\delta}\} \\
\emptyset & = & \\
\widehat{\delta}(\emptyset, ay) & &
\end{array}$$

– $X = ay$. Temos:

$$\begin{array}{lcl}
\widehat{\delta'}(X, ay) & = & \{def.\ de\ \widehat{\delta}\} \\
\widehat{\delta'}(\delta'(X, a), y) & = & \{def.\ de\ \delta'\} \\
\widehat{\delta'}(\bigcup_{e \in X} \delta(e, a), y) & = & \{H.I.\} \\
\widehat{\delta}(\bigcup_{e \in X} \delta(e, a), y) & = & \{def.\ de\ \widehat{\delta}\} \\
\widehat{\delta}(X, ay) & &
\end{array}$$

□

Teorema 3 (Correção da construção de subconjunto.)**.** *Suponha $M = (E, \Sigma, \delta, I, F)$ um AFN arbitrário. Existe um AFD M' tal que $L(M) = L(M')$.*

Demonstração. Seja M' o AFD obtido a partir de M usando a Definição 28. Suponha $w \in \Sigma^*$ arbitrário. Temos:

$$\begin{array}{lcl}
w \in L(M') & \leftrightarrow & \{def.\ de\ L(M)\} \\
\delta'(I, w) \in F' & \leftrightarrow & \{def.\ de\ F'\} \\
\delta'(I, w) \cap F \neq \emptyset & \leftrightarrow & \{Lema\ 5\} \\
\delta(I, w) \cap F \neq \emptyset & \leftrightarrow & \{def.\ de\ L(M)\} \\
w \in L(M) & &
\end{array}$$

Como $w \in \Sigma^*$ é arbitrário, temos que $L(M') = L(M)$. □

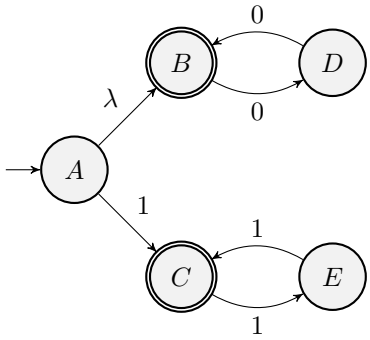
5.2 AFNs com transições λ

Definição 29 (AFN com transições λ (AFNλ))**.** Um AFNλ $M = (E, \Sigma, \delta, I, F)$ é uma quíntupla, em que:

- E, Σ, I e F são como em AFNs.
- $\delta : E \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(E)$, uma função total.

■

Exemplo 28. Considere a tarefa de construir um AFN para a linguagem: $\{00\}^* \cup \{1\}\{11\}^*$. Um AFN$\lambda$ pode representar essa linguagem de maneira simples, como abaixo:



■

Definição 30 (Função fecho de λ ($f\lambda$))**.** Seja $M = (E, \Sigma, \delta, I, F)$ um AFNλ. A função fecho de λ, $f\lambda : \mathcal{P}(E) \to \mathcal{P}(E)$, é definida recursivamente como:

- $X \subseteq f\lambda(X)$, para todo $X \subseteq E$.
- Se $e \in f\lambda(X)$ então $\delta(e, \lambda) \subseteq f\lambda(X)$.

■

Exemplo 29. Considere o AFNλ do Exemplo 28. Abaixo apresentamos o resultado de $f\lambda$ para cada um dos estados deste AFN.

$$f\lambda(\{A\}) = \{A, B\} \quad f\lambda(\{B\}) = \{B\}$$
$$f\lambda(\{C\}) = \{C\} \quad f\lambda(\{D\}) = \{D\}$$
$$f\lambda(\{E\}) = \{E\}$$

■

Definição 31 (Função de transição estendida para AFNλ)**.** Seja $M = (E, \Sigma, \delta, I, F)$ um AFNλ. A função de transição estendida para M, $\widehat{\delta} : \mathcal{P}(E) \times \Sigma^* \to \mathcal{P}(E)$, é definida recursivamente como:

- $\widehat{\delta}(\emptyset, a) = \emptyset$
- $\widehat{\delta}(A, \lambda) = f\lambda(A)$
- $\widehat{\delta}(A, ay) = \widehat{\delta}(\bigcup_{e \in f\lambda(A)} \delta(e, a), y)$, para $a \in \Sigma$ e $y \in \Sigma^*$.

A linguagem aceita por um AFNλ é definida como

$$L(M) = \{w \in \Sigma^* \mid \widehat{\delta}(I, w) \cap F \neq \emptyset\}$$

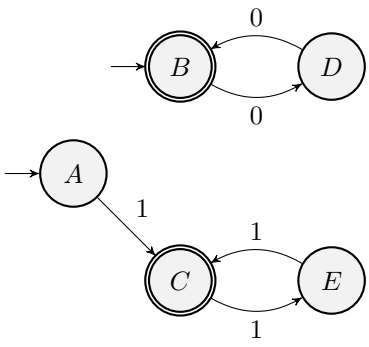
■

Definição 32 (Construção de um AFN a partir de um AFNλ)**.** Seja $M = (E, \Sigma, \delta, I, F)$ um AFNλ. O AFN equivalente a M é $M' = (E, \Sigma, \delta', I', F)$ em que:

- $I' = f\lambda(I)$
- $\delta'(e, a) = f\lambda(\delta(e, a))$, para $e \in E$ e $a \in \Sigma$.

■

Exemplo 30. Considere o Exemplo 28. A seguir apresentamos o AFN equivalente produzido pela construção da Definição 32.



■

5.3 Exercícios

5.3.1 Exercícios de fixação

1. Construa AFNs para as seguintes linguagens. Em seguida, aplique a construção de conjunto para obter o AFD equivalente. Considere o alfabeto $\Sigma = \{a, b, c\}$.

   (a) Palavras em que o último símbolo seja igual ao primeiro.

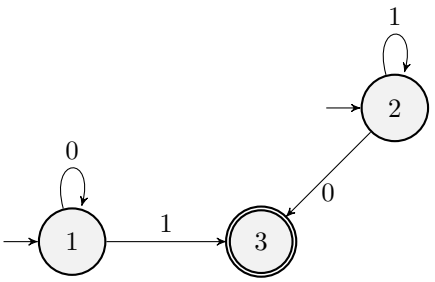
   (b) Palavras em que o útlimo símbolo seja diferente do primeiro.

   (c) Palavras terminadas em abab.

   (d) Palavras em que o último símbolo não tenha ocorrido antes.

5.3.2 Exercícios de tutoria

1. Seja o AFN abaixo.



   (a) Construa um AFD equivalente a este AFN usando o algoritmo descrito em sala.

2. Considere o AFN $M = (\{1, 2, 3, 4\}, \{0, 1\}, \delta, \{1, 3\}, \{1, 4\})$, em que:

δ	0	1
1	$\{2\}$	$\{1\}$
2	$\{2, 3\}$	$\emptyset$
3	$\{3\}$	$\{4\}$
4	$\{4\}$	$\{3\}$

   Faça o que se pede.

   (a) Apresente o diagrama de estados correspondente ao AFN descrito formalmente.

   (b) Apresente o AFD equivalente ao AFN acima apresentado.

3. Considere o AFNλ $M = (\{A, B, C, D\}, \{a, b\}, \delta, \{A\}, \{C, D\})$, em que:

δ	a	b	λ
A	$\{A\}$	$\emptyset$	$\{B, C\}$
B	$\emptyset$	$\{B\}$	$\{D\}$
C	$\{C\}$	$\emptyset$	$\{D\}$
D	$\{D\}$	$\{C\}$	$\{B\}$

   Faça o que se pede.

   (a) Apresente o diagrama de estados correspondente ao AFNλ descrito formalmente.

   (b) Apresente o AFN equivalente ao AFNλ acima apresentado.

   (c) Converta o AFN obtido por você no passo anterior em um AFD.

6

Aula 6 - Linguagens regulares e o lema do bombeamento

Objetivos

- Apresentar o conceito de linguagem regular.
- Apresentar o lema do bombeamento e como esse pode ser utilizado para provar que uma linguagem não é regular.

6.1 Linguagens Regulares

Definição 33 (Linguagem regular (LR))**.** Dizemos que uma linguagem $L \subseteq \Sigma^*$ é regular se existe um autômato finito determinístico que a reconhece. ■

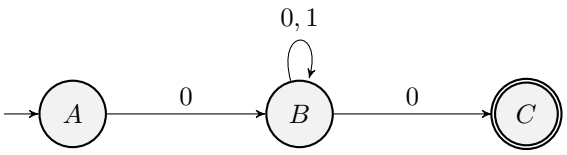
Note que como um AFNλ é equivalente a um AFN e estes são equivalentes a AFDs, é suficiente existir um autômato finito determinístico ou não.

6.2 O lema do bombeamento

O lema do bombeamento é uma importante caracterísitca possuída por toda linguagem regular (e algumas não regulares). Antes de apresentar o enunciado, demonstração e exemplos de uso do lema, é conveniente apresentar uma explicação informal.

6.2.1 Uma introdução informal

Para nossa introdução informal, vamos considerar um AFD para uma linguagem simples, mas que permitirá analisarmos o conteúdo do lema do bombeamento em um contexto concreto. A linguagem que consideraremos é $\{0\}\{0,1\}^*\{0\}$. O AFN para essa linguagem é apresentado a seguir.



Pelo diagrama acima, é fácil ver que as seguintes palavras pertencem a essa linguagem: $00, 000, 010, 0000, 0010, 0100, ...$ Como esse AF possui 3 estados, qualquer palavra w tal que $|w| \geq 3$ deve passar pelo ciclo no estado B pelo menos uma vez. Dessa forma, podemos "aumentar" o tamanho de palavras aceitas por esse AF simplesmente percorrendo o ciclo. Dessa forma, qualquer z, $|z| \geq 3$, deve ser da forma $0\{0,1\}^k0$, $k \geq 1$, em que o primeiro 0 é processado pela transição de A para B, a subpalavra $\{0,1\}^k$ é processada por k

iterações do ciclo sobre o estado B e, finalmente, o último 0 é processado pela transição de B para C. Note que $|0\{0,1\}^k0| = k+2$. Podemos reconhecer palavras de tamanho maior (por exemplo, $k+10$) simplesmente percorrendo mais vezes o ciclo sobre o estado B. Ou seja, a existência de um ciclo em um AF é uma condição para que a linguagem aceita por ele seja infinita.

A estrutura apresentada para a linguagem acima é, na verdade, compartilhada por todas as linguagens regulares. Na próxima subseção será apresentado e demonstrado o lema do bombeamento que consiste de uma versão "geral" do raciocínio acima descrito.

6.2.2 Enunciando o lema

Lema 6 (Lema do bombeamento para linguagens regulares)**.** *Seja L uma linguagem regular. Então, existe uma constrante $k > 0$, dependente de L, tal que para qualquer palavra $z \in L$, $|z| \geq k$, existem u, v e $w \in \Sigma^*$ que satisfazem as seguintes condições:*

- $z = uvw$
- $|uv| \leq k$
- $v \neq \lambda$
- $\forall i.i \in \mathbb{N} \rightarrow uv^iw \in L$

Demonstração. Suponha L uma linguagem regular. Logo, existe um AFD M tal que $L(M) = L$. Seja $k = |E|$. Suponha $z \in L$ arbitrário e que $|z| \geq k$. Seja $c(z)$ o caminho percorrido em M para aceitar z. Evidentemente, $|c(z)| = |z| + 1$ e, uma vez que $|z| \geq k$, temos, pelo princípio da casa dos pombos que existe pelo menos um estado $e \in E$ que repete em $c(z)$. Seja u a palavra formada pelas transições antes da primeira ocorrência de e em $c(z)$, v a palavra formada pelas transições entre duas ocorrências de $c(z)$ e w a palavra formada pelas transições a partir da segunda ocorrência de e. Pelo argumento apresentado, temos que $z = uvw$. Como o subcaminho que processa v possui duas ocorrências de um estado e, temos que $|uv| \leq k$. Finalmente, como v é processado por um ciclo, podemos percorrê-lo um número arbitrário e ainda produzir palavras pertencentes a $L(M)$. □

A principal aplicação do lema do bombeamento é demonstrar que uma certa linguagem não é regular. Essa demonstração é feita por contradição. Supomos que a linguagem em questão é regular, supomos a existência da constante do lema, k, e escolhemos uma palavra $z \in L$ tal que $|z| > k$. Em seguida, supomos que $z = uvw$, $|uv| \leq k$ e $v \neq \lambda$. A contradição do lema é feita encontrando um valor $i \geq 0$ tal que $uv^iw \notin L$, o que invalida a última fórmula do lema.

6.2.3 Exemplos

Exemplo 31. Suponha que $L = \{0^n1^n \mid n \geq 0\}$ seja uma LR. Seja k a constante referida no LB e $z = 0^k1^k \in L$. Como $|z| > k$, pelo LB as seguintes condições se verificam: $z = uvw$, $|uv| \leq k$, $v \neq \lambda$ e $uv^iw \in L$, para todo $i \geq 0$. Como $z = uvw = 0^k1^k$ e $|uv| \leq k$, temos que uv é formado apenas por 0's. Além disso, como $v \neq \lambda$, v deve possuir pelo menos um 0. Seja $i = 0$. Temos que $uv^0w = 0^{k-|v|}1^k$. Porém, como $v \neq \lambda$, $|v| > 0$ e, portanto, $0^{k-|v|}1^k$ possuirá mais 1's do que 0's. Portanto, $uv^0w \notin L$, o que contraria o LB. Portanto, $L = \{0^n1^n \mid n \geq 0\}$ não é uma LR. ■

Exemplo 32. Suponha que $L = \{0^n1^m \mid n > m\}$ seja uma LR. Seja k a constante referida no LB e $z = 0^{k+1}1^k \in L$. Como $|z| > k$, pelo LB as seguintes condições se verificam: $z = uvw$, $|uv| \leq k$, $v \neq \lambda$ e $uv^iw \in L$, para todo $i \geq 0$. Como $z = uvw = 0^{k+1}1^k$ e $|uv| \leq k$, temos que uv é formado apenas por 0's. Além disso, como $v \neq \lambda$, v deve possuir pelo menos um 0. Seja $i = 0$. Temos que $uv^0w = 0^{k-|v|}1^k$. Porém, como $v \neq \lambda$, $|v| > 0$ e, portanto, $0^{k-|v|}1^k$ possuirá uma quantidade de 1's maior ou igual a de 0's. Portanto, $uv^0w \notin L$, o que contraria o LB. Portanto, $L = \{0^n1^m \mid n > m\}$ não é uma LR. ■

6.3 Exercícios

6.3.1 Exercícios de fixação

1. Prove que cada uma das linguagens a seguir não é regular usando o LB.

(a) $\{0^m 1^n \mid m < n\}$

(b) $\{0^n 1^{2n} \mid n \geq 0\}$

(c) $\{ww \mid w \in \{0,1\}^*\}$

(d) $\{0^{n^2} \mid n \geq 0\}$

6.3.2 Exercícios de tutoria

1. Prove que as seguintes linguagens não são regulares usando o lema do bombeamento.

   (a) $\{0^n y \mid y \in \{0,1\}^* \wedge |y| < n\}$.

   (b) $\{0^m 1^n \mid n < m < 2n\}$.

   (c) $\{w\overline{w} \mid w \in \{0,1\}^*\}$. A notação $\overline{w}$ representa a negação bit-a-bit de w. Exemplo: se $w = 001$ então $\overline{w} = 110$.

6.3.3 Exercícios suplementares

1. Seja $L = \{ab^n c^n \mid n \geq 0\} \cup \{a^k w \mid k \geq 2 \wedge w \in \{a,b,c\}^*\}$ uma linguagem não regular. Mostre que L satisfaz o lema do bombeamento.

2. Diversas linguagens de cunho prático não são descritas como linguagens regulares. Um exemplo de tal linguagem é a descrição de pacotes de rede IPV6. Explique, usando o lema do bombeamento, porque a linguagem de tais pacotes não é regular.

7

Aula 7 - Propriedades de fechamento para linguagens regulares

Objetivos

- Apresentar o conceito matemático de operação fechada.
- Demonstrar que a classe das linguagens regulares é fechada sobre união, interseção, complementação, concatenação e fecho de Kleene.
- Apresentar aplicações de propriedades de fechamento.

7.1 Propriedades de fechamento de LRs

Definição 34 (Operação fechada)**.** Dizemos que uma função $f : A \times A \times ... \times A \to A$ é fechada sobre um conjunto A se $f(a_1, a_2, ..., a_n) \in A$, em que $a_i \in A$. ■

Exemplo 33. Alguns exemplos de operações fechadas: A adição e multiplicação são fechadas sobre o conjunto dos números naturais. Pois, para qualquer $n, m \in \mathbb{N}$ temos que $n + m \in \mathbb{N}$ e $nm \in \mathbb{N}$. Porém, a operação de subtração não é fechada sobre $\mathbb{N}$. Pois, sempre que $m > n$ temos que $n - m < 0$ e, portanto, não pertence a $\mathbb{N}$. ■

Teorema 4. *A classe das linguagens regulares é fechada sobre união, interseção e complementação.*

Demonstração. Consequência das construções de produto e complementação apresentadas na Aula 4. □

Teorema 5. *A classe das linguagens regulares é fechada sobre a concatenação.*

Demonstração. Sejam $M_1 = (E_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$ dois AFDs que aceitam $L(M_1)$ e $L(M_2)$. O AFNλ $M_3 = (E_1 \cup E_2, \Sigma, \delta_3, \{i_1\}, F_2)$ aceita $L(M_1)L(M_2)$, em que δ_3 é definida como:

$$\delta_3 = \delta_1 \cup \delta_2 \cup \{(e, \lambda, i_2) \,|\, e \in F_1\}$$

□

Teorema 6. *A classe das linguagens regulares é fechada sobre o fecho de Kleene.*

Demonstração. Seja $M = (E, \Sigma, \delta, i, F)$ um AFD que aceita $L(M)$. O AFNλ $M' = (E', \Sigma, \delta', I', F')$ aceita $L(M)^*$, em que:

- $E' = E \cup \{i'\}$, $i' \notin E$.
- $I' = \{i'\}$
- $F' = F \cup \{i'\}$

- δ' é definida como:
  - $\delta'(i', \lambda) = \{i\}$.
  - $\delta'(e, a) = \{\delta(e, a)\}$, para todo $e \in E$, $a \in \Sigma$.
  - $\delta'(e, \lambda) = \{i'\}$, para todo $e \in F$.

□

7.1.1 Aplicando as propriedades de fechamento

Propriedades de fechamento podem ser utilizadas para: 1) provar que uma linguagem é regular; 2) provar que uma linguagem não é regular e 3) permitir a construção incremental de AF para linguagens. Apresentaremos exemplos de cada uma dessas aplicações.

Exemplo 34 (Aplicação 1)**.** Considere a seguinte linguagem

$$L = \{w \in \{0,1\}^* \mid w \textit{ possui um } n^o \textit{ par de 0's e um único 1}\}$$

Podemos mostrar que essa linguagem é regular usando propriedades de fechamento. Para isso, note que $L = L_1 \cap L_2$, em que:

$$L_1 = \{w \in \{0,1\}^* \mid w \textit{ possui um } n^o \textit{ par de 0's}\}$$
$$L_2 = \{0\}^*\{1\}\{0\}^*$$

Para mostrar que tanto L_1 quanto L_2 são regulares, basta construir um AF que as aceite. ■

Exemplo 35 (Aplicação 2)**.** Seja $L = \{0^n 1^m 2^p \mid n = m + p\}$. Provaremos que L não é regular usando propriedades de fechamento. A idéia para mostrar que L não é regular é aplicarmos propriedades de fechamento de forma a obter uma linguagem que já sabemos a priori não ser regular. Ora, se L fosse regular, a aplicação de propriedades de fechamento deveria produzir somente linguagens regulares. Com isso, obtemos a contradição necessária. Segue a demonstração abaixo.

Suponha que L seja uma linguagem regular. Sabe-se que $\{0\}^*\{1\}^*$ é uma linguagem regular. Sendo assim, temos que $L \cap \{0\}^*\{1\}^*$ também deve ser uma linguagem regular, já que as LRs são fechadas sobre a interseção. Porém, $L \cap \{0\}^*\{1\}^* = \{0^n 1^n \mid n \geq 0\}$ que não é uma LR. Logo, L não pode ser uma linguagem regular. ■

7.2 Minimização de AFDs pelo algoritmo de Brzozowski

7.2.1 Preliminares

Antes de apresentar o algoritmo é necessário algumas definições.

Teorema 7 (Fechamento de reverso)**.** *A classe das linguagens regulares é fechada sobre a operação de reverso.*

Demonstração. Seja L uma linguagem regular qualquer e $M = (E, \Sigma, \delta, i, F)$ um AFD tal que $L(M) = L$. Podemos construir um AFN M^R que aceita L^R, da seguinte forma: $M^R = (E, \Sigma, \delta', F, \{i\})$, em que:

$$\delta'(e', a) = \{e \mid \delta(e, a) = e'\}$$

□

Ou seja, para construir um AF para o inverso de uma linguagem, invertemos o sentido de todas as arestas do AFD, tornando-o um AFN que possui como estado final o estado inicial do AFD original e conjunto de estados iniciais o conjunto de estados finais do AFD de origem.

Exemplo 36. Considere o seguinte AFD.

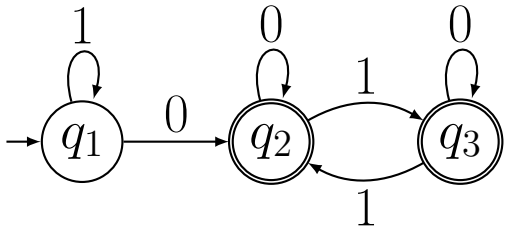


Figura 7.1: AFD de exemplo.

Vamos aplicar a construção de reverso ao AFD anterior. Primeiro, vamos inverter as arestas de sua função de transição.

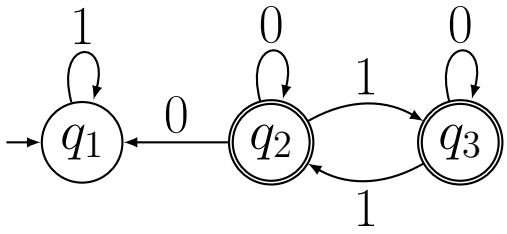


Figura 7.2: AFN resultante após a inversão de arestas

Agora para finalizar, basta tornar o estado inicial um estado final e os finais novos estados iniciais.

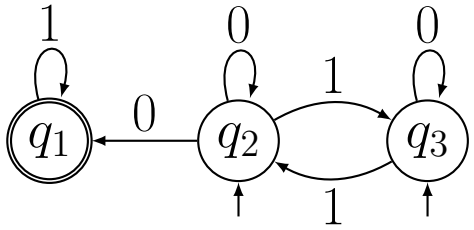


Figura 7.3: AFN para o reverso.

■

A seguir, usando a construção de reverso, podemos definir o algoritmo para minimização de AFDs proposto por Brzozowski.

7.2.2 Algoritmo de Brzozowski

O algoritmo de minimização de AFDs proposto por Brzozowski pode ser definido pelo seguinte pseudo-código Haskell:

```
minimize = determ . rev . determ . rev
```

Em que, `determ` é uma função para converter AFNs em AFDs e `rev` a construção do reverso para linguagens regulares. Ou seja, o algoritmo de Brzozowski consiste em calcular o reverso do AFD, produzindo um AFN que será transformado em AFD pela função `determ`. Repetindo o processo de reverso e conversão em AFD obtemos o mínimo para a linguagem em questão. Antes de apresentar um argumento formal do porquê esse algoritmo funciona, vamos apresentar um exemplo.

Exemplo 37. Vamos continuar o algoritmo de minimização a partir do exemplo da construção de reverso apresentada anteriormente. O resultado do reverso será repetido por conveniência do leitor.

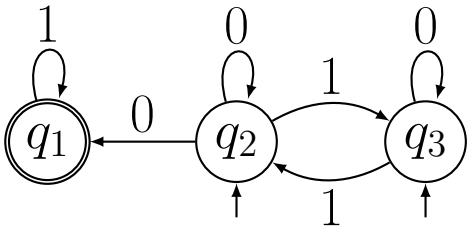


Figura 7.4: AFN para o reverso.

O próximo passo do algoritmo de minimização consiste em converter o AFN anterior em um AFD.

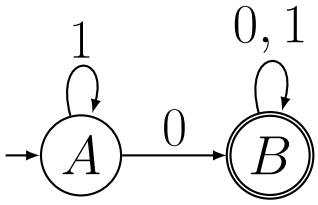


Figura 7.5: AFD resultante.

em que o estado A representa o conjunto de estados $\{q_2, q_3\}$ e B o conjunto $\{q_1, q_2, q_3\}$.

O próximo passo do algoritmo consiste em aplicar a construção de reverso sobre o AFD anterior. Para esse esse próximo passo, os conjuntos A e B são exatamente como descrito no parágrafo anterior.

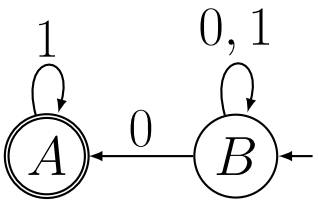


Figura 7.6: Resultado da construção de reverso

Concluímos o algoritmo por executar a conversão em AFD mais uma vez, em que $C = \{B\}$ e $D = \{A, B\}$.

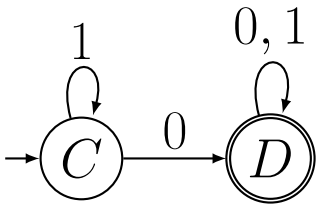


Figura 7.7: Autômato mínimo resultante.

■

7.2.3 Correção do algoritmo de Brzozowski

A demonstração de correção do algoritmo de minimização proposto por Brzozowski é uma consequência direta do seguinte resultado.

Lema 7. *Seja $M = (E, \Sigma, \delta, i, F)$ um AFD para L sem estados inalcançáveis a partir de i. Então,* `determ(rev(`M`))` *é o AFD mínimo para L^R.*

Demonstração. Seja $M^R = (E, \Sigma, \delta', F, i)$ o resultado de $\texttt{rev}(M)$ e que o resultado de $\texttt{determ}(M^R)$ seja $N = (E_N, \Sigma, \delta_N, i_N, F_N)$. Para mostrarmos que N é mínimo, vamos provar que este autômato não possui estados equivalentes. Para isso, suponha $A, B \in E_N$ e que A é equivalente a B[1]. Suponha $e \in A$. Como todo estado de M é alcançável a partir de i, temos que existe w tal que $\widehat{\delta}(i, w) = e$. Logo, $i \in \widehat{\delta'}(e, w^R)$ e portanto, $\widehat{\delta}_N(A, w^R) \in F_N$. Como A é equivalente a B, temos que $\widehat{\delta}_N(B, w^R) \in F_N$. Logo, existe $e' \in B$ tal que $i \in \widehat{\delta'}(e', w^R)$ em M^R. Portanto, $e' = \widehat{\delta}(i, w)$ em M. Porém, como M é um AFD, $\widehat{\delta}(i, w) = e$ e $\widehat{\delta}(i, w) = e'$ temos que $e = e'$. Como $e \in A$ e $e' \in B$ são arbitrários, temos que $A = B$. Dessa forma, podemos concluir que $\texttt{determ(rev}(M)\texttt{)}$ é o AFD mínimo para L^R. □

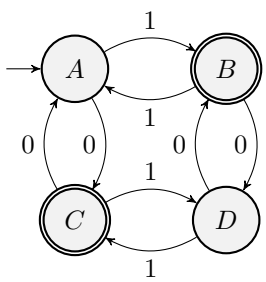
Teorema 8 (Correção da minimização de Brzozowski)**.** *O algoritmo de Brzozowski produz o AFD mínimo a partir de um AFD qualquer.*

Demonstração. Seja $M = (E, \Sigma, \delta, i, F)$ um AFD qualquer e $M' = \texttt{determ(rev}(M)\texttt{)}$. Pelo Lema 1, temos que M' é o AFD mínimo para $L(M)^R$. Seja $M_1 = \texttt{determ(rev}(M')\texttt{)}$. Novamente, temos que M_1 é o AFD mínimo para $L(M')^R$. Porém, $L(M') = L(M)^R$ e portanto, temos que $L(M')^R = (L(M)^R)^R = L(M)$. Logo, M_1 é o AFD mínimo para $L(M)$. □

7.3 Exercícios

7.3.1 Exercícios de fixação

1. Considerando as linguagens L, L_1 e L_2 apresentadas no Exemplo 34. Faça o que se pede.
   (a) Apresente AFDs para L_1 e L_2.
   (b) Usando as propriedades de fechamento, construa um AF para L.
2. Sejam L_1 e L_2 duas linguagens. Mostre que sim ou que não:
   (a) se L_1 e L_2 não são regulares, $L_1 \cup L_2$ não é regular.
   (b) se L_1 e L_2 não são regulares, $L_1 \cap L_2$ não é regular.
   (c) se L_1 não é regular, $\overline{L_1}$ não é regular.
   (d) se L_1 é regular e L_2 não é regular, $L_1 \cup L_2$ não é regular.
   (e) se L_1 é regular, L_2 não é regular e $L_1 \cap L_2$ é regular, $L_1 \cup L_2$ não é regular.
   (f) se L_1 é regular, L_2 não é regular e $L_1 \cap L_2$ não é regular, $L_1 \cup L_2$ não é regular.
3. Prove que os seguintes conjuntos não são regulares usando propriedades de fechamento.
   (a) $\{0,1\}^* - \{0^n 1^n \mid n \geq 0\}$.
   (b) $\{0^n 1^m \mid n < m\} \cup \{0^n 1^m \mid m < n\}$
4. Utilize o algoritmo de Brzozowski para minimizar o seguinte AFD.



[1] Lembre-se que A e B correspondem a conjuntos de estados em M^R

7.3.2 Exercícios de tutoria

1. Seja L uma linguagem regular sobre $\Sigma = \{a, b, c\}$. Mostre, usando propriedades de fechamento, que cada uma das linguagens seguintes é regular.

   (a) $\{w \in L \mid w$ contém pelo menos um $a\}$.

   (b) $\{w \mid w \in L \vee w$ contém pelo menos um $a\}$.

   (c) $\{w \notin L \mid w$ não contém as$\}$.

2. Seja L uma linguagem não regular e F uma linguagem finita sobre Σ. Mostre que $L \cup F$ e $L - F$ não são linguagens regulares.

3. Classifique cada uma das alternativas a seguir como sendo verdadeira ou falsa. No caso de ser verdadeira, justique esse fato. No caso de ser falsa, apresente um contra-exemplo.

   (a) Se $L_1 L_2$ é uma linguagem regular, então L_1 é regular.

   (b) Se $L_1 \cup L_2$ é uma linguagem regular, então L_1 é regular.

7.3.3 Exercícios suplementares

1. Mostre que as linguagens regulares são fechadas sobre a operação de reverso.

2. Seja L uma linguagem regular sobre um alfabeto Σ. Mostre que a seguinte linguagem também é regular, para $a \in \Sigma$:

$$\delta(L, a) = \{y \mid ay \in L \wedge y \in \Sigma^*\}$$

8

Aula 8 - Expressões regulares

Objetivos

- Apresentar expressões regulares, sua sintaxe e semântica.
- Apresentar a equivalência entre expressões regulares e autômatos finitos.
- Apresentar derivadas de expressões regulares e seu uso para obtenção de AFDs e casamento de padrão.

8.1 Expressões regulares

Definição 35 (Sintaxe e semântica de expressões regulares)**.** O conjunto de expressões regulares (ERs) sobre um alfabeto Σ é definido recursivamente como:

- $\emptyset$ denota a linguagem regular $\emptyset$.
- λ denota a linguagem regular $\{\lambda\}$.
- a, $a \in \Sigma$, denota a linguagem regular $\{a\}$.
- $e_1 + e_2$ denota a linguagem regular $L(e_1) \cup L(e_2)$.
- $e_1\, e_2$ denota a linguagem regular $L(e_1)\, L(e_2)$.
- e^* denota a linguaegm regular $L(e)^*$.

■

Exemplo 38. Abaixo apresentamos alguns exemplos de expressões regulares e os respectivos conjunto que elas denotam.

- $\emptyset$ denota a linguagem $\emptyset$.
- 01 denota a linguagem $\{0\}\{1\} = \{01\}$.
- $(0 + 1)1$ denota a linguagem $(\{0\} \cup \{1\})\{1\} = \{0, 1\}\{1\} = \{01, 11\}$.
- 0^* denota a linguagem $\{0\}^*$.

■

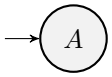
8.2 Equivalência de ERs e AFs

8.2.1 Construindo um AF a partir de uma ER

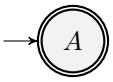
Teorema 9. *Toda expressão regular denota uma linguagem regular.*

Demonstração. Suponha e uma expressão regular arbitrária. Para mostrar que e denota uma linguagem regular, devemos mostrar como construir, a partir de e um AF. Para isso, vamos proceder por indução sobre a estrutura de e.

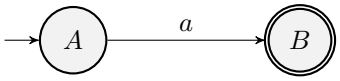
- Caso $e = \emptyset$. O AF correspondente a $e = \emptyset$ é:



- Caso $e = \lambda$. O AF correspondente a $e = \lambda$ é:



- Caso $e = a$, $a \in \Sigma$. O AF correspondente é:



- Caso $e = e_1 + e_2$. Pela hipótese de indução, temos que M_1 é o AF correspondente a e_1 e de maneira similar M_2 é o AF para e_2. O AF para e é obtido pela construção de produto para união.

- Caso $e = e_1\, e_2$. Pela hipótese de indução, temos que M_1 é o AF correspondente a e_1 e de maneira similar M_2 é o AF para e_2. O AF para e é obtido pela propriedade de fechamento para concatenação.

- Caso $e = e_1^*$. Pela hipótese de indução, temos que M_1 é o AF correspondente a e_1. O AF para e é obtido pela propriedade de fechamento para o fecho de Kleene.

□

8.2.2 Construindo uma ER a partir de um AF

Antes de se apresentar a construção de uma ER a partir de um AF, faz-se necessária a seguinte definição.

Definição 36. Um diagrama ER sobre Σ é um diagrama de estados em que as arestas são rotuladas com ER ao invés de símbolos de Σ. ■

Definição 37 (Eliminando estados de um diagrama ER)**.** Pode-se eliminar estados de diagramas ER preservando a linguagem aceita por esses usando os seguintes esquemas.

Figura 8.1: Esquema de eliminação de estados.

Outro esquema de eliminação de estados é apreseentado a seguir.

Figura 8.2: Esquema de eliminação de estados.

■

Definição 38 (Diagramas ER básicos)**.** Dizemos que um diagrama ER é básico se este é formado apenas por um estado inicial e final. A partir de um diagrama ER básico é imediato obter a ER que o denota. Abaixo apresentamos os dois possíveis casos.

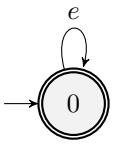


Figura 8.3: Diagrama ER para e^*.

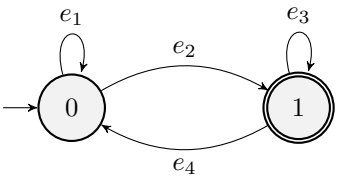


Figura 8.4: Diagrama ER para $e_1^*e_2(e_3 + e_4e_1^*e_2)^*$.

■

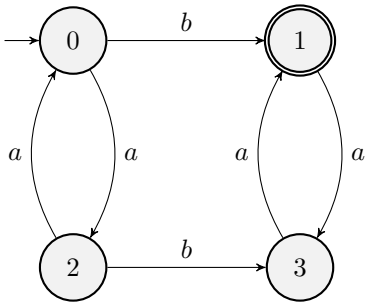
Teorema 10 (Construção de ER a partir de AFs)**.** *Toda linguagem regular é denotada por uma expressão regular.*

Demonstração. Suponha $M = (E, \Sigma, \delta, i, \{f_1, ..., f_k\})$ um AF para uma linguagem regular L. Pela definição de aceitação de um AF, temos que $L = L_1 \cup ... \cup L_k$, em que

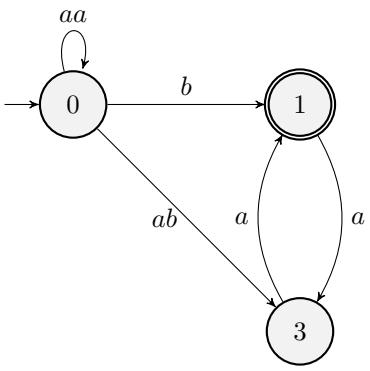
$$L_j = \{w \in \Sigma^* \mid \widehat{\delta}(i, w) = f_j\}$$

isto é, cada L_j denota a sub-linguagem cujas palavras terminam seu processamento em f_j. Dessa forma, podemos encontrar a ER correspondente a cada f_j eliminando todos os estados, exceto i e f_j. Após essa eliminação de estados, pode-se ober uma ER e_j usando os diagramas ER básicos. De posse das ER para cada um dos estados f_j, basta combiná-las para formar a ER do AF como um todo. A ER correspondente a M é $e = e_1 + ... + e_k$. □

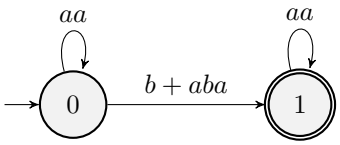
Exemplo 39. Para exemplificar a construção de uma ER a partir de um AF, considere o seguinte AF:



Removendo o estado 2, temos o seguinte diagrama ER.



Eliminando o estado 3 obtemos:



A partir deste diagrama ER básico, obtemos a ER correspondente ao AF original: $(aa)^*(b+aba)(aa)^*$. ■

8.3 Derivadas de expressões regulares

O método apresentado na seção anterior para obter um AF a partir de uma ER possui o inconveniente de produzir um AFNλ. Nesta seção apresentaremos uma maneira de obter um AFD diretamente a partir de uma expressão regular usando o conceito de *derivada*, que é muito útil para implementar algoritmos para casamento de padrão baseados em ERs.

Definição 39 (Derivada de linguagens)**.** Seja $L \subseteq \Sigma^*$ e $a \in \Sigma$. A derivada de L com respeito ao símbolo a é definida como

$$L_a = \{w \in \Sigma^* \mid aw \in L\}$$

■

Exemplo 40. Considere a seguinte linguagem $L = \{01, 10, 11, 000, 101\}$. Temos que $L_0 = \{1, 00\}$. Isso é, L_0 é o conjunto formado pelo sufixo de palavras pertencentes a L que começam com 0, sem esse símbolo. Evidentemente, se uma palavra não inicia com 0, essa não faz parte do conjunto L_0. ■

A definição acima apresenta o conceito de derivada em termos de conjuntos. Uma definição alternativa define esse conceito em termos de expressões regulares. Para isso, é necessário definir uma função que determina se uma certa expressão regular e aceita ou não a palavra vazia, λ.

Definição 40. A função ν determina se λ pertence a linguagem de uma expressão regular e.

$$\begin{array}{lcl} \nu(\emptyset) & = & \bot \\ \nu(\lambda) & = & \top \\ \nu(a) & = & \bot \\ \nu(e_1\, e_2) & = & \nu(e_1) \wedge \nu(e_2) \\ \nu(e_1 + e_2) & = & \nu(e_1) \vee \nu(e_2) \\ \nu(e^*) & = & \top \end{array}$$

■

Exemplo 41. A expressão regular $a^*(b + \lambda)$ aceita a palavra vazia, conforme mostrado pela execução da função ν a seguir.

$$\begin{array}{ll} \nu(a^*(b+\lambda)) & = \\ \nu(a^*) \wedge \nu(b+\lambda) & = \\ \top \wedge (\nu(b) \vee \nu(\lambda)) & = \\ \top \wedge (\bot \vee \top) & = \\ \top & \end{array}$$

Por sua vez, a expressão regular $(a + b)c^*$ não aceita a string vazia, uma vez que $\nu((a + b)c^*) = \bot$, conforme apresentado a seguir.

$$\begin{array}{ll} \nu((a+b)c^*) & = \\ \nu(a+b) \wedge \nu(c^*) & = \\ (\nu(a) \vee \nu(b)) \wedge \top & = \\ (\bot \vee \bot) \wedge \top & = \\ \bot & \end{array}$$

■

Utilizando a função ν, podemos definir a função $\delta_a(e)$ que calcula a derivada de uma expressão regular e com respeito a um símbolo a.

Definição 41 (Derivada de uma expressão regular)**.** A função $\delta_a(e)$ calcula a derivada de uma expressão regular e com respeito a um símbolo a e é definida como:

$$\begin{array}{lcll} \delta_a(\emptyset) & = & \emptyset & \\ \delta_a(\lambda) & = & \emptyset & \\ \delta_a(b) & = & \begin{cases} \lambda & \textit{se } a = b \\ \emptyset & \textit{caso contrário} \end{cases} & \\ \delta_a(e_1\, e_2) & = & \begin{cases} \delta_a(e_1)e_2 + \delta_a(e_2) & \textit{se } \nu(e_1) = \top \\ \delta_a(e_1)e_2 & \textit{caso contrário} \end{cases} & \\ \delta_a(e_1 + e_2) & = & \delta_a(e_1) + \delta_a(e_2) & \\ \delta_a(e^*) & = & \delta_a(e)e^* & \end{array}$$

■

Exemplo 42. Considere a seguinte expressão regular $(ab)^*$. A seguir apresentamos o cálculo de $\delta_a((ab)^*)$.

$$\begin{array}{ll} \delta_a((ab)^*) & = \\ \delta_a(ab)(ab)^* & = \\ \delta_a(a)b(ab)^* & = \\ \lambda b(ab)^* & = \\ b(ab)^* & \end{array}$$

Note que $(ab)^*$ é o conjunto de palavras formadas por repetições de ab. Logo, a derivada com repeito a a de $(ab)^*$ é o conjunto de palavras $b(ab)^*$, isto é palavras que começam com b, visto que o primeiro a é removido pela operação de derivada.

Por sua vez, $\delta_b((ab)^*) = \emptyset$, como pode ser observado pelo cálculo a seguir.

$$\begin{array}{ll} \delta_b((ab)^*) & = \\ \delta_b(ab)(ab)^* & = \\ \delta_b(a)b(ab)^* & = \\ \emptyset b(ab)^* & = \\ \emptyset & \end{array}$$

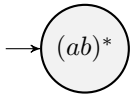
■

8.3.1 Construindo um AFD utilizando derivadas

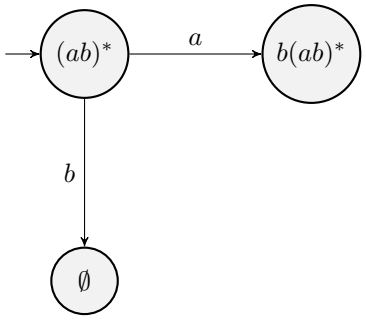
Para ilustrar a construção de um AFD a partir de uma expressão regular utilizando derivadas, usaremos o exemplo a seguir.

Exemplo 43. Considere a seguinte expressão regular: $(ab)^*$. O AFD equivalente a essa expressão pode ser construído usando derivadas da seguinte maneira.

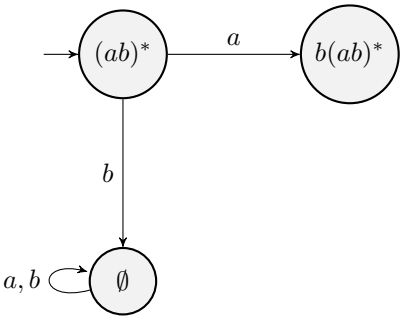
Primeiramente, temos que o estado inicial do AFD corresponde a expressão regular original.



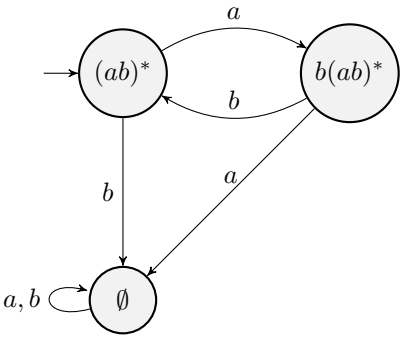
Agora, calculamos a derivada com respeito a cada símbolo do alfatebo para a expressão regular do estado inicial. Como já mostramos nos exemplos anteriores, essas são $\delta_a((ab)^*) = b(ab)^*$ e $\delta_b((ab)^*) = \emptyset$. Deste resultado, obtemos duas novas expressões regulares que serão incluídas como novos estados neste AFD. Além disso, incluiremos transições do estado inicial para estes novos estados usando o símbolo do alfabeto que foi utilizado no cálculo da derivada.



Tendo realizado todas as transições sobre o estado correspondente a $(ab)^*$, repetimos o processo para os outros estados. Evidentemente, o resultado de $\delta_a(\emptyset) = \delta_b(\emptyset) = \emptyset$.

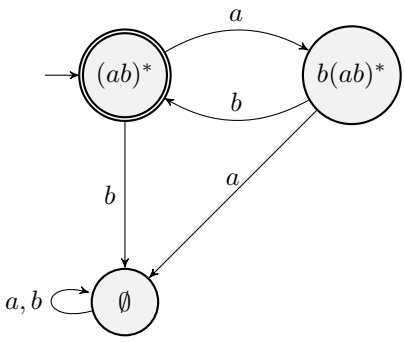


Para o estado $b(ab)^*$, temos que $\delta_a(b(ab)^*) = \emptyset$ e $\delta_b(b(ab)^*) = (ab)^*$, o que nos faz incluir novas transições no AFD anterior.



Como todos os estados possuem transiçõs para todos os símbolos do alfabeto, temos que a construção do AFD está quase concluída. Falta apenas marcar os estados finais que são aqueles cujas expressões

regulares aceitam a palavra vazia. Abaixo apresentamos a versão final do AFD para a expressão regular $(ab)^*$.



■

O exemplo mostra que o processo de construção de um AFD usando derivadas é bem simples. Porém, deve-se ter cuidado para não criar estados desnecessários. Para evitar esse problema basta adotar a seguinte relação de equivalência sintática entre ERs, que garante que o processo de cálculo de derivadas a partir de qualquer expressão regular termina em um número finito de passos.

$$\begin{array}{cc} e + (e' + e'') \approx (e + e') + e'' & e + e \approx e \\ e + e' \approx e' + e & (ee')e'' \approx e(e'e'') \\ \emptyset e \approx \emptyset & e\emptyset \approx \emptyset \\ e\lambda \approx e & \lambda e \approx e \\ e + \emptyset \approx e & \emptyset + e \approx e \end{array}$$

8.3.2 Casamento de padrão utilizando derivadas

Podemos estender a noção de derivada para palavras ao invés de símbolos de um alfabeto. A idéia é calcular a derivada para cada símbolo da palavra até que todos sejam processados. Se ao final desse processo obtermos uma expressão regular que aceita a palavra vazia, temos que a string processada pertence a linguagem da expressão regular em questão.

A seguir apresentamos essa função.

$$\begin{array}{lcl} \delta^*(e, \lambda) & = & \nu(e) \\ \delta^*(e, ay) & = & \delta^*(\delta_a(e), y) \end{array}$$

Exemplo 44. A palavra *abab* pertence à linguagem da expressão regular $(ab)^*$, conforme ilustrado pelo cálculo a seguir.

$$\begin{array}{ll} \delta^*((ab)^*, abab) & = \\ \delta^*(\delta_a((ab)^*), bab) & = \\ \delta^*(b(ab)^*, bab) & = \\ \delta^*(\delta_b(b(ab)^*), ab) & = \\ \delta^*((ab)^*, ab) & = \\ \delta^*(\delta_a((ab)^*), b) & = \\ \delta^*(b(ab)^*, b) & = \\ \delta^*(\delta_b(b(ab)^*), \lambda) & = \\ \delta^*((ab)^*, \lambda) & = \\ \nu((ab)^*) & = \\ \top & \end{array}$$

■

8.4 Exercícios

8.4.1 Exercícios de fixação

1. Para cada linguagem abaixo, apresente um AFD para a mesma. A partir do AFD apresentado, construa a ER usando a técnica do Teorema 12.
   (a) Palavras que começam e terminam com 1.
   (b) Palavras que começam e terminam com 1 e tem pelo menos um 0.
2. Utilizando a construção do Teorema 11, apresente AFs para as seguintes ERs.
   (a) $(ab)^*ac$
   (b) $(ab)^*(ba)^*$
   (c) $((aa + bb)^*cc)^*$
3. Utilizando o conceito de derivadas, apresente AFDs para as seguintes ERs.
   (a) $(ab)^*ac$
   (b) $(ab)^*(ba)^*$
   (c) $((aa + bb)^*cc)^*$

8.4.2 Exercícios de tutoria

1. Considere o problema de se estender a sintaxe de expressões regulares para prover suporte a operações de interseção e complementação. Mostre que esse tipo de expressão regular não aumenta o poder de expressões regulares. Para isso, mostre que a linguagem denotada por tais expressões é também é uma linguagem regular.
2. Obtenha AFDs, usando derivadas, para expressões regulares descrevendo as seguintes linguagens:
   (a) Palavras contendo pelo menos duas ocorrências de 0.
   (b) Palavras que não terminam em 01.

8.4.3 Exercícios suplementares

1. Uma expressão regular está na **forma normal disjuntiva** se ela está na forma normal disjuntiva $r_1 + r_2 + ... + r_n$. para algum $n \geq 1$, sendo que nenhuma das sub-expressões possui uma ocorrência de "+". Mostre que toda expressão regular é equivalente a outra na forma normal disjuntiva.
2. Implemente, em sua linguagem de programação favorita, o algoritmo para casamento de padrão utilizando derivadas de expressões regulares. *Dica*: A implementação desse algoritmo em Haskell é uma tradução da notação matemática em código.

9

Aula 9 - Gramáticas regulares e problemas de decisão para LRs.

Objetivos

- Apresentar os conceito de gramática, derivação e linguagem gerada por uma gramática.
- Apresentar a definição de gramática regular e sua equivalência com autômatos finitos.
- Apresentar alguns problemas de decisão para LRs.

9.1 Gramáticas

Definição 42 (Gramática)**.** Uma gramática $G = (V, \Sigma, R, P)$ é uma quádrupla em que:

- V: conjunto finito de variáveis.
- Σ: alfabeto
- $R \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma^*)$: conjunto de regras.
- $P \in V$: variável de partida.

■

Exemplo 45. Assim como autômatos são representados, informalmente, usando grafos dirigidos, gramáticas são representadas informalmente por uma sequência de suas regras. A definição matemática de uma gramática mostra que uma regra é apenas um par: (x, y). Porém, para melhor visualização, regras são representadas usando a seguinte notação: $x \to y$.

Abaixo apresentamos um exemplo de gramática.

$$A \quad \to \quad 0A1 \,|\, \lambda$$

A gramática acima é formada por duas regras: 1) $A \to 0A1$ e 2 $A \to \lambda$. Normalmente, omitidos o lado esquerdo de regras consecutivas que compartilham o mesmo lado esquerdo. Para isso, simplesmente usamos o símbolo $|$.

A gramática acima é formalmente representada por $(\{A\}, \{0, 1\}, R, A)$, em que $R = \{(A, 0A1), (A, \lambda)\}$.

A seguir, apresentamos o mecanismo de produção de palavras em uma gramática. ■

Definição 43 (Derivação)**.** Seja $G = (V, \Sigma, R, P)$ uma gramática qualquer. Dizemos que $x \Rightarrow y$ em G, em que $x, y \in (V \cup \Sigma)^*$, se há uma regra $u \to v \in R$ tal que u ocorre em x e y é o resultado de substituir uma ocorrência de u em x por v. A relação $\Rightarrow^n$ é definida recursivamente como:

- $x \Rightarrow^0 x$ para todo $x \in (V \cup \Sigma)^*$;
- $w \Rightarrow^n xuy$ e $u \to v \in R$ então $w \Rightarrow^{n+1} xvy$; para todo $w, x, y \in (V \cup \Sigma)^*$, $n \geq 0$.

Finalmente, dizemos que x deriva y, $x \Rightarrow^* y$, se existe $n \geq 0$ tal que $x \Rightarrow^n y$. ■

Exemplo 46. Vamos considerar a seguinte gramática:

$$A \rightarrow 0A1 \mid \lambda$$

Abaixo, mostramos a derivação de 0011.

$$\begin{array}{lll} A & \Rightarrow & \{A \rightarrow 0A1\} \\ 0A1 & \Rightarrow & \{A \rightarrow 0A1\} \\ 00A11 & \Rightarrow & \{A \rightarrow \lambda\} \\ 0011 & & \end{array}$$

■

Definição 44 (Linguagem gerada por uma gramática)**.** Seja $G = (V, \Sigma, R, P)$ uma gramática qualquer. A linguagem gerada por G, $L(G)$, é definida como:

$$L(G) = \{w \in \Sigma^* \mid P \Rightarrow^* w\}$$

■

9.2 Gramáticas regulares

Definição 45 (Gramática regular)**.** Seja $G = (V, \Sigma, R, P)$ uma gramática. Dizemos que G é uma gramática regular (GR), se todas as suas regras possuem uma das formas:

- $X \rightarrow a$
- $X \rightarrow aZ$
- $X \rightarrow \lambda$

Para $X, Z \in V$, $a \in \Sigma$. ■

Exemplo 47. A seguinte gramática

$$A \rightarrow 0A1 \mid \lambda$$

não é uma GR. Pois a regra $A \rightarrow 0A1$ não está de acordo com a Definição 45. Por sua vez, a seguinte gramática é uma GR:

$$\begin{array}{lll} A & \rightarrow & 0A \mid 0 \mid 1B \\ B & \rightarrow & 1B \mid \lambda \end{array}$$

A próxima subseção descreverá a equivalência entre GR e AFs. ■

9.2.1 Construindo um AF a partir de uma GR

Teorema 11. *Toda gramática regular gera uma linguagem regular.*

Demonstração. Seja $G = (V, \Sigma, R, P)$ uma GR. O AFN $M = (E, \Sigma, \delta, \{P\}, F)$ é tal que $L(G) = L(M)$, em que:

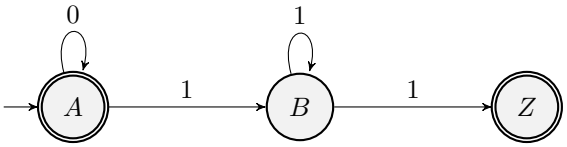
- $E = \begin{cases} V \cup \{Z\} & \text{se } R \text{ possui alguma regra } X \rightarrow a \\ V & \text{caso contrário} \end{cases}$
- $\delta(X, a) = \begin{cases} \{Y\} & \text{se } R \text{ possuir a regra } X \rightarrow aY \\ \{Z\} & \text{se } R \text{ possuir a regra } X \rightarrow a \end{cases}$
- $F = \begin{cases} \{X \mid X \rightarrow \lambda\} \cup \{Z\} & \text{se existe regra } X \rightarrow a \\ \{X \mid X \rightarrow \lambda\} & \text{caso contrário.} \end{cases}$

Pode-se mostrar que $P \Rightarrow^* w$ se e somente se $\widehat{\delta}(P, w) \cap F \neq \emptyset$ por indução sobre w. □

Exemplo 48. Considere a seguinte gramática regular:

$$\begin{array}{lcl} A & \to & 0A \,|\, 1B \,|\, \lambda \\ B & \to & 1B \,|\, 1 \end{array}$$

A partir dessa GR, podemos construir o seguinte AF:



■

9.2.2 Construindo uma GR a partir de um AF

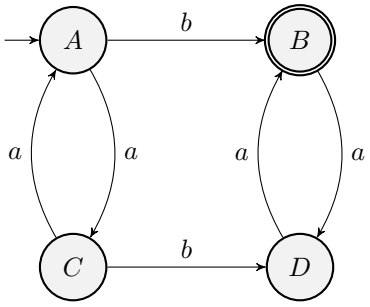
Teorema 12. *Toda linguagem regular é gerada por uma gramática regular.*

Demonstração. Seja $M = (E, \Sigma, \delta, i, F)$ um AFD. A GR $G = (E, \Sigma, R, i)$ é tal que $L(G) = L(M)$, em que:

$$R = \{e \to ae' \,|\, e' \in \delta(e, a)\} \cup \{e \to \lambda \,|\, e \in F\}$$

□

Exemplo 49. Como exemplo da construção do Teorema 12, considere o seguinte AF:



A partir deste, podemos obter a seguinte gramática regular:

$$\begin{array}{lcl} A & \to & aC \,|\, bB \\ B & \to & aD \,|\, \lambda \\ C & \to & aA \,|\, bD \\ D & \to & aB \end{array}$$

■

9.3 Problemas de decisão para LRs

Os seguintes problemas de decisão sobre LRs são decidíveis (isto é, existem algoritmos que terminam e dão a resposta correta para todas as entradas):

1. Determinar se uma LR L é infinita. Se L é uma LR, existe um AFD M para L. Sabe-se que a linguagem de um AFD é infinita se este possui um ciclo. Logo, o algoritmo para determinar se uma LR é infinita consiste em testar se o grafo correspondente ao seu AFD possui ciclos.

2. Determinar se $L(M) = \emptyset$ para um AF M. Para saber se a linguagem aceita por um AF é vazia basta determinar se existe um caminho entre o estado inicial e algum estado final, o que pode ser determinado por um algoritmo de caminhamento em grafos.

3. Dados dois AFs M_1 e M_2 determinar se $L(M_1) = L(M_2)$. Sejam M_1' o AFD mínimo equivalente a M_1 e M_2' o AFD mínimo equivalente a M_2. Temos que $L(M_1) = L(M_2)$ se, e somente se, os grafos correspondentes a M_1' e M_2' forem isomórficos.

9.4 Exercícios

9.4.1 Exercícios de fixação

1. Para cada linguagem abaixo, apresente um AFD para a mesma. A partir do AFD apresentado, construa a GR usando a técnica do Teorema 12.

   (a) Palavras que começam e terminam com 1.

   (b) Palavras que começam e terminam com 1 e tem pelo menos um 0.

2. Utilizando a construção do Teorema 11, apresente AFs para a seguinte GR.

$$\begin{array}{lcl} A & \rightarrow & 0A \,|\, 0 \,|\, 1B \\ B & \rightarrow & 1B \,|\, \lambda \end{array}$$

3. Apresente um algoritmo para decidir o seguinte problema: Dadas duas expressões regulares e_1 e e_2, decidir se $L(e_1) \subseteq L(e_2)$.

4. Apresente um algoritmo para decidir o seguinte problema: Dados dois AF's M_1 e M_2, determinar se $L(M_1)$ e $L(M_2)$ são disjuntas.

9.4.2 Exercícios de tutoria

1. Classifique as seguintes afirmativas como verdadeiras ou falsas. Justifique sua resposta.

   (a) Existe linguagem infinita que pode ser reconhecida por um AFD de apenas um estado.

   (b) Existe linguagem finita que pode ser reconhecida por um AFD de, no mínimo, um trilhão de estados.

   (c) Se uma linguagem pode ser reconhecida por um AFD, qualquer subconjunto dela também pode.

   (d) Se uma linguagem não pode ser reconhecida por um AFD e ela é subconjunto de L, então L também não pode ser reconhecida por um AFD.

2. Gramáticas lineares à direita são gramáticas em que todas as regras possuem uma das seguintes formas:

$$\begin{array}{lcl} X & \rightarrow & \lambda \\ X & \rightarrow & Ya \\ X & \rightarrow & a \end{array}$$

   em que X, Y são variáveis e $a \in \Sigma$. Mostre que gramáticas lineares à direita geram linguagens regulares.

9.4.3 Exercícios suplementares

1. Sejam M_C um AFD "correto" e M_E um AFD "incorreto" para uma certa linguagem regular L. Descreva métodos (algoritmos) para determinar:

   (a) Quais palavras M_E deveria reconhecer e não reconhece.

   (b) Quais palavras M_E reconhece e não deveria reconhecer.

2. Seja $L \neq \emptyset$ uma linguagem regular em que toda palavra $w \in L$ é tal que $|w| \geq k$, para algum $k \in \mathbb{N}$. Mostre que L não pode ser reconhecida por um AFD com menos que $k+1$ estados.

Parte II

Linguagens livres de contexto

10

Aula 10 - Autômatos de pilha

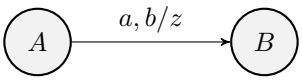
Objetivos

- Apresentar os conceitos de autômatos de pilha determinísticos e transições compatíveis.
- Apresentar o conceito de linguagem aceita por um autômato de pilha.
- Exemplos de autômatos de pilha determinísticos.

10.1 Introdução

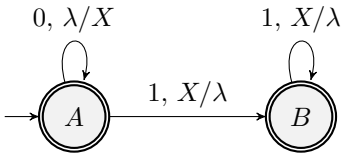
Conforme apresentado em aulas passadas, linguagens regulares não são capazes de representar a estrutura de linguagens muito simples, como por exemplo, $\{0^n 1^n \mid n \geq 0\}$. Isso se deve ao fato de que AF não possuem "memória" suficiente para armazenar o fato de que uma quantidade arbitrária de 0's foi processada e usar esse fato para verificar se a quantidade de 1's é a mesma. Como resolver esse problema?

Bem, a solução óbvia é "adicionar mais memória". Os autômatos de pilha (AP)são, informalmente, autômatos finitos com uma pilha ilimitada. Uma transição em um AP possui o seguinte formato geral:



que pode ser formalmente descrito como $\delta(A, a, b) = [B, z]$. Informalmente, a transição sai do estado A processando a na palavra de entrada, com b no topo da pilha, resulta no estado B e empilha z após desempilhar b. A seguir apresentamos um exemplo concreto de AP e como a execução deste procede.

Exemplo 50. Considere o seguinte diagrama de estados para um AP que aceita $\{0^n 1^n \mid n \geq 0\}$.



Para exemplificar o funcionamento deste AP, vamos utilizar a noção de configuração instântanea, que consiste de uma tripla formada por um estado, a palavra atual e o estado da pilha. Representaremos a pilha como uma palavra sobre um alfabeto Γ de pilha. Em nosso exemplo, o alfabeto de pilha é $\Gamma = \{X\}$ e a pilha vazia é representada por λ.

De maneira intuitiva, dizemos que uma palavra $w \in \Sigma^*$ pertence a linguagem de um AP M se esta iniciando seu processamento no estado inicial o termina com pilha vazia em estado final tendo sido completamente consumida. Apresentaremos alguns exemplos de transições nesse AP usando configurações

instântaneas. Primeiramente, vamos mostrar o processamento de uma palavra aceita 0011:

$$\begin{array}{lll} [A, 0011, \lambda] & \vdash & \{\delta(A, 0, \lambda) = [A, X]\} \\ [A, 011, X] & \vdash & \{\delta(A, 0, \lambda) = [A, X]\} \\ [A, 11, XX] & \vdash & \{\delta(A, 1, X) = [B, \lambda]\} \\ [B, 1, X] & \vdash & \{\delta(B, 1, X) = [B, \lambda]\} \\ [B, \lambda, \lambda] & & \end{array}$$

Logo, como $[A, 0011, \lambda] \vdash^* [B, \lambda, \lambda]$, temos que 0011 é aceita por esse AP.

Agora, vamos considerar dois exemplos de palavras recusadas pelo AP apresentado. Primeiro, vamos considerar a palavra 001. O processamento desta é apresentado a seguir.

$$\begin{array}{lll} [A, 001, \lambda] & \vdash & \{\delta(A, 0, \lambda) = [A, X]\} \\ [A, 01, X] & \vdash & \{\delta(A, 0, \lambda) = [A, X]\} \\ [A, 1, XX] & \vdash & \{\delta(A, 1, X) = [B, \lambda]\} \\ [B, \lambda, X] & & \end{array}$$

Neste caso, a palavra não é aceita por ter seu processamento finalizado com a pilha não vazia. Outro exemplo de não aceitação acontece quando a palavra de entrada não é completamente processada pelo AP. A palavra 011 é um exemplo deste caso. Sua execução é apresentada abaixo.

$$\begin{array}{lll} [A, 011, \lambda] & \vdash & \{\delta(A, 0, \lambda) = [A, X]\} \\ [A, 11, X] & \vdash & \{\delta(A, 1, X) = [B, \lambda]\} \\ [B, 1, \lambda] & & \end{array}$$

■

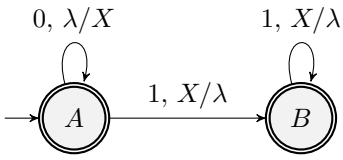
10.2 Autômatos de pilha determinísticos

Definição 46 (Transições compatíveis)**.** Seja $\delta : E \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \to E \times \Gamma^*$. Duas transições $\delta(e, a, b)$ e $\delta(e', a', b')$ são compatíveis se, e somente se:

$$(a = a' \vee a = \lambda \vee a' = \lambda) \wedge (b = b' \vee b = \lambda \vee b' = \lambda)$$

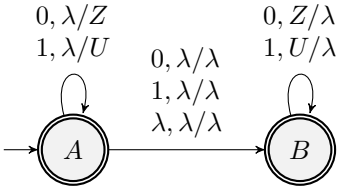
■

Exemplo 51. Vamos considerar novamente o APD para $\{0^n 1^n \mid n \geq 0\}$:



Somente o estado A possui duas transições e, portanto, tem a possibilidade de possuir transições compatíveis. Porém, as transições $\delta(A, 0, \lambda) = [A, X]$ e $\delta(A, 1, X) = [B, \lambda]$ não são compatíveis, visto que operam sobre diferentes símbolos do alfabeto de entrada.

Como um exemplo de AP possuindo transições compatíveis considere:



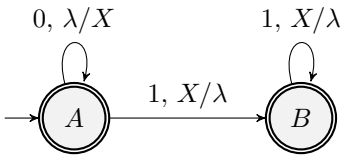
Observe que as transições $\delta(A, 0, \lambda) = [B, \lambda]$ e $\delta(A, \lambda, \lambda) = [B, \lambda]$ são compatíveis, como pode ser facilmente verificado. ■

Definição 47 (Autômato de pilha determinístico)**.** Um autômato de pilha determinístico (APD) é uma sêxtupla $M = (E, \Sigma, \Gamma, \delta, i, F)$ em que:

- E: conjunto de estados.
- Σ: alfabeto de entrada.
- Γ: alfabeto de pilha.
- $\delta : E \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \to E \times \Gamma^*$: função de transição. Uma função parcial sem transições compatíveis.
- $i \in E$: estado inicial.
- $F \subseteq E$: conjunto de estados finais.

■

Exemplo 52. Vamos considerar novamente o APD para $\{0^n 1^n \mid n \geq 0\}$:



Como discutido em um exemplo anterior, este AP não possui transições compatíveis e, portanto, é determinístico. ■

Definição 48 (Transição entre configurações instântaneas)**.** Seja $M = (E, \Sigma, \Gamma, \delta, i, F)$ um APD. A relação $\vdash (E \times \Sigma^* \times \Gamma^*)^2$ para M é tal que para todo $e \in E$, $a \in \Sigma \cup \{\lambda\}$, $b \in \Gamma \cup \{\lambda\}$ e $x \in \Gamma^*$:

$$[e, ay, bz] \vdash [e', y, xz]$$

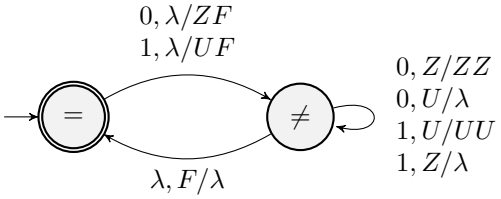
sempre que $\delta(e, a, b) = [e', x]$. Como usual, $\vdash^*$ irá denotar o fecho transitivo e reflexivo de $\vdash$. ■

Definição 49 (Linguagem aceita por um AP)**.** Seja $M = (E, \Sigma, \Gamma, \delta, i, F)$ um APD. A linguagem aceita por M, $L(M)$, é definida como:

$$L(M) = \{w \in \Sigma^* \mid \exists e.e \in F \wedge [i, w, \lambda] \vdash^* [e, \lambda, \lambda]\}$$

■

Exemplo 53. Considere a linguagem de palavras sobre $\Sigma = \{0, 1\}$ que possuem o mesmo número de 0s e de 1s. O APD para essa linguagem é apresentado a seguir.



Neste exemplo, o APD usa um símbolo (F) para marcar o final da pilha. Como a cada 0 encontrado empilha-se um novo Z ou desempilha-se um U (o mesmo raciocínio aplica-se para 1), temos que a pilha está vazia apenas quando um prefixo contendo a mesma quantidade de 0s e de 1s foi processado. Para detectarmos essa fato, usamos o símbolo F como o fundo de pilha. ■

10.3 Exercícios

10.3.1 Exercícios de Fixação

1. Construa APDs para as seguintes linguagens.
   (a) $\{0^n 1^{2n} \mid n \geq 0\}$.
   (b) $\{0^{3n} 1^{2n} \mid n \geq 0\}$.
   (c) $\{w0w^R \mid w \in \{1,2\}^*\}$.
   (d) $\{0^m 1^n \mid m < n\}$.

11

Aula 11 - Autômatos de pilha não determinísticos

Objetivos

- Apresentar os conceito de autômatos de pilha determinísticos não determinístico (APNs).
- Apresentar o critérios alternativos de aceitação por APNs.
- Demonstrar a equivalência entre os critérios de aceitação

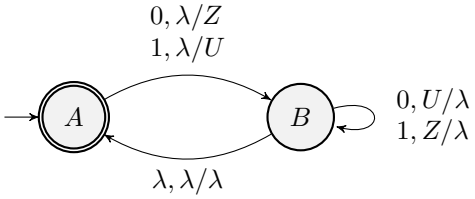
11.1 Autômatos de pilha não determinísticos

Definição 50 (Autômato de pilha não determinístico)**.** Um autômato de pilha não determinístico (APN) é uma sêxtupla $M = (E, \Sigma, \Gamma, \delta, I, F)$ em que:

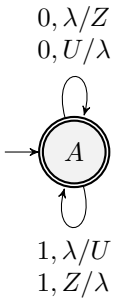
- E: conjunto de estados.
- Σ: alfabeto de entrada.
- Γ: alfabeto de pilha.
- $\delta : E \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \to D$, $D \subseteq E \times \Gamma^*$, D finito: função de transição. Uma função parcial.
- $I \subseteq E$: conjunto de estados iniciais.
- $F \subseteq E$: conjunto de estados finais.

■

Exemplo 54. Considere a linguagem de palavras sobre $\Sigma = \{0, 1\}$ que possuem o mesmo número de 0s e de 1s. A seguir apresentaremos um APN que aceita essa linguagem.



Note que esse APN é apenas uma simplificação do APD visto na aula 10. Apesar de mais compacto, esse APN pode ser simplificado ainda mais. O seguinte APN de um único estado reconhece a mesma linguagem.



■

11.2 Critérios alternativos de reconhecimento

Na aula 10, vimos um critério de reconhecimento para um APD $M = (E, \Sigma, \Gamma, \delta, i, F)$, que repetimos abaixo:

$$L(M) = \{w \in \Sigma^* \mid \exists e.e \in F \wedge [i, w, \lambda] \vdash^* [e, \lambda, \lambda]\}$$

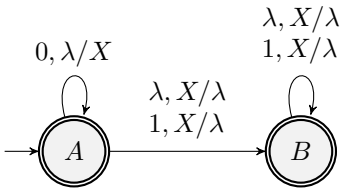
Quando pensamos em APNs, esse critério é conhecido por aceitação por estado final e pilha vazia, conforme a próxima definição.

Definição 51 (Aceitação por estado final e pilha vazia)**.** Seja $M = (E, \Sigma, \Gamma, \delta, I, F)$ um APN. A linguagem de aceita por M por estado final e pilha vazia é:

$$L(M) = \{w \in \Sigma^* \mid \exists i.\exists e.i \in I \wedge e \in F \wedge [i, w, \lambda] \vdash^* [e, \lambda, \lambda]\}$$

■

Exemplo 55. Apresentaremos um APN que aceita por estado final e pilha vazia a linguagem $\{0^m 1^n \mid m \geq n\}$.



■

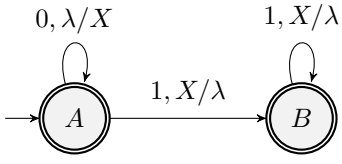
Um possível critério alternativo de aceitação é eliminar a restrição de toda computação terminar com pilha vazia. Esse critério é chamado de aceitação por estado final.

Definição 52 (Aceitação por estado final)**.** Seja $M = (E, \Sigma, \Gamma, \delta, I, F)$ um APN. A linguagem de aceita por M por estado final é:

$$L_F(M) = \{w \in \Sigma^* \mid \exists i.\exists e.\exists y.i \in I \wedge e \in F \wedge y \in \Gamma^*. \wedge [i, w, \lambda] \vdash^* [e, \lambda, y]\}$$

■

Exemplo 56. Apresentaremos um APN que aceita por estado final a linguagem $\{0^m 1^n \mid m \geq n\}$.



■

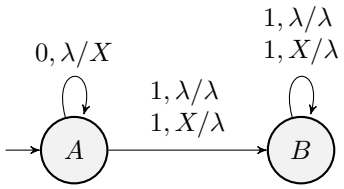
O último critério alternativo aceita palavras cujo processamento pára em qualquer estado com pilha vazia. Este critério é conhecido como aceitação por pilha vazia.

Definição 53 (Aceitação por pilha vazia)**.** Seja $M = (E, \Sigma, \Gamma, \delta, I, F)$ um APN. A linguagem de aceita por M por pilha vazia é:

$$L_V(M) = \{w \in \Sigma^* \mid \exists i.i \in I \wedge e \in E \wedge [i, w, \lambda] \vdash^* [e, \lambda, \lambda]\}$$

■

Exemplo 57. Apresentaremos um APN que aceita por pilha vazia a linguagem $\{0^m 1^n \mid m \leq n\}$.



■

Teorema 13. *Seja L uma linguagem. As seguintes afirmativas são equivalentes:*

1. *L pode ser reconhecida por pilha vazia e estado final.*
2. *L pode ser reconhecida por estado final.*
3. *$L \cup \{\lambda\}$ pode ser reconhecida por pilha vazia.*

Demonstração. $1 \rightarrow 2$: Seja um APN $M = (E, \Sigma, \Gamma, \delta, I, F)$. Podemos obter um APN M', de forma que $L_F(M') = L(M)$, em que $M' = (E \cup \{i', g\}, \Sigma, \Gamma \cup \{F\}, \delta', \{i'\}, \{g\})$ de forma que δ' inclui δ mais as seguintes transições:

1. para cada $i_k \in I$, $\delta'(i', \lambda, \lambda) = \{[i_k, F]\}$.
2. para cada $f_j \in F$, $\delta'(f_j, \lambda, F) = \{[g, \lambda]\}$.

em que $i', g \notin E$ e $F \notin \Gamma$.
$2 \rightarrow 3$: Seja um APN $M = (E, \Sigma, \Gamma, \delta, I, F)$. Podemos obter um APN M' de forma que $L_v(M') = L_F(M) \cup \{\lambda\}$, em que $M' = (E \cup \{i', g, h\}, \Sigma, \Gamma \cup \{F\}, \delta', \{i'\})$ de forma que δ' inclui δ mais as seguintes transições:

1. Para cada $i_k \in I$, $\delta'(i', \lambda, \lambda) = \{[i_k, F]\}$.
2. Para cada $f_j \in F$, $\delta'(f_j, \lambda, \lambda) = \{[g, \lambda]\}$.
3. Para cada $X \in \Gamma$, $\delta'(g, \lambda, X) = \{[g, \lambda]\}$.
4. $\delta'(g, \lambda, F) = \{[h, \lambda]\}$

$3 \rightarrow 1$: Seja um APN M que aceita uma linguagem por pilha vazia. Para obtermos outro APN que aceita por pilha vazia e estado final, basta fazer todos os estados de M finais. □

11.3 Exercícios

11.3.1 Exercícios de Fixação

1. Construa APNs que aceitem por pilha vazia e estado final para as seguintes linguagens.
   (a) $\{0^n 1^n \mid n \geq 0\} \cup \{0^n 1^{2n} \mid n \geq 0\}$
   (b) $\{0^n 1^n 0^k \mid n, k \geq 0\}$
   (c) $\{0^n 1^m \mid m > n\}$
2. Construa APNs que aceitem $\{0^n 1^n \mid n \geq 0\}$:
   (a) Por estado final.
   (b) Por pilha vazia.

11.3.2 Exercícios de Tutoria

1. Explique como construir um APN de um único estado equivalente a um AFD.

12

Aula 12 - Gramáticas livres de contexto

Objetivos

- Apresentar os conceito de gramática livre de contexto.
- Apresentar os conceitos de derivação, derivação mais à esquerda e mais à direita.
- Apresentar o conceito de árvore de derivação.
- Apresentar o conceito de ambiguidade.
- Definir linguagens livres de contexto.

12.1 Gramáticas livres de contexto

Definição 54 (Gramática livre de contexto)**.** Uma gramática é dita ser livre de contexto $G = (V, \Sigma, R, P)$ (GLC) se toda regra possui a forma $X \to w \in R$, em que $X \in V$ e $w \in (V \cup \Sigma)^*$. ■

Exemplo 58. Como um exemplo de GLC, considere a seguinte gramática que gera $\{0^n 1^n \mid n \geq 0\}$.

$$P \to 0P1 \mid \lambda$$

Por exemplo, podemos mostrar que $P \Rightarrow^* 0011$ usando a seguinte derivação.

$$\begin{array}{lcl} P & \Rightarrow & \{P \to 0P1\} \\ 0P1 & \Rightarrow & \{P \to 0P1\} \\ 00P11 & \Rightarrow & \{P \to \lambda\} \\ 0011 & & \end{array}$$

■

Exemplo 59. Outro exemplo de GLC é a para a linguagem de palíndromos sobre $\{0, 1\}$.

$$P \to 0P0 \mid 1P1 \mid 0 \mid 1 \mid \lambda$$

Note que as regras dessa gramática mostram a estrutura recursiva de palíndromos. Como caso base, temos os palíndromos $\lambda, 0$ e 1. Palíndromos maiores são criados concatenando um 0 (1) à esquerda e a direita de um palíndromo. ■

Exemplo 60. Considere a linguagem sobre $\Sigma = \{0, 1\}$ em que toda palavra possui o mesmo número de 0s e de 1s.

$$P \to 0P1P \mid 1P0P \mid \lambda$$

Novamente, temos que as regras dessa gramática exibem a mesma estrutura de uma definição recursiva para essa linguagem. São 3 casos, a saber:

1. λ, que é a menor palavra com o mesmo número de 0s e de 1s.

2. A palavra começa com um 0: Se a palavra começa com um 0, este possui um 1 correspondente em algum ponto da palavra. O mesmo vale para o caso da palavra iniciar com 1.

■

Exemplo 61. A gramática a seguir descreve a estrutura de expressões aritméticas encontradas em linguagens de programação.

$$E \to E + T \,|\, T$$
$$T \to T * F \,|\, F$$
$$F \to (E) \,|\, n$$

em que n denota um número qualquer. Note que o alfabeto dessa gramática é $\Sigma = \{(,), +, *, n\}$. ■

Definição 55 (Linguagem livre de contexto)**.** Dizemos que uma linguagem $L \subseteq \Sigma^*$ é uma linguagem livre de contexto (LLC) se existe uma gramática livre de contexto que a gera. ■

12.2 Derivações e ambiguidade

Definição 56 (Forma sentencial)**.** Seja $G = (V, \Sigma, R, P)$ uma gramática. Uma forma sentencial para G é uma palavra $w \in (V \cup \Sigma)^*$. ■

Definição 57 (Árvore de derivação)**.** Seja $G = (V, \Sigma, R, P)$ uma GLC. Uma árvore de derivação (AD) de uma forma sentencial de G é uma árvore ordenada construída recursivamente como se segue:

1. uma árvore cujo único vértice possui o rótulo P é uma AD para P;

2. Se $X \in V$ é o rótulo de uma folha f de uma AD A, então:

   (a) Se $X \to \lambda \in R$, então a árvore de derivação obtida adicionando mais um vértice v com rótulo λ e uma aresta $\{f, v\}$ é uma AD.

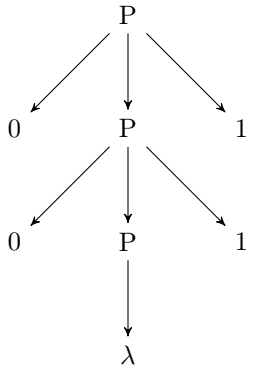
   (b) Se $X \to x_1...x_n$, $x_1...x_n \in V \cup \Sigma$, então a árvore obtida adicionando a A n vértices $v_1, ..., v_n$ com os rótulos $x_1, ..., x_n$ e as arestas $\{f, v_1\}, ..., \{f, v_n\}$ é uma AD.

■

Exemplo 62. Para exemplificar o conceito de árvore de derivação, vamos considerar a seguinte gramática para $\{0^n 1^n \,|\, n \geq 0\}$.

$$P \to 0P1 \,|\, \lambda$$

A árvore de derivação para 0011 será:



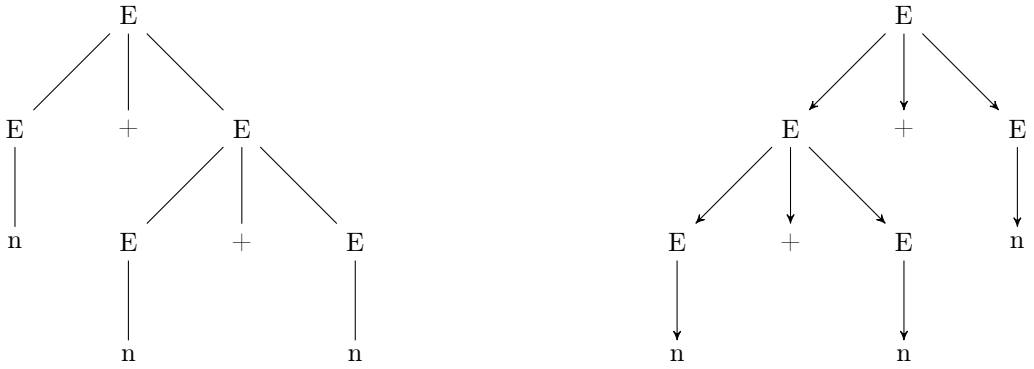
■

Definição 58 (Gramática ambígua)**.** Dizemos que uma GLC G é ambígua se esta possui mais de uma AD para alguma sentença que ela gera. ■

Exemplo 63. Vamos considerar uma outra GLC para expressões aritméticas.

$$E \to E + E \,|\, E * E \,|\, (E) \,|\, n$$

A seguir, apresentamos duas árvores de derivação distintas para $n + n + n$.



Como a gramática acima gera duas distintas ADs para a mesma sentença, temos que essa é ambígua. ■

Definição 59 (Derivação mais a esquerda (DME) e Derivação mais a direita (DMD))**.** Uma derivação é dita mais a esquerda (mais a direita) se a cada passo é expandida a variáveis mais a esquerda (mais a direita). Usa-se o símbolo $\Rightarrow_E$ para representar uma DME e $\Rightarrow_D$ para uma DMD.

Podemos dizer que uma gramática é ambígua se existem duas ou mais DMEs ou DMDs para uma mesma sentença. ■

Exemplo 64. Vamos considerar novamente a seguinte GLC para expressões aritméticas

$$E \to E + E \,|\, E * E \,|\, (E) \,|\, n$$

e a seguinte palavra: $n + n + n$. Apresentaremos duas derivações mais a direita para essa palavra.

A primeira derivação é:

$$\begin{array}{lll} E & \Rightarrow_D & \{E \to E + E\} \\ E + E & \Rightarrow_D & \{E \to E + E\} \\ E + E + E & \Rightarrow_D & \{E \to n\} \\ E + E + n & \Rightarrow_D & \{E \to n\} \\ E + n + n & \Rightarrow_D & \{E \to n\} \\ n + n + n & & \end{array}$$

A segunda é:

$$\begin{array}{lll} E & \Rightarrow_D & \{E \to E + E\} \\ E + E & \Rightarrow_D & \{E \to n\} \\ E + n & \Rightarrow_D & \{E \to E + E\} \\ E + E + n & \Rightarrow_D & \{E \to n\} \\ E + n + n & \Rightarrow_D & \{E \to n\} \\ n + n + n & & \end{array}$$

O que também mostra que a gramática acima é ambígua. ■

Definição 60 (Linguagem inerentemente ambígua)**.** Dizemos que uma linguagem é inerentemente ambígua se existem apenas gramáticas livres de contexto ambíguas para ela. ■

Exemplo 65. A seguinte linguagem é inerentemente ambígua:

$$\{a^n b^m c^k \,|\, n = m \vee n = k\}$$

Palavras do formato $a^n b^n c^n$, $n \geq 0$, possuem mais de uma AD. ■

12.3 Exercícios

12.3.1 Exercícios de Fixação

1. Construa GLCs para as seguintes linguagens.
   (a) $\{0^n 1^n \mid n \geq 0\} \cup \{0^n 1^{2n} \mid n \geq 0\}$
   (b) $\{0^n 1^n 0^k \mid n, k \geq 0\}$
   (c) $\{0^n 1^m \mid m > n\}$
   (d) $\{a^n b^m c^k \mid n = m \vee n = k\}$
2. Considerando a gramática que você construiu para $\{a^n b^m c^k \mid n = m \vee n = k\}$ no item d) do exercício anterior, apresente duas AD e derivações mais a esquerda e a direita para a palavra $a^2 b^2 c^2$ mostrando que sua gramática é ambígua.

13

Aula 13 - Manipulação de gramáticas livres de contexto

Objetivos

- Apresentar os algoritmos para remoção de símbolos inúteis, regras λ, eliminar regras encadeadas e regras recursivas à esquerda.

13.1 Remoção de símbolos inúteis

Definição 61 (Variável útil)**.** Seja uma GLC $G = (V, \Sigma, R, P)$. Dizemos que $X \in V$ é útil se existem $u, v \in (V \cup \Sigma)^*$ e $w \in \Sigma^*$ tais que

$$P \Rightarrow^* uXv \Rightarrow^* w$$

■

Exemplo 66. Considere a seguinte GLC $G = (\{P, A, B, C\}, \{a, b, c\}, R, P)$ em que R é dado por:

$$\begin{aligned} P &\to AB \mid a \\ B &\to c \\ C &\to c \end{aligned}$$

Observe que, de acordo com a Definição 61:

- C é inútil: não existem u, v tais que $P \Rightarrow^* uCv$.
- A é inútil: não existe w tal que $A \Rightarrow^* w$.
- B é inútil: pois somente ocorre em uma regra contendo outra variável inútil (A).

Eliminado todas as variáveis inúteis e os símbolos do alfabeto somente referenciados por elas, temos a seguinte gramática equivalente.

$$G = (\{P\}, \{a\}, \{P \to a\}, P)$$

■

A seguir apresentaremos o método para remoção de símbolos inúteis. Porém, antes faz-se necessário introduzir a notação $\Rightarrow_G$ que especifica que uma derivação está sendo feita usando as regras da gramática G.

Teorema 14. *Seja G uma gramática tal que $L(G) \neq \emptyset$. Então, existe uma GLC G', equivalente a G, sem variáveis inúteis.*

Demonstração. Seja $G = (V, \Sigma, R, P)$ uma GLC tal que $L(G) \neq \emptyset$. Pode-se construir uma GLC G_2, equivalente a G, sem variáveis inúteis seguindo os passos:

1. Obtenha $G_1 = (V_1, \Sigma, R_1, P)$, em que:
   - $V_1 = \{X \in V \mid X \Rightarrow_G^* w \wedge w \in \Sigma^*\}$
   - $R_1 = \{r \in R \mid r \text{ não contém símbolo de } V - V_1\}$
2. Obtenha $G_2 = (V_2, \Sigma, R_2, P)$, a partir de G_1, em que:
   - $V_2 = \{X \in V \mid P \Rightarrow_{G_1}^* uXv\}$
   - $R_1 = \{r \in R \mid r \text{ não contém símbolo de } V_1 - V_2\}$

□

Exemplo 67. Para exemplificar a construção de uma gramática sem símbolos inúteis, considere a seguinte GLC $G = (\{A, B, C, D, E, F\}, \{0, 1\}, R, A)$, em que R é formado pelas segras a seguir.

$$\begin{aligned} &A \to ABC \mid AEF \mid BD \\ &B \to B0 \mid 0 \\ &C \to 0C \mid EB \\ &D \to 1D \mid 1 \\ &E \to EB \\ &F \to 1F1 \mid 1 \end{aligned}$$

Ao aplicarmos o primeiro passo do Teorema 14, obtemos o seguinte conjunto $V_1 = \{B, D, F, A\}$. Logo, a GLC G_1 possui as seguintes regras:

$$\begin{aligned} &A \to BD \\ &B \to B0 \mid 0 \\ &D \to 1D \mid 1 \\ &F \to 1F1 \mid 1 \end{aligned}$$

Usando o segundo passo do referido teorema, obtemos o conjunto $V_2 = \{A, B, D\}$ e as seguintes regras:

$$\begin{aligned} &A \to BD \\ &B \to B0 \mid 0 \\ &D \to 1D \mid 1 \end{aligned}$$

■

13.2 Eliminação de uma regra $X \to w$

Teorema 15. *Seja uma GLC $G = (V, \Sigma, R, P)$ e $X \to w \in R$, $X \neq P$. Existe uma GLC $G_1 = (V, \Sigma, R_1, P)$ equivalente a G sem a regra $X \to w$.*

Demonstração. A gramática G_1 possui o seguinte conjunto de regras R_1, obtido a partir de R:

1. para cada regra $Y \to z \in R$, para cada forma de escrever $x_1 X x_2 ... X_{n+1}$, em que $n \geq 0$ e $x_i \in (V \cup \Sigma)^*$, coloque a regra $Y \to x_1 w x_2 w ... w x_{n+1}$ em R_1.
2. Retire $X \to w$ de R_1.

□

Exemplo 68. Considere a gramática $G = (\{P, A, B\}, \{a, b, c\}, R, P)$ em que R é formado pelas regras:

$$\begin{aligned} &P \to ABA \\ &A \to aA \mid a \\ &B \to bBc \mid \lambda \end{aligned}$$

Considere a necessidade de eliminar a regra $A \to a$. Primeiramente, a regra $P \to ABA$ dá origem as seguintes regras:

$$P \to ABA \mid aBA \mid ABa \mid aBa$$

e a regra $A \to aA$ dá origem a:

$$A \to aA \,|\, aa$$

Com isso, a gramática resultante é formada pelas seguintes regras.

$$\begin{array}{l} P \to ABA \,|\, aBA \,|\, ABa \,|\, aBa \\ A \to aA \,|\, aa \\ B \to bBc \,|\, \lambda \end{array}$$

■

13.3 Eliminação de variáveis anuláveis

Definição 62. Seja uma GLC $G = (V, \Sigma, R, P)$. Dizemos que uma variável $X \in V$ é anulável se $X \Rightarrow^* \lambda$. O conjunto de variáveis anuláveis de G, $\mathcal{A}_G$, pode ser definido indutivamente da seguinte maneira:

1. $\mathcal{A}_G = \{X \in V \,|\, X \to \lambda \in R\}$.
2. $\mathcal{A}_G = \{X \in V \,|\, X \to z \wedge z \in \mathcal{A}_G^*\}$.

■

Exemplo 69. Considere a seguinte GLC:

$$\begin{array}{l} P \to ABC \,|\, C \\ A \to AaaA \,|\, \lambda \\ B \to BBb \,|\, C \\ C \to cC \,|\, \lambda \end{array}$$

Pela Definição 62, identificamos o primeiro passo de $\mathcal{A}_G = \{A, C\}$. Pelo segundo passo, temos que $\mathcal{A}_G = \{A, C, P, B\}$. ■

Teorema 16. *Para cada GLC existe uma GLC equivalente cuja única regra λ, se houver, é $P \to \lambda$, em que P é o símbolo de partida.*

Demonstração. Seja uma GLC $G = (V, \Sigma, R, P)$. Seja a GLC $G_1 = (V, \Sigma, R_1, P)$ em que R_1 é obtido da seguinte forma:

1. para cada regra $Y \to z \in R$, para cada forma de escrever z como $x_1X_1x_2...X_nx_{n+1}$, $n \geq 0$, $x_i \in (V \cup \Sigma)^*$, $x_1x_2...x_{n+1} \neq \lambda$ e $X_i \in \mathcal{A}_G$, coloque a regra $Y \to x_1x_2...x_{n+1}$ em R_1.
2. Se $P \in \mathcal{A}_G$, então $P \to \lambda \in R_1$.

□

Exemplo 70. Vamos retomar o exemplo anterior e eliminar as variáveis anuláveis da seguinte gramática:

$$\begin{array}{l} P \to ABC \,|\, C \\ A \to AaaA \,|\, \lambda \\ B \to BBb \,|\, C \\ C \to cC \,|\, \lambda \end{array}$$

Conforme apresentado anteriormente, temos que $\mathcal{A}_G = \{A, C, P, B\}$. Aplicando o exposto no teorema anterior, obtemos a seguinte gramática.

$$\begin{array}{l} P \to ABC \,|\, AB \,|\, BC \,|\, AC \,|\, A \,|\, B \,|\, C \,|\, \lambda \\ A \to AaaA \,|\, aaA \,|\, Aaa \,|\, aa \\ B \to BBb \,|\, C \,|\, Bb \,|\, b \\ C \to cC \,|\, c \end{array}$$

■

13.4 Eliminação de variáveis encadeadas

Definição 63 (Variável encadeada)**.** Seja uma GLC $G = (V, \Sigma, R, P)$. Dizemos que uma variável $Z \in V$ é encadeada a $X \in V$ se:

- $X \to Z \in R$;
- $X \to Y$ e Z é encadeada a Y.

■

Exemplo 71. Vamos considerar a seguinte gramática para expressões aritméticas:

$$\begin{aligned} E &\to E + T \,|\, T \\ T &\to T * F \,|\, F \\ F &\to (E) \,|\, n \end{aligned}$$

Abaixo listamos o conjunto de variáveis encadeadas a cada uma das variáveis dessa gramática.

$$\begin{aligned} enc(F) &= \{F\} \\ enc(T) &= \{T, F\} \\ enc(E) &= \{E, T, F\} \end{aligned}$$

■

Teorema 17. *Seja uma GLC $G = (V, \Sigma, R, P)$. Existe uma GLC, equivalente a G, sem variáveis encadeadas.*

Demonstração. A gramática equivalente a G é $G_1 = (V, \Sigma, R_1, P)$, em que

$$R_1 = \{X \to w \,|\, Y \in enc(X), Y \to w \in R \wedge w \notin V\}$$

□

Exemplo 72. Vamos considerar a seguinte gramática para expressões aritméticas:

$$\begin{aligned} E &\to E + T \,|\, T \\ T &\to T * F \,|\, F \\ F &\to (E) \,|\, n \end{aligned}$$

Abaixo listamos o conjunto de variáveis encadeadas a cada uma das variáveis dessa gramática.

$$\begin{aligned} enc(F) &= \{F\} \\ enc(T) &= \{T, F\} \\ enc(E) &= \{E, T, F\} \end{aligned}$$

A gramática equivalente sem variáveis encadeadas é:

$$\begin{aligned} E &\to E + T \,|\, T * F \,|\, (E) \,|\, n \\ T &\to T * F \,|\, (E) \,|\, n \\ F &\to (E) \,|\, n \end{aligned}$$

■

13.5 Eliminação de regras recursivas à esquerda

Definição 64 (Regra recursiva à esquerda)**.** Seja uma GLC $G = (V, \Sigma, R, P)$. Dizemos que a regra $A \to Ay$, $A \in V$, $y \in (V \cup \Sigma)^*$ é recursiva à esquerda em G. ■

Teorema 18. *Para qualquer GLC existe uma GLC equivalente sem regras recursivas à esquerda.*

Demonstração. Sejam as seguintes todas as regras X para G:

$$X \to Xy_1 \,|\, Xy_2 \,|\, ... \,|\, Xy_n \,|\, w_1 \,|\, w_2 \,|\, ... \,|\, w_k$$

Para eliminar a recursão a esquerda basta substituir as regras X por (Z é uma nova variável):

$$X \to w_1 \,|\, w_2 \,|\, ... \,|\, w_k \,|\, w_1Z \,|\, w_2Z \,|\, ... \,|\, w_kZ$$
$$Z \to y_1 \,|\, y_2 \,|\, ... \,|\, y_n \,|\, y_1Z \,|\, y_2Z \,|\, ... \,|\, y_nZ$$

□

Exemplo 73. Vamos considerar a tarefa de remover regras recursivas à esquerda da seguinte GLC:

$$E \to E + E \,|\, E * E \,|\, (E) \,|\, n$$

A GLC equivalente sem regras recursivas a esquerda é obtida usando o Teorema 18.

$$E \to (E) \,|\, n \,|\, (E)Z \,|\, nZ$$
$$Z \to +E \,|\, *E \,|\, +EZ \,|\, *EZ$$

■

13.6 Exercícios

13.6.1 Exercícios de Fixação

1. Construa uma gramática equivalente sem variáveis anuláveis.

$$P \to BPA \,|\, A$$
$$A \to aA \,|\, \lambda$$
$$B \to Bba \,|\, \lambda$$

2. Construa uma gramática equivalente sem variáveis encadeadas e símbolos inúteis.

$$P \to A \,|\, BC$$
$$A \to B \,|\, C$$
$$C \to cC \,|\, c$$

3. Construa uma gramática equivalente sem regras recursivas à esquerda. O alfabeto para essa gramática é $\{0, 1, 2\}$.

$$T \to 0 \,|\, 1 \,|\, TT \,|\, 21T$$

13.6.2 Exercícios de Tutoria

1. Mostre que para toda GLC $G = (V, \Sigma, R, P)$, existe uma GLC equivalente em que toda regra são da seguinte forma:
   - $P \to \lambda$, se $\lambda \in L(G)$.
   - $X \to a$, para $X \in V$ e $a \in \Sigma$.
   - $X \to w$, para $X \in V$ e $w \in (V \cup \Sigma)^*$, $|w| \geq 2$.

14

Aula 14 - Formas normais de Chomsky e Greibach

Objetivos

- Apresentar a definição das formas normais de Chomsky e Greibach.
- Apresentar a construção de GLCs equivalentes nas formais normais a partir de uma GLC qualquer.

14.1 Forma normal de Chomsky

Definição 65 (Forma normal de Chomsky (FNC))**.** Seja uma GLC $G = (V, \Sigma, R, P)$. Dizemos que G está na forma normal de Chomsky (FNC) se todas as suas regras estão nas formas:

- $P \to \lambda$
- $X \to a$, $X \in V$ e $a \in \Sigma$
- $X \to YZ$, $X, Y, Z \in V$.

■

Lema 8. *Para qualquer GLC $G = (V, \Sigma, R, P)$ existe uma GLC equivalente em que todas as regras são das formas:*

- $P \to \lambda$, *se* $\lambda \in L(G)$.
- $X \to a$, *para* $X \in V$ *e* $a \in \Sigma$.
- $X \to w$, *para* $X \in V$, $|w| \geq 2$ *e* $w \in (V \cup \Sigma)^*$.

Demonstração. A GLC equivalente a G com essas propriedades é a resultante de

- eliminar variáveis anuláveis.
- eliminar variáveis encadeadas.
- eliminar símbolos inúteis.

da gramática $G_1 = (V \cup \{P_1\}, \Sigma, R \cup \{P_1 \to P\}, P_1)$. □

Teorema 19. *Seja uma GLC G. Existe uma GLC equivalente a G na FNC.*

Demonstração. Pelo Lema 8, temos uma GLC em que toda regra são das formas:

- $P \to \lambda$, se $\lambda \in L(G)$.
- $X \to a$, para $X \in V$ e $a \in \Sigma$.

- $X \to w$, para $X \in V$, $|w| \geq 2$ e $w \in (V \cup \Sigma)^*$.

Logo, para estar na FNC, basta lidar com as regras da forma $X \to w$, $|w| \geq 2$. Para isso:

1. Modificar cada regra $X \to w$, $|w| \geq 2$, de forma que ela contenha apenas variáveis. Para isso, substitua cada $a \in \Sigma$ em w por uma variável $Y \to a$, caso essa regra exista. Caso não haja tal regra na gramática, criar uma nova variável Z com a regra $Z \to a$ e substituir a por Z em w.

2. Substituir cada regra $X \to Y_1 Y_2 ... Y_n$, $n \geq 3$, em que cada Y_i é uma variável, pelas regras:

$$\begin{aligned} &X \to Y_1 Z_1 \\ &Z_1 \to Y_2 Z_2 \\ &\vdots \\ &Z_{n-2} \to Y_{n-1} Y_n \end{aligned}$$

   em que cada Z_i é uma nova variável.

□

Exemplo 74. Considere a seguinte GLC:

$$\begin{aligned} &L \to (S) \\ &S \to SE \mid \lambda \\ &E \to a \mid L \end{aligned}$$

Vamos mostrar o passo-a-passo da obtenção da GLC equivalente na FNC. Primeiro, vamos obter uma GLC com um novo símbolo de partida, conforme especificado pelo Lema 8.

$$\begin{aligned} &P \to L \\ &L \to (S) \\ &S \to SE \mid \lambda \\ &E \to a \mid L \end{aligned}$$

O próximo passo consiste em eliminar variáveis anuláveis. O conjunto de variáveis anuláveis de G é: $\mathcal{A}_G = \{S\}$. Eliminando as regras λ, temos:

$$\begin{aligned} &P \to L \\ &L \to (S) \mid () \\ &S \to SE \mid E \\ &E \to a \mid L \end{aligned}$$

Na sequência, devemos eliminar as variáveis encadeadas. O conjunto de variáveis encadeadas é apresentado a seguir.

$$\begin{aligned} &enc(L) = \{L\} \\ &enc(E) = \{E, L\} \\ &enc(S) = \{S, E, L\} \\ &enc(P) = \{P, L\} \end{aligned}$$

Eliminando as variáveis encadeadas, obtemos a seguinte gramática:

$$\begin{aligned} &P \to (S) \mid () \\ &L \to (S) \mid () \\ &S \to SE \mid a \mid (S) \mid () \\ &E \to a \mid (S) \mid () \end{aligned}$$

Finalmente, resta eliminar os símbolos inúteis. Note que a variável L não é alcançável a partir de P. Logo, todas as regras que mencionam L devem ser removidas da gramática. Com isso obtemos:

$$\begin{aligned} &P \to (S) \mid () \\ &S \to SE \mid a \mid (S) \mid () \\ &E \to a \mid (S) \mid () \end{aligned}$$

Que é a gramática resultante da aplicação da construção do Lema 8. Para obter a GLC na FNC, basta transformar todas as regras de tamanho maior ou igual a 2. Fazendo esse passo, obtemos:

$$
\begin{aligned}
&P \to AZ_1 \,|\, AF \\
&A \to (\\
&F \to) \\
&Z_1 \to SF \\
&S \to SE \,|\, a \,|\, AZ_1 \,|\, AF \\
&E \to a \,|\, AZ_1 \,|\, AF
\end{aligned}
$$

que é a GLC equivalente a gramática original na FNC. ■

14.2 Forma Normal de Greibach (FNG)

Definição 66 (Forma normal de Greibach (FNG))**.** Seja uma GLC $G = (V, \Sigma, R, P)$. Dizemos que G está na forma normal de Greibach (FNG) se todas as suas regras estão nas formas:

- $P \to \lambda$
- $X \to ay$, $X \in V, a \in \Sigma$ e $y \in V^*$

■

Lema 9. *Seja uma GLC $G = (V, \Sigma, R, P)$ em que $X \to uYv \in R$, $X, Y \in V$ e $X \neq Y$. Sejam $Y \to w_1 \,|\, ... \,|\, w_n$ todas as regras Y em G. Seja $G' = (V, \Sigma, R', P)$ em que:*

$$R' = \{R - \{X \to uYv\}\} \cup \{X \to uw_1v \,|\, ... \,|\, uw_nv\}$$

Temos que $L(G') = L(G)$.

Teorema 20. *Seja uma GLC G. Existe uma GLC equivalente a G na FNG.*

Demonstração. Primeiramente, usamos o Lema 8 e aplicamos o primeiro passo (converter $a \in \Sigma$ em variáveis) do Teorema 19. Com isso, teremos uma gramática em que toda regra possui a forma $X \to Yy$, em que $X, Y \in V$ e $y \in V^+$. Em seguida, para cada $X \in V$ faça enquanto possível:

1. Se existe uma regra $X \to Yy$, $|y| \geq 1$, aplica-se o Lema 9 para substituir Y (isso não se aplica a variáveis introduzidas por eliminação de recursão à esquerda).
2. Se existe uma regra $X \to Xy$, para $|y| \geq 1$, aplica-se a remoção de recursão à esquerda.
3. Se existir uma regra $X \to aw$ em que w possui símbolos do alfabeto, substituí-lo por uma variável.

□

Exemplo 75. Como um exemplo da construção de uma gramática equivalente na FNG, vamos considerar a gramática utilizada anteriormente para a FNC. Aqui começaremos o processo a partir da aplicação do Lema 8.

$$
\begin{aligned}
&P \to (S) \,|\, () \\
&S \to SE \,|\, a \,|\, (S) \,|\, () \\
&E \to a \,|\, (S) \,|\, ()
\end{aligned}
$$

Agora, vamos trocar todos os símbolos do alfabeto em questão por variáveis, em regras $X \to w$, $|w| \geq 2$. Isso resulta na seguinte gramática:

$$
\begin{aligned}
&P \to ASF \,|\, AF \\
&S \to SE \,|\, a \,|\, ASF \,|\, AF \\
&E \to a \,|\, ASF \,|\, AF \\
&A \to (\\
&F \to)
\end{aligned}
$$

Agora, vamos proceder à execução dos passos adicionais presentes no Teorema 20.

Inicialmente, vamos usar o Lema 9 para eliminar a regra $A \to (.$ Isso resulta na gramática:

$$\begin{aligned} P &\to (SF \,|\, (F \\ S &\to SE \,|\, a \,|\, (SF \,|\, (F \\ E &\to a \,|\, (SF \,|\, (F \\ F &\to) \end{aligned}$$

Observe que a única regra que não está de acordo com a FNG é $S \to SE$. Agora aplicamos a eliminação de recursão à esquerda. Com isso, obtemos:

$$\begin{aligned} P &\to (SF \,|\, (F \\ S &\to aZ_1 \,|\, (SFZ_1 \,|\, (FZ_1 \,|\, a \,|\, (SF \,|\, (F \\ E &\to a \,|\, (SF \,|\, (F \\ F &\to) \\ Z_1 &\to EZ_1 \,|\, E \end{aligned}$$

O próximo passo, é eliminar as regras E. Com isso, obtemos:

$$\begin{aligned} P &\to (SF \,|\, (F \\ S &\to aZ_1 \,|\, (SFZ_1 \,|\, (FZ_1 \,|\, a \,|\, (SF \,|\, (F \\ F &\to) \\ Z_1 &\to aZ_1 \,|\, (SFZ_1 \,|\, (FZ_1 \,|\, a \,|\, (SF \,|\, (F \end{aligned}$$

que é uma gramática equivalente na FNG. ■

14.3 Exercícios

14.3.1 Exercícios de Fixação

1. Construa uma gramática equivalente na FNC.

$$\begin{aligned} P &\to BPA \,|\, A \\ A &\to aA \,|\, \lambda \\ B &\to Bba \,|\, \lambda \end{aligned}$$

2. Construa uma gramática equivalente na FNG

$$\begin{aligned} P &\to A \,|\, BC \\ A &\to B \,|\, C \\ C &\to cC \,|\, c \end{aligned}$$

3. Construa uma gramática equivalente na FNC. O alfabeto para essa gramática é $\{0, 1, 2\}$.

$$T \to 0 \,|\, 1 \,|\, T\,T \,|\, 2\,1\,T$$

14.3.2 Exercícios de Tutoria

1. Mostre como construir uma gramática na FNC a partir de uma gramática na FNG.

15

Aula 15 - Equivalência entre autômatos de pilha e gramáticas livres de contexto

Objetivos

- Apresentar a construção de um APN a partir de uma gramática.
- Apresentar a construção de uma gramática a partir de um APN.

15.1 Equivalência entre GLCs e APs

15.1.1 Construindo um AP a partir de uma GLC

Teorema 21. *Para qualquer GLC G, existe um AP M que aceita $L(G)$.*

Demonstração. Seja $G' = (V, \Sigma, R, P)$ a gramática equivalente a G na FNG. Um APN que aceita $L(G')$ é: $M = (\{i, f\}, \Sigma, V, \delta, \{i\}, \{f\})$, em que δ é tal que:

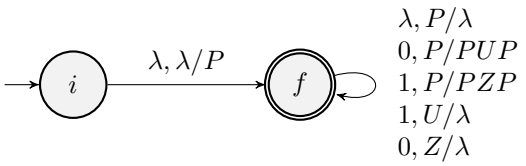
- $\delta(i, \lambda, \lambda) = \{[f, P]\}$
- Se $P \to \lambda \in R$, $\delta(f, \lambda, P) = \{[f, \lambda]\}$
- $\delta(f, a, X) = \{[f, y] \mid y \in V^* \wedge X \to ay \in R\}$.

□

Exemplo 76. Para ilustrar a construção de um AP a partir de uma gramática, vamos considerar a seguinte gramática na FNG:

$$P \to 0PUP \mid 1PZP \mid \lambda$$
$$U \to 1$$
$$Z \to 0$$

O AP que aceita $L(G)$ é apresentado abaixo:



■

15.1.2 Construindo uma GLC a partir de um AP

Seja $M = (E, \Sigma, \Gamma, I, F)$ um APN qualquer, $e, e' \in E$ e $X \in \Gamma \cup \{\lambda\}$. Denote por $C(e, X, e')$ o conjunto de palavras $w \in \Sigma^*$ em que M começando em e, com pilha contendo X termina em e' com pilha vazia, isto é:

$$C(e, X, e') = \{w \in \Sigma^* \mid [e, w, X] \vdash^* [e', \lambda, \lambda]\}$$

Note que $L(M) = \bigcup_{(i,f) \in I \times F} C(i, \lambda, f)$.

Suponha que seja possível construir uma GLC que gere todas as palavras em $C(e, X, e')$ usando, como símbolo de partida, $[e, X, e']$. Dessa forma, o problema de obter uma GLC equivalente a um AP consiste em gerar regras para a variável $[i, \lambda, f]$, para cada $i \in I, f \in F$. O próximo teorema dá os detalhes dessa construção.

Teorema 22. *Para cada AP M, existe uma GLC que gera $L(M)$.*

Demonstração. Seja $M = (E, \Sigma, \Gamma, I, F)$. A gramática $G = (V, \Sigma, R, P)$ tal que $L(G) = L(M)$ é:

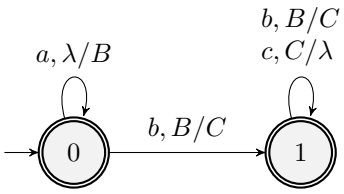
- $V = \{P\} \cup E \times (\Gamma \cup \{\lambda\}) \times E$.
- O conjunto de regras R é formado por:
  - Para cada $i \in I$ e $f \in F$, $P \to [i, \lambda, f]$;
  - Para cada $e \in E$, $[e, \lambda, e] \to \lambda$;
  - Para cada transição $[e', z] \in \delta(e, a, A)$, em que $a \in \Sigma \cup \{\lambda\}$, $z \in \Gamma^*$ e $A \in \Gamma \cup \{\lambda\}$, temos as seguintes regras:
    - Se $z = \lambda$, $[e, A, d] \to a[e', \lambda, d]$ para cada $d \in E$.
    - Se $z \neq \lambda$, $[e, A, d] \to a[e', B_1, d_1]...[d_{n-1}, B_n, d_n]$, em que $z = B_1...B_n$ e $(d_1, ..., d_n) \in E^n$.
    - Se $A = \lambda$, adicionamente temos as seguintes regras:
      - Se $z = \lambda$, $[e, C, d] \to a[e', C, d]$, para cada $C \in \Gamma$ e cada $d \in E$.
      - Se $z \neq \lambda$, $[e, C, d] \to a[e', B_1, d_1]...[d_{n-1}, B_n, d_n][d_n, C, d_{n+1}]$ para cada $C \in \Gamma$ e cada $(d_1, ..., d_{n+1}) \in E^{n+1}$.

□

A construção de um GLC usando o teorema anterior irá produzir muitas variáveis inúteis. Podemos minimizar esse efeito seguindo as seguintes recomendações:

1. Começando com as regras de P, gerar apenas variáveis alcançáveis por P.
2. Se não há caminho de e para e' no AP M, variáveis $[e, X, e']$ são inúteis.
3. Se toda computação de e para e' só aumenta a pilha, então variáveis $[e, X, e']$ são inúteis.

Exemplo 77. Para ilustrar a construção de uma gramática equivalente a um APN, considere o seguinte AP:



Seguinte a construção especificada no teorema 22, obtemos as seguintes regras:

$$P \to [0, \lambda, 0] \mid [0, \lambda, 1]$$
$$[0, \lambda, 0] \to \lambda$$
$$[1, \lambda, 1] \to \lambda$$

A partir da transição, $\delta(0, a, \lambda) = [0, B]$, temos:

$$[0, \lambda, 0] \to a[0, B, 0]$$
$$[0, \lambda, 1] \to a[0, B, 1]$$
$$[0, B, 1] \to a[0, B, 0][0, B, 1] \,|\, a[0, B, 1][1, B, 1]$$

A partir da transição $\delta(0, b, B) = [1, C]$, obtemos:

$$[0, B, 1] \to b[1, \lambda, 1]$$

Finalmente, a transição $\delta(1, b, B) = [1, C]$ gera as seguintes regras:

$$[1, B, 1] \to b[1, \lambda, 1]$$
$$[1, C, 1] \to c[1, \lambda, 1]$$

■

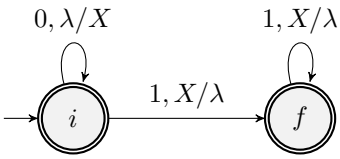
15.2 Exercícios

15.2.1 Exercícios de Fixação

1. Construa um APN equivalente a gramática a seguir.

$$P \to BPA \,|\, A$$
$$A \to aA \,|\, \lambda$$
$$B \to Bba \,|\, \lambda$$

2. Construa a gramática equivalente ao seguinte AP:



16

Aula 16 - Lema do bombeamento para linguagens livres de contexto

Objetivos

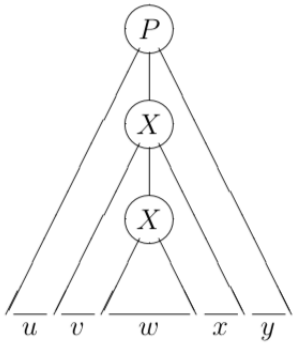
- Apresentar e provar o lema do bombeamento para linguagens livres de contexto.
- Exemplos de uso do lema para mostrar que uma linguagem não é livre de contexto.

16.1 O lema do bombeamento

Lema 10. *Seja L uma linguagem livre de contexto. Então, existe uma constante $k > 0$ tal que para qualquer palavra $z \in L$ com $|z| \geq k$ existem $u, v, w, x, y \in \Sigma^*$ que satisfazem as seguintes condições:*

- $z = uvwxy$
- $|vwx| \leq k$
- $vx \neq \lambda$
- $\forall i.i \geq 0 \rightarrow uv^iwx^iy \in L$

Demonstração. Suponha $G = (V, \Sigma, R, P)$ uma GLC na FNC. Seja $k = 2^{|V|+1}$. Suponha $z \in L(G)$ e que $|z| \geq k$. Como toda AD de uma GLC na FNC é binária, temos que a AD para z tem altura não inferior a $|V| + 1$. Considere o maior caminho da raiz a uma das folhas nesta AD. Pelo princípio da casa dos pombos, algum terminal ocorre mais de uma vez neste caminho. Escolha a variável que repete pela primeira vez neste caminho iniciando das folhas para a raiz. A partir do formato da árvore de derivação (apresentado na figura a seguir) temos que $z = uvwxy$.



Como selecionamos a variável X a como sendo a primeira que repete no maior caminho iniciando das folhas para a raiz, temos que $|vwx| \leq n + 1 = k$. Uma vez que G está na FNC, não pode ser o caso de $vx = \lambda$ pois, nesta situação, teríamos que a variável X seria encadeada, o que não ocorre na FNC. Evidentemente, podemos repetir derivações iniciando com X um número $i \geq 0$, produzindo $uv^iwx^iy \in L$. □

Exemplo 78. Vamos usar o lema do bombeamento para mostrar que a linguagem $\{0^n1^n2^n \mid n \geq 0\}$ não é livre de contexto. A estratégia é similar ao lema do bombeamento para linguagens regulares.

Suponha que $L = \{0^n1^n2^n \mid n \geq 0\}$ é uma LLC e seja k a constante referida no lema. Seja $z = 0^k1^k2^k$. Como $|z| \geq k$, pelo LB as seguintes condições devem ser verdadeiras:

- $z = uvwxy$
- $|vwx| \leq k$
- $vx \neq \lambda$
- $\forall i.i \geq 0 \rightarrow uv^iwx^iy \in L$

Suponha que $0^k1^k2^k = uvwxy$, $|vwx| \leq k$ e que $vx \neq \lambda$. Considere os seguintes casos:

- vx contém algum 0. Nessa situação, temos que vx não possui nenhum 2. Seja $i = 0$. Temos que $uv^0wx^0y \notin L$ pois terá menos 0's que 1's ou 2's.
- vx não contém 0. Nesse caso, vx possui apenas 1's e 2's. Seja $i = 0$. Temos que $uv^0wx^0y \notin L$ pois terá mais 0's que 1's ou 2's.

Logo, em ambos os casos, $uv^0wx^0y \notin L$, contrariando o LB. Portanto, $L = \{0^n1^n2^n \mid n \geq 0\}$ não pode ser uma LLC. ■

16.2 Exercícios

16.2.1 Exercícios de Fixação

1. Prove que as seguintes linguagens não são livres de contexto usando o lema do bombeamento.
   (a) $\{ww \mid w \in \{0,1\}^*\}$.
   (b) $\{0^{n^2} \mid n \geq 0\}$.
   (c) $\{a^nb^kc^nd^k \mid n,k \geq 0\}$.
   (d) $\{a^nb^{2n}c^n \mid n \geq 0\}$.

17

Aula 17 - Propriedades de fechamento e problemas de decisão para linguagens livres de contexto

Objetivos

- Demonstrar que as linguagens livres de contexto são fechadas sobre união, fecho de Kleene, concatenação e interseção com linguagens regulares.
- Apresentar alguns problemas de decisão para linguagens livres de contexto.
- Aplicar propriedades de fechamento para construção de gramáticas e demonstração de que linguagens não são livres de contexto.

17.1 Propriedades de fechamento para LLCs

Teorema 23. *A classe das linguagens livres de contexto é fechada sobre união, fecho de Kleene e concatenação.*

Demonstração. Sejam $G_1 = (V_1, \Sigma, R_1, P_1)$ e $G_2 = (V_2, \Sigma, R_2, P_2)$ duas GLCs.

- A GLC $G_3 = (V_1 \cup V_2 \cup \{P_3\}, \Sigma, R_1 \cup R_2 \cup \{P_3 \to P_1 \mid P_2\})$ gera $L(G_1) \cup L(G_2)$.
- A GLC $G_3 = (V_1 \cup V_2 \cup \{P_3\}, \Sigma, R_1 \cup R_2 \cup \{P_3 \to P_1\, P_2\})$ gera $L(G_1)\, L(G_2)$.
- A GLC $G_3 = (V_1 \cup \{P_3\}, \Sigma, R_1 \cup \{P_3 \to P_1\, P_3 \mid \lambda\})$ gera $L(G_1)^*$.

□

Exemplo 79. Vamos considerar a linguagem $\{0^k 1^k 2^n \mid n, k \geq 0\}$. Vamos usar propriedades de fechamento para construir uma GLC para essa linguagem. Primeiramente, considere a seguinte GLC para $\{0^k 1^k \mid k \geq 0\}$:

$$P_1 \to 0P_1 1 \mid \lambda$$

A seguinte gramática gera $\{2^n \mid n \geq 0\}$:

$$P_2 \to 2P_2 \mid \lambda$$

Note que $\{0^k 1^k 2^n \mid n, k \geq 0\}$ é a concatenação de $\{0^k 1^k \mid k \geq 0\}$ com a linguagem $\{2^n \mid n \geq 0\}$. Logo, Usando as propriedades de fechamento, a gramática para $\{0^k 1^k 2^n \mid n, k \geq 0\}$ é:

$$\begin{aligned} P_3 &\to P_1\, P_2 \\ P_1 &\to 0P_1 1 \mid \lambda \\ P_2 &\to 2P_2 \mid \lambda \end{aligned}$$

■

Teorema 24. *A classe das linguagens livres de contexto não é fechada sobre a complementação e interseção.*

Demonstração. Sejam $L_1 = \{0^n 1^n 2^k \mid n, k \geq 0\}$ e $L_2 = \{0^k 1^n 2^n \mid n, k \geq 0\}$ duas LLCs. Temos que $L_1 \cap L_2 = \{0^n 1^n 2^n \mid n \geq 0\}$ que não é uma LLC. Logo, as LLCs não são fechadas sobre a interseção. Da teoria de conjunto sabemos que $A \cap B = \overline{\overline{A} \cup \overline{B}}$. Como as LLCs são fechadas sobre a união, se as LLCs fossem fechadas sobre a complementação elas também seriam sobre a interseção. Logo, as LLCs não são fechadas sobre a complementação. □

Teorema 25. *Sejam L uma linguagem livre de contexto e R uma linguagem regular. Então, $L \cap R$ é uma linguagem livre de contexto.*

Demonstração. Sejam $M_1 = (E_1, \Sigma, \Gamma, \delta_1, I_1, F_1)$ um APN para L e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$ um AFD para R. Um APN para $L \cap R$ é $M_3 = (E_3, \Sigma, \Gamma, \delta_2, I_3, F_3)$, em que:

- $E_3 = E_1 \times E_2$
- $I_3 = I_1 \times \{i_2\}$
- $F_3 = F_1 \times F_2$
- para cada par de transições $[e'_1, z] \in \delta_1(e_1, a, A)$ e $\delta_2(e_2, a) = e'_2$, em que $e_1, e'_1 \in E_1$, $e_2, e'_2 \in E_2$ e $a \in \Gamma \cup \{\lambda\}$ e $z \in \Gamma^*$, há uma transição $[(e'_1, e'_2), z] \in \delta_3((e_1, e_2), a, A)$ e para cada transição λ $[e'_1, z] \in \delta_1(e_1, \lambda, A)$ há transições $[(e'_1, e_2), z] \in \delta_3((e_1, e_2), \lambda, A)$ para cada $e_2 \in E_2$.

□

17.2 Problemas de decisão para LLCs

O seguintes problemas são decidíveis para LLCs

- Dados uma palavra w e uma LLC L, determinar se $w \in L$?
- Determinar se a linguagem gerada por uma gramática é vazia.

O primeiro consiste em determinar se $w \in L(G)$, em que G é uma GLC para L e o segundo pode ser resolvido verificando se o símbolo de partida da gramática em questão é inútil.

Diversos problemas envolvendo GLCs são indecidíveis. Esse assunto será estudado posteriormente na disciplina.

17.3 Exercícios

17.3.1 Exercícios de Fixação

1. Construa gramáticas para as seguintes linguagens.
   (a) $L_1 = \{a^m b^n \mid n \neq m\}$.
   (b) $L_2 = \{a^{m+n} b^m c^n \mid m \text{ é par e } n \text{ é ímpar}\}$.
   (c) $L_1 L_2 \cup L_1^*$.

Parte III

Linguagens sensíveis ao contexto, recursivas e recursivamente enumeráveis. Máquinas de Turing e Decidibilidade

18

Aula 18 - Introdução às Máquinas de Turing

Objetivos

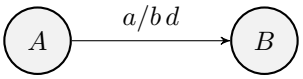
- Apresentar o conceito de máquina de Turing.
- Apresentar os conceitos de linguagens recursivas e recursivamente enumeráveis.
- Apresentar os conceitos de aceitação por parada em estado final e por parada.

18.1 Introdução

Conforme apresentado em aulas passadas, linguagens regulares e livres de contexto não são capazes de representar a estrutura de linguagens muito simples, como por exemplo, $\{0^n1^n2^n \mid n \geq 0\}$. Nesta aula apresentaremos as máquinas de Turing (MT), um modelo matemático de máquina proposta por Alan Turing na década de 30. As MTs são tão expressivas, que até hoje não se conhece um formalismo mais poderoso.

Ao contrário das máquinas que vimos até o presente momento, a MT permite sobreescrever a palavra de entrada. Esse fato aliado a uma memória infinita proporciona maior expressividade que os autômatos de pilha. Em MTs consideramos que o próximo símbolo a ser processado é o que está sobre o *cabeçote*, que pode mover para a direita ou esquerda durante transições. O conteúdo da fita de uma MT é da forma $\langle w\sqcup^*$, em que $\langle$ é o marcador de início de fita e $\sqcup$ é o símbolo de "branco", que preenche todas as posições da fita de uma MT após o fim da palavra de entrada. Note que se a MT inicia com a palavra λ ela será da forma $\langle\sqcup^*$.

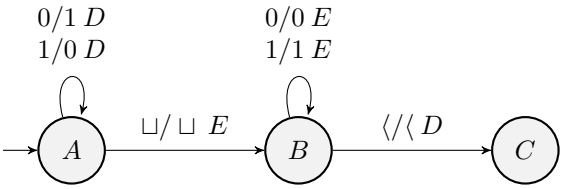
Abaixo, apresentamos o formato geral de uma transição em uma máquina de Turing:



A transição acima pode ser descrita formalmente como $\delta(A, a) = [B, b, d]$, em que $d \in \{D, E\}$ e, pode ser entendida como: a máquina estando no estado A com símbolo sobre o cabeçote a, escreve b na posição atual e move o cabeçote na direção d.

A MT utiliza uma fita, infinita à direita, possuindo células capazes de armazenar um símbolo de alfabeto cada. No próximo exemplo, apresentaremos uma MT simples e ilustraremos como a computação funciona nesse tipo de máquina.

Exemplo 80. Considere o seguinte diagrama de estados para uma MT que calcula a operação de not bit-a-bit.

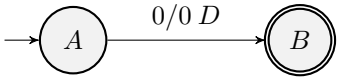


Para exemplificar o funcionamento desta MT, vamos utilizar a noção de configuração instântanea, que consiste de um par formado pelo estado atual e pelo conteúdo da fita. Vamos mostrar o processamento desta MT sobre a palavra 0011:

$$
\begin{array}{lcl}
[A, \langle \underline{0}011] & \vdash & \{\delta(A,0) = [A,1,D]\} \\
[A, \langle 1\underline{0}11] & \vdash & \{\delta(A,0) = [A,1,D]\} \\
[A, \langle 11\underline{1}1] & \vdash & \{\delta(A,1) = [A,0,D]\} \\
[A, \langle 110\underline{1}] & \vdash & \{\delta(A,1) = [A,0,D]\} \\
[A, \langle 1100\underline{\sqcup}] & \vdash & \{\delta(A,\sqcup) = [B,\sqcup,E]\} \\
[B, \langle 110\underline{0}] & \vdash & \{\delta(B,0) = [B,0,E]\} \\
[B, \langle 11\underline{0}0] & \vdash & \{\delta(B,0) = [B,0,E]\} \\
[B, \langle 1\underline{1}00] & \vdash & \{\delta(B,1) = [B,1,E]\} \\
[B, \langle \underline{1}100] & \vdash & \{\delta(B,1) = [B,1,E]\} \\
[B, \langle\langle 1100] & \vdash & \{\delta(B,\langle) = [C,\langle,D]\} \\
[C, \langle \underline{1}100] & &
\end{array}
$$

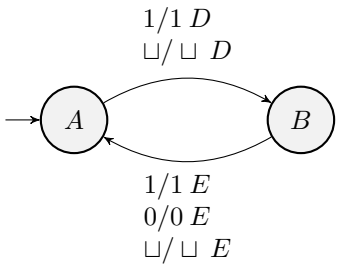
Com isso, a MT pára contendo 1100 que é o resultado do not bit-a-bit de 0011, que era a palavra original de entrada. Note que, como o objetivo desta MT não é a aceitação de uma palavra, não faz sentido possuir estados finais. O próximo exemplo ilustra uma MT que aceita uma linguagem. ■

Exemplo 81. A seguinte MT aceita a linguagem regular $0(0+1)^*$:



Essa MT apresenta uma característica importante: uma MT não precisa processar uma palavra inteira para aceitar uma palavra. Note que se a palavra não inicia com o símbolo 0, a MT permanece no estado inicial A, que é não final. Caso inicie com 0, a MT faz a transição para o estado B e pára (pois este não possui transições) em estado final. Dizemos que uma MT aceita uma palavra por parada em estado final se o processamento de uma palavra faz com que a MT pare em um estado final.

Considere, agora outra MT que aceita a mesma linguagem usando outro critério de aceitação:



Observe que essa MT entra em loop para qualquer palavra que não inicia com 0. Dizemos que uma MT aceita uma palavra por parada se o processamento da MT pára em algum estado e a recusa se a MT entra em loop. Logo, a MT anterior, aceita $0(0+1)^*$ por parada. ■

18.2 Máquinas de Turing

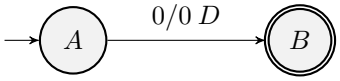
Segue a definição formal de uma MT.

Definição 67 (Máquina de Turing)**.** Uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ é uma óctupla, em que:

- E: conjunto de estados.
- Σ: alfabeto de entrada.
- Γ: alfabeto de fita.
- $\langle \in \Gamma - (\{\sqcup\} \cup \Sigma)$: marcador de início de fita.
- $\sqcup \in \Gamma - (\{\langle\} \cup \Sigma)$: símbolo de branco.
- $\delta : E \times \Gamma \to E \times \Gamma \times \{E, D\}$, função de transição. Uma função parcial.
- $i \in E$: estado inicial.
- $F \subseteq E$: conjunto de estados finais.

Uma MT que segue essa definição é dita ser uma MT padrão. ■

Exemplo 82. Para ilustrar a definição acima, vamos considerar novamente a MT que aceita $0(0+1)^*$:



Temos que:

- $E = \{A, B\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{\langle, \sqcup, 0, 1\}$
- A função de transição é formada por uma única equação $\delta(A, 0) = [B, 0, D]$.
- $i = A$
- $F = \{B\}$

■

18.3 Linguagem aceita por MT padrão

Para definir a linguagem aceita por uma MT, é necessário uma função que elimina os símbolos de branco presentes à direita da palavra de entrada na fita. A função $\pi : \Gamma^* \to \Gamma^*$ é definida como:

$$\pi(w) = \begin{cases} \lambda & se\ w \in \{\sqcup\}^* \\ xa & se\ w = xay \wedge a \neq \sqcup \wedge y \in \{\sqcup\}^* \end{cases}$$

Usando a função π, podemos definir a relação de transição entre configurações instantâneas de uma MT.

Definição 68 (Transição entre configurações intantâneas)**.** Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ uma MT. A relação $\vdash \subseteq (E \times \Gamma^+)^2$, para M, é tal que para todo $e \in E$ e $a \in \Gamma$:

1. Se $\delta(e, a) = [e', b, D]$ então $[e, x\underline{a}cy] \vdash [e', xb\underline{c}y]$ para $c \in \Gamma$ e, $[e, x\underline{a}] \vdash [e', xb\underline{\sqcup}]$;
2. Se $\delta(e, a) = [e', b, E]$ então $[e, xc\underline{a}y] \vdash [e', x\underline{c}\pi(by)]$ para $c \in \Gamma$;
3. Se $\delta(e, a)$ é indefinido então não existe f tal eque $[e, x\underline{a}y] \vdash f$.

Como usual, $\vdash^*$ denotará o fecho reflexivo e transitivo de $\vdash$. ■

Definição 69 (Linguagem aceita por parada em estado final)**.** Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ um MT. A linguagem aceita por M por parada em estado final é:

$$L(M) = \{w \in \Sigma^* \mid [i, \langle w] \vdash^* [e', x\underline{a}y], \delta(e, a) = \bot \wedge e' \in F\}$$

A notação $\delta(e, a) = \bot$ representa que a transição $\delta(e, a)$ é indefinida, isto é, que a MT pára. ■

Definição 70 (Linguagem recursivamente enumerável)**.** Dizemos que $L \subseteq \Sigma^*$ é uma linguagem recursivamente enumerável (LRE) se existe uma MT que a reconhece. ■

Definição 71 (Linguagem recursiva)**.** Dizemos que $L \subseteq \Sigma^*$ é uma linguagem recursiva (LRec) se existe uma MT que a reconhece e pare para todas as palavras do alfabeto de entrada. ■

Definição 72 (Linguagem aceita por estado final)**.** Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ um MT. A linguagem aceita por M por estado final é:

$$L_F(M) = \{w \in \Sigma^* \mid [i, \langle w] \vdash^* [e', x\underline{a}y] \wedge e' \in F\}$$

Note que nesse critério de reconhecimento basta existir um único estado final e este não deve possuir transições. ■

Definição 73 (Linguagem aceita por parada)**.** Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i)$ um MT. A linguagem aceita por M por parada é:

$$L_P(M) = \{w \in \Sigma^* \mid [i, \langle w] \vdash^* [e', x\underline{a}y], \delta(e, a) = \bot\}$$

■

Todos esses 3 critérios de aceitação são equivalentes entre si, conforme mostrado pelo teorema a seguir.

Teorema 26. *Seja L uma linguagem. As seguintes afirmativas são equivalentes:*

1. *L é uma linguagem recursivamente enumerável.*
2. *L pode ser reconhecida por uma MT que aceita por estado final.*
3. *L pode ser reconhecida por uma MT que aceita por parada.*

Demonstração.
$1 \to 2$): Suponha que L é uma LRE. Se L é uma LRE, existe uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ tal que $L(M) = L$. Seja $f \notin E$ um novo estado. A MT $M' = (E \cup \{f\}, \Sigma, \Gamma, \langle, \sqcup, \delta', i, \{f\})$, em que δ' é definida como:

- Se $\delta(e, a)$ é definido, então $\delta'(e, a) = \delta(e, a)$.
- Para todo $(e, a) \in F \times \Gamma$, se $\delta(e, a) = \bot$, então $\delta'(e, a) = [f, a, D]$.
- Para todo $(e, a) \in (E - F) \times \Gamma$, se $\delta(e, a) = \bot$, então $\delta'(e, a) = \bot$.
- Para todo $a \in \Gamma$, $\delta'(f, a) = \bot$.

$2 \to 3$): Suponha que L pode ser reconhecida por uma MT que aceita por estado final e seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ tal MT. A MT $M' = (E \cup \{l\}, \Sigma, \Gamma, \langle, \sqcup, \delta', i)$, $l \notin E$ é equivalente a M, e aceita por parada, em que δ' é definida como:

- para todo $(e, a) \in (E - F) \times \Gamma$, se $\delta(e, a)$ é definido, temos que $\delta'(e, a) = \delta(e, a)$. Se $\delta(e, a) = \bot$, então $\delta'(e, a) = [l, a, D]$.
- Para todo $a \in \Gamma$, $\delta'(l, a) = [l, a, D]$.
- Para todo $(e, a) \in F \times \Gamma$, $\delta'(e, a) = \bot$.

$3 \to 1$): Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i)$ uma MT que aceita L por parada. A MT $M' = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, E)$ aceita L por parada em estado final. □

18.4 Exercícios

18.4.1 Exercícios de Fixação

1. Construa uma MT padrão que aceita a linguagem $\{0^{2n} \mid n \geq 0\}$.
2. Construa uma MT padrão que aceita a linguagem $\{a^n b^n \mid n \geq 0\}$.
3. Utilize o Teorema 1 para construir MTs que aceitem por parada as linguagens dos exercícios 1 e 2.

19

Aula 19 - Variantes de Máquinas de Turing

Objetivos

- Apresentar algumas extensões ao formalismo de máquinas de Turing.
- Apresentar, informalmente, a equivalência entre as extensões e a MT padrão.

19.1 MT com cabeçote imóvel

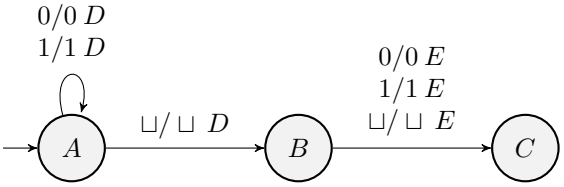
Essa é a mais simples das variações e consiste em permitir que o cabeçote fique imóvel durante uma transição ao invés de obrigatoriamente se deslocar para esquerda ou direita.

Definição 74 (MT com cabeçote imóvel)**.** Uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ é uma MT com cabeçote imóvel, em que $E, \Sigma, \Gamma, \langle, \sqcup, i$ e F são como em MTs padrão. A única diferença está na função de transição:

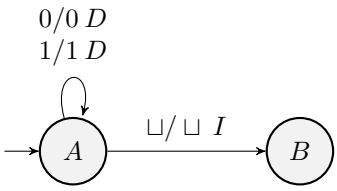
$$\delta : E \times \Gamma \to E \times \Gamma \times \{E, D, I\}$$

que pode usar o movimento de cabeçote I que indica que o mesmo deve ficar imóvel nesta transição. ■

Exemplo 83. Considere a tarefa de elaborar uma MT que para o cabeçote na primeira posição após o final da palavra de entrada. Uma MT padrão para essa tarefa é apresentada abaixo:

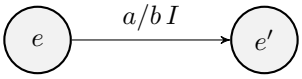


Usando a extensão de cabeçote imóvel, essa mesma máquina pode ser expressa de maneira mais concisa.

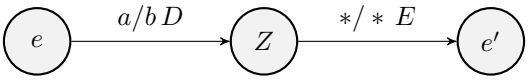


■

É fácil perceber que para uma MT com cabeçote imóvel, existe uma padrão equivalente. Note que a seguinte transição entre estados e, e':



é equivalente a:



em que $*$ denota uma transição para cada um dos símbolos $a \in \Gamma$.

19.2 MT com múltiplas trilhas

Em uma MT com múltiplas trilhas cada célula da fita armazena uma k-upla, $k \geq 1$, de símbolos. Assume-se que o primeiro símbolo da trilha 1 é $\langle$ e palavra de entrada encontra-se na trilha 1, a partir da segunda posição. O restante da trilha 1 e as demais trilhas são preenchidas com o símbolo $\sqcup$.

Definição 75 (MT com múltiplas trilhas)**.** Uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ é uma MT com múltiplas trilhas, em que $E, \Sigma, \Gamma, \langle, \sqcup, i$ e F são como em MTs padrão. A única diferença está na função de transição:

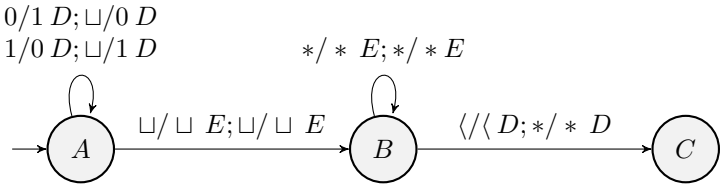
$$\delta : E \times \Gamma^k \to E \times \Gamma^k \times \{E, D\}$$

em que $k \geq 1$ é o número trilhas utilizadas na máquina. ■

Note que se M é uma máquina de k trilhas, então as transições de M são da forma $\delta(e, a_1, a_2, ..., a_k) = [e', b_1, b_2, ..., b_k, d]$ indicando que cada a_i deve substituir um b_i.

A configuração instantânea de uma MT com múltiplas trilhas tem a forma $[e, x_1\underline{a_1}, y_1, x_k\underline{a_k}, y_k]$, em que $|x_i| = |x_j|$ para $i \neq j$. A seguir apresentamos um exemplo simples de MT com 2 trilhas.

Exemplo 84. A seguir apresentamos uma MT de 2 trilhas que calcula o not bit-a-bit da palavra presente na trilha 1 e copia a palavra original para a trilha 2.



■

Pode-se mostrar que a MT com múltiplas trilhas é equivalente a uma MT padrão da seguinte maneira: Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ uma MT com k trilhas. A MT padrão $M' = (E, \Sigma \times \{\sqcup\}^{k-1}, \Gamma^k, \langle, \sqcup, \delta', i, F)$ em que se $\delta(e, a_1, ..., a_k) = [e', b_1, ..., b_k, d]$ então $\delta'(e, [a_1, ..., a_k]) = (e', [b_1, ..., b_k], d)$.

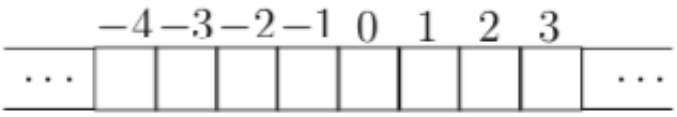
Intuitivamente, a equivalência é baseada no fato de que uma MT de k-trilhas é uma MT padrão em que cada símbolo do alfabeto foi subdividido em k componentes, ao invés de serem considerados indivisíveis.

19.3 MT com fita ilimitada em ambas as direções

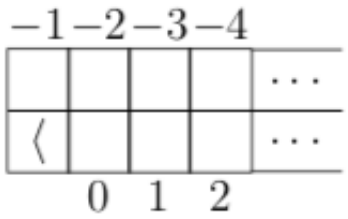
Uma MT com fita ilimitada em ambas as direções difere da MT padrão por possuir um número infinito de células tanto à esquerda quanto à direita. Nesse tipo de MT não há necessidade de um marcador de início de fita. Inicialmente, o cabeçote desta MT inicia no primeiro símbolo da palavra de entrada. Todas as células não ocupadas pela palavra de entrada nesta MT são preenchidas como o símbolo $\sqcup$.

Mostraremos como uma MT com fita ilimitada em ambas as direções é equivalente a uma MT padrão, transformando a MT com fita ilimitada em uma MT com 2 trilhas. Como MTs com k trilhas são equivalentes a MT padrão, a equivalência desejada é imediata.

Seja $M = (E, \Sigma, \Gamma, \sqcup, \delta, i, F)$ uma MT com fita ilimitada e $\langle \notin \Gamma$. A MT de duas trilhas $M' = (E', \Sigma, \Gamma', \langle, \sqcup, \delta', i', F')$ simula M da seguinte forma. A primeira trilha conterá o símbolo de início de fita, $\langle$, a palavra de entrada e todo o conteúdo à direita da fita ilimitada. Por sua vez, a trilha 2 conterá todo o conteúdo à esquerda da palavra de entrada. Para ilustrar essa idéia, considere a seguinte fita infinita contendo índices para as posições de suas células.



A idéia é representar as posições $k \geq 0$ na trilha 1 e as posições menores que zero na trilha 2:



O conjunto de estados de M' é $E' = E \times \{1, 2\}$, que consiste de pares formados pelos estados de M e de qual trilha (lado da fita infinita) está o cabeçote. O alfabeto de fita é $\Gamma' = \Gamma \cup \{\langle\}$, $i' = (i, 1)$ e $F' = F \times \{1, 2\}$. A função de transição δ' é obtida da seguinte forma.

- Para cada transição $\delta(e, a) = [e', b, D]$, deve-se ter:
  - $\delta'([e, 1], a, c) = [[e', 1], b, c, D]$, para cada $c \in \Gamma$;
  - $\delta'([e, 2], c, a) = [[e', 2], c, b, D]$, para cada $c \in \Gamma$;
  - $\delta'([e, 1], \langle, a) = \delta'([e, 2], \langle, a) = [[e', 1], \langle, b, D]$.
- Para cada transição $\delta(e, a) = [e', b, E]$, deve-se ter:
  - $\delta'([e, 1], a, c) = [[e', 1], b, c, E]$, para cada $c \in \Gamma$;
  - $\delta'([e, 2], c, a) = [[e', 2], c, b, D]$, para cada $c \in \Gamma$;
  - $\delta'([e, 1], \langle, a) = \delta'([e, 2], \langle, a) = [[e', 2], \langle, b, D]$.

19.4 MT com múltiplas fitas

Em uma MT com múltiplas fitas, cada uma das $k \geq 1$ fitas possui seu próprio cabeçote de leitura / escrita que podem atuar de maneira independente dos demais. A palavra de entrada encontra-se na fita 1 e as demais fitas são formadas por $\langle \sqcup^*$.

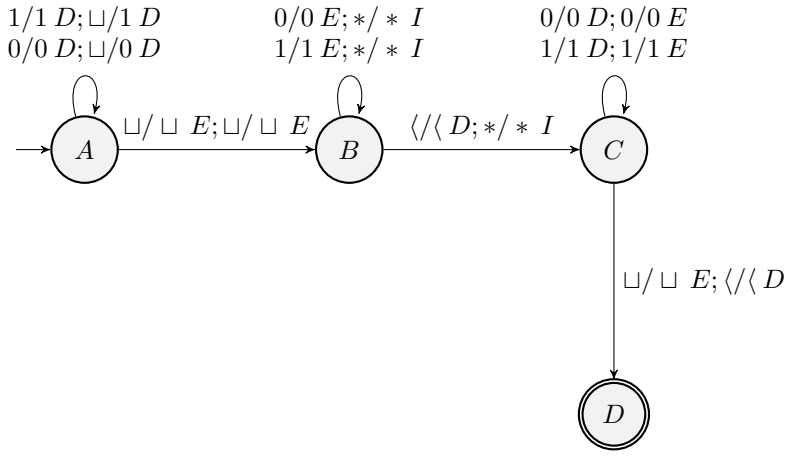
Definição 76 (MT com múltiplas fitas)**.** Uma MT com k-fitas $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ possui $E, \Sigma, \Gamma, \langle, \sqcup, i$ e F como MTs padrão e a função de transição $\delta : E \times \Gamma^k \to E \times (\Gamma \times \{E, D, I\})^k$. ■

O uso de várias fitas pode simplificar muito o projeto de MTs. O seguinte exemplo ilustra esse fato.

Exemplo 85. A seguinte MT de duas fitas reconhece palíndromos usando a seguinte estratégia:

- Copia a palavra de entrada para a fita 2.

- Volta o cabeçote da fita 1 para o início enquanto o da fita 2 fica no último símbolo da palavra de entrada.
- Inicia a comparação, símbolo a símbolo, dos símbolos sobre o cabeçote de cada fita.



■

Para mostrar a equivalência de MTs com várias fitas e a MT padrão, mostremos como uma MT de 2-fitas pode ser simulada por uma MT de 4 trilhas. A idéia é usar a trilha 1 para armazenar o conteúdo da fita 1 e a trilha 2 armazena um símbolo $X \notin \Gamma$ que indica a posição atual do cabeçote na fita 1. O mesmo raciocínio aplica-se a fita 2 e trilhas 3 e 4. Após a escrita da posição inicial dos cabeçotes das fitas 1 e 2 nas trilhas 2 e 4, escrever $\langle$ no início da trilha 3. Transições da nova MT simulam as transições realizadas pela MT de duas fitas e atualizam as trilhas 2 e 4 para refletir a movimentação do cabeçote.

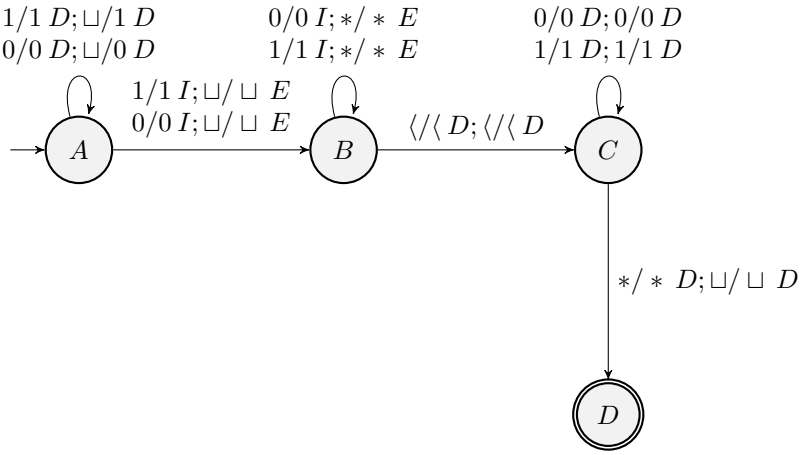
19.5 MT não determinísticas

Um MT não determinística é uma MT que admite mais de uma transição partindo de um certo estado sobre um certo símbolo. Em uma MT não determinística, a aceitação de uma palavra é determinada pela existência de uma computação que pára em estado final.

Definição 77 (MT não determinística)**.** Uma MT não determinística $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ possui $E, \Sigma, \Gamma, \langle, \sqcup, i$ e F como MTs padrão e a função de transição $\delta : E \times \Gamma \to \mathcal{P}(E \times (\Gamma \times \{E, D, I\})$. ■

Evidentemente, o uso de não-determinismo simplifica muito o projeto de MTs. A seguir, ilustramos isso usando um exemplo.

Exemplo 86. Considere uma MT não determinística de duas fitas para a linguagem $\{xx \mid x \in \{0,1\}^*\}$:



■

Pode-se simular uma MT não determinística usando uma MT determinística de 3 fitas. A idéia é simular, sistematicamente, todas as computações de tamanho n antes de tentar uma possuindo $n+1$ passos. A execução da MT de 3 fitas procede de acordo com o seguinte algoritmo:

1. Inicialize a fita 3 com a palavra 1.
2. repita
   (a) copie a palavra de entrada da fita 1 para a fita 2.
   (b) Simule M na fita 2 de acordo com a palavra na fita 3.
   (c) se M parar em estado final, aceite.
   (d) apague a fita 2.
   (e) gere a próxima palavra para a fita 3.

É importante notar que as palavras na fita 3 representam possíveis transições não deterministas de M. Se n é o número máximo de transições não determinísticas em M, pode numerar as transições da MT usando palavras formadas por símbolos do conjunto $\{1..n\}$. Cada transição recebe um número e palavras sobre esse alfabeto representam computações em M seguindo as transições correspondentes a estes números.

Dessa forma, temos a equivalência entre MTs não determinísticas e MTs padrão.

19.6 Exercícios

1. Construa uma MT não determinística de duas fitas que reconheça a linguagem $\{w \mid \eta_0(w) = \eta_1(w)\}$.
2. Construa uma MT de duas fitas que reconheça $\{wcw \mid w \in \{0,1\}^*\}$.

20

Aula 20 - Gramáticas, Máquinas de Turing e Propriedades de Fechamento de Linguagens Recursivas e Recursivamente Enumeráveis

Objetivos

- Apresentar a equivalência de máquinas de Turing e gramáticas irrestritas.
- Apresentar os conceitos de autômato linearmente limitado, gramática sensível ao contexto e linguagem sensível ao contexto.
- Apresentar a hierarquia de Chomsky.
- Apresentar as propriedades de fechamento para as classes de linguagens recursivas e recursivamente enumeráveis.

20.1 Equivalência entre Gramáticas e MTs

Definição 78 (Gramática irrestrita)**.** Seja $G = (V, \Sigma, R, P)$ uma gramática. Dizemos que G é irrestrita se toda regra de R possui a forma $x \to y$, em que $x \in (V \cup \Sigma)^+$ e $y \in (V \cup \Sigma)^*$. ■

Teorema 27. *A linguagem gerada por uma gramática irrestrita é uma linguagem recursivamente enumerável.*

Demonstração. Seja $G = (V, \Sigma, R, P)$ uma gramática irrestrita. Para mostrar que G produz uma linguagem recursivamente enumerável, mostraremos (informalmente) como construir uma MT M de duas fitas não determinística tal que $L(G) = L(M)$. A fita 1 de M armazenará a palavra de entrada, que não será modificada, e a fita 2 uma etapa da derivação de w em G. O seguinte algoritmo descreve os passos a serem realizados pela MT em alto nível.

Escreva P (variável de partida) na fita 2 ;
while *true* **do**
  Selecione uma posição p na palavra contida na fita 2 (Não determinismo);
  Selecione uma regra $u \to v \in R$ (Não determinismo);
  if *se u ocorre a partir da posição p da fita 2* **then**
    Substitua u por v na fita 2 ;
    if *Se a palavra na fita* 2 *for igual a da fita 1* **then**
      Aceite ;
    end
  else
    Rejeite ;
  end
end

□

Teorema 28. *Uma linguagem recursivamente enumerável pode ser gerada por uma gramática irrestrita.*

Demonstração. Seja L uma linguagem recursivamente enumerável e uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ tal que $L(M) = L$. Mostraremos que existe uma gramática irrestrita $G = (V, \Sigma, R, P)$ que gera L. Informalmente, G possuirá regras para:

1. Gerar formas sentenciais do tipo $w\langle iw\rangle$, em que $w \in \Sigma^*$ e os símbolos $i, \langle, \rangle$ são variáveis em G.
2. Simular a execução de M sobre a configuração instantânea $\langle iw\rangle$, deixando o prefixo w inalterado.
3. Apagar a configuração instantânea quando ela for da forma $\langle xeay\rangle$, em que $e \in F$ e $\delta(e, a) = \perp$.

A primeira etapa é construir regras para produzir $w\langle iw\rangle$. Seja $\Sigma = \{a_1, ..., a_n\}$. Para cada $a_i \in \Sigma$ criaremos uma variável A_i em G e uma variável B, diferente de todo A_i. Criamos então as seguintes regras.

$$\begin{array}{lcl} P & \to & B\rangle \\ B & \to & a_k B A_k, 1 \leq k \leq n \\ B & \to & \langle i \\ A_k\rangle & \to & a_k\rangle, 1 \leq k \leq n \\ A_j a_k & \to & a_k A_j, 1 \leq j, k \leq n \end{array}$$

A segunda parte das regras simula a execução de M sobre $\langle iw\rangle$.

- Para cada transição da forma $\delta(e, a) = [e', b, D]$:

$$\begin{array}{lcl} ea & \to & be' \\ e\rangle & \to & be'\rangle, \ se\ a = \sqcup \end{array}$$

- Para cada transição da forma $\delta(e, a) = [e', b, E]$:

$$\begin{array}{lcl} cea & \to & e'cb, \ para\ cada\ c \in \Gamma \\ ce\rangle & \to & e'cb\rangle, \ para\ cada\ c \in \Gamma, a = \sqcup \end{array}$$

Finalmente, resta regras para apagar a configuração instantânea quando esta for da forma $\langle xeay\rangle$, com $e \in F$ e $\delta(e, a) = \perp$. Para isso, introduz-se uma variável nova $\#$:

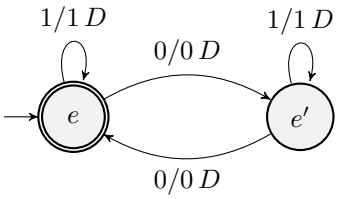
- Para cada par (e, a) tal que $e \in F$ e $\delta(e, a) = \perp$:

$$\begin{array}{lcl} ea & \to & a\# \\ e\rangle & \to & \#\rangle, se\ a = \sqcup \\ \#c & \to & \#, c \in \Gamma - \{\langle\} \\ c\# & \to & \#, c \in \Gamma - \{\langle\} \\ \langle\# \rangle & \to & \lambda \end{array}$$

□

Apresentamos um exemplo para ilustrar essa construção de gramática em termos de uma MT.

Exemplo 87. Vamos utilizar a construção do teorema anterior para construir a gramática irrestrita correspondente a seguinte MT:



Vamos considerar a palavra $w = 00$. Logo, temos as seguintes regras para construir a configuração inicial.

$$\begin{array}{lcl} P & \to & B\rangle \\ B & \to & 0BZ \,|\, 1BU \,|\, \langle e \\ Z\rangle & \to & 0\rangle \\ U\rangle & \to & 1\rangle \\ Z0 & \to & 0Z \\ Z1 & \to & 1Z \\ U0 & \to & 0U \\ U1 & \to & 1U \end{array}$$

Agora, vamos apresentar as regras para cada uma das transições.

- Transição $\delta(e, 0) = [e', 0, D]$.

$$e0 \to 0e'$$

- Transição $\delta(e, 1) = [e, 1, D]$.

$$e1 \to 1e$$

- Transição $\delta(e', 1) = [e', 1, D]$.

$$e'1 \to 1e'$$

- Transição $\delta(e', 0) = [e, 0, D]$.

$$e'0 \to 0e$$

Finalmente, incluimos regras para apagar a configuração $\langle xeay\rangle$.

- Para $\delta(e, \sqcup) = \perp$, temos:

$$\begin{array}{lcl} e\sqcup & \to & \langle\# \\ e\rangle & \to & \#\rangle \\ \#0 & \to & \# \\ \#1 & \to & \# \\ 0\# & \to & \# \\ 1\# & \to & \# \\ \langle\#\rangle & \to & \lambda \end{array}$$

Desta forma, temos que o conjunto total de regras da gramática é dado por:

$$
\begin{array}{lcl}
P & \to & B\rangle \\
B & \to & 0BZ \mid 1BU \mid \langle e \\
Z\rangle & \to & 0\rangle \\
U\rangle & \to & 1\rangle \\
e0 & \to & 0e' \\
e1 & \to & 1e \\
e'1 & \to & 1e' \\
e'0 & \to & 0e \\
e\sqcup & \to & \langle\# \\
e\rangle & \to & \#\rangle \\
\#0 & \to & \# \\
\#1 & \to & \# \\
0\# & \to & \# \\
1\# & \to & \# \\
\langle\#\rangle & \to & \lambda
\end{array}
$$

Usando o conjunto de regras desta gramática, podemos construir uma derivação para simular a computação da MT apresentada anteriormente para a palavra 00 como se segue.

$$
\begin{array}{lcll}
P & \Rightarrow & B\rangle & \text{Pela regra } P \to B\rangle \\
 & \Rightarrow & 0BZ\rangle & \text{Pela regra } B \to 0BZ \\
 & \Rightarrow & 00BZZ\rangle & \text{Pela regra } B \to 0BZ \\
 & \Rightarrow & 00\langle eZZ\rangle & \text{Pela regra } B \to \langle e \\
 & \Rightarrow & 00\langle eZ0\rangle & \text{Pela regra } Z\rangle \to 0\rangle \\
 & \Rightarrow & 00\langle e0Z\rangle & \text{Pela regra } Z0 \to 0Z \\
 & \Rightarrow & 00\langle e00\rangle & \text{Pela regra } Z\rangle \to 0\rangle \\
 & \Rightarrow & 00\langle 0e'0\rangle & \text{Pela regra } e0 \to 0e' \\
 & \Rightarrow & 00\langle 00e\rangle & \text{Pela regra } e'0 \to 0e \\
 & \Rightarrow & 00\langle 00\#\rangle & \text{Pela regra } e\rangle \to \# \\
 & \Rightarrow & 00\langle 0\#\rangle & \text{Pela regra } 0\# \to \# \\
 & \Rightarrow & 00\langle\#\rangle & \text{Pela regra } \langle\#\rangle \to \lambda \\
 & \Rightarrow & 00 &
\end{array}
$$

■

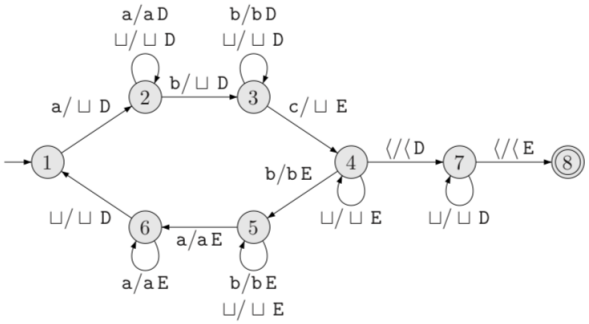
20.2 Autômatos Linearmente Limitados

Definição 79 (Autômato linearmente limitado)**.** Um autômato linearmente limitado (ALL) é uma máquina de Turing não determinística, $M = (E, \Sigma, \Gamma, \langle, \rangle, \delta, i, F)$, em que:

- $\rangle$ é um símbolo especial que não pode ser escrito na fita;
- A configuração inicial é $[i, \langle w\rangle]$; e
- Se $\delta(e, \rangle)$ é definida, então $\delta(e, \rangle) = [e', \rangle, E]$ para algum $e' \in E$.

■

Exemplo 88. A linguagem $\{a^n b^n c^n \mid n \geq 0\}$ é reconhecida pelo seguinte ALL:



■

Definição 80 (Gramática sensível ao contexto)**.** Uma gramática $G = (V, \Sigma, R, P)$ é dita ser sensível ao contexto (GSC) se toda regra $x \to y \in R$ é tal que $x, y \in (V \cup \Sigma)^*$ e $|x| \leq |y|$. ■

Exemplo 89. A seguir apresentamos uma GSC que gera a linguagem $\{a^n b^n c^n \mid n \geq 0\}$.

$$\begin{array}{lcl} P & \to & aPBc \mid abc \\ cB & \to & Bc \\ bB & \to & bb \end{array}$$

■

Definição 81 (Linguagem sensível ao contexto)**.** Uma linguagem é dita ser sensível ao contexto (LSC) se existe uma gramática sensível ao contexto que a gere. ■

Desta forma, temos que se $\lambda \in L$, então L não é uma LSC, visto que GSC não podem gerar λ.

20.3 A hierarquia de Chomsky

Com a apresentação dos ALLs completamos a caracterização de linguagens formais, seus reconhecedores e suas respectivas gramáticas. A tabela seguinte resume esses resultados.

Classe	Reconhecedor	Gramática
Ling. Reg.	AFD	Gram. Regular
Ling. Livre Cont.	APN	Gram. Livre cont.
Ling. Sens. Cont.	ALL	Gram. Sens. cont.
Ling. Rec.	MTs	Gram. irrestritas
Ling. Rec. En.	MTs	Gram. irrestritas

Note que é possível construir um AP que simule um AFD qualquer. Além disso, existem linguagens aceitas por APs que não podem ser reconhecidas por AFs quaisquer. Logo, podemos concluir que a classe das linguagens regulares é um subconjunto próprio da classe das linguagens livres de contexto, i.e., $LRs \subset LLCs$.

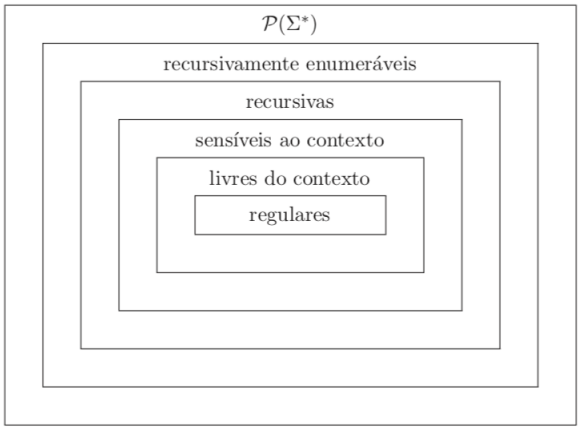
De maneira similar, pode-se mostrar que existem $LSCs$ que não são aceitas por nenhum AP e que, a menos de linguagens L tais que $\lambda \in L$, ALLs podem simular quaisquer APs. Logo, as linguagens livres de contexto são um subconjunto próprio de linguagens sensíveis ao contexto.

As linguagens sensíveis ao contexto podem ser reconhecidas por MTs que sempre param. Logo, podemos dizer que as linguagens sensíveis ao contexto são um subconjunto próprio de linguagens recursivas. Como linguagens recursivas são as que possuem MTs que as aceitam e param para todas as possíveis entradas e linguagens recursivamente enumeráveis são as que são possuem MTs que as aceitam, temos que a classe das linguagens recursivas é um subconjunto próprio das linguagens recursivamente enumeráveis.

Do apresentado acima, podemos concluir a seguinte hierarquia de linguagens:

$$LRs \subset LLCs \subset LSCs \subset LRecs \subset LREs \subset \mathcal{P}(\Sigma^*)$$

Note que a hierarquia acima, além de mostrar a relação de subconjunto entre diferentes classes de linguagens, também mostra o crescente poder computacional de reconhecedores e de suas gramáticas. Essa hierarquia que define o poder de máquinas, gramáticas e suas respectivas classes de linguagens é chamada de hierarquia de Chomsky. A figura abaixo ilustra esse relacionamento entre as diferentes classes de linguagens formais.



20.4 Propriedades de Fechamento de LREs e das Linguagens Recursivas

Teorema 29. *A classe das linguagens recursivas é fechada sobre união, interseção e complementação*

Demonstração. Sejam $M_1 = (E_1, \Sigma, \Gamma, \langle, \sqcup, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \Gamma, \langle, \sqcup, \delta_2, i_2, F_2)$ duas MTs que aceitam linguagens recursivas. Para mostrar o fechamento com respeito a interseção e união, utilizaremos a mesma técnica utilizada para AFs: simular a execução em paralelo. Para isso, vamos construir uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$, em que:

- $E = (E_1 \times E_2) \cup \{i, j\}$

Para facilitar, M será uma MT de duas fitas em que a primeira simulará a execução de M_1 e a segunda a execução de M_2. Para isso, primeiramente copiamos a palavra de entrada da fita 1 para a fita 2 usando as seguintes transições:

- $\delta(i, a, \sqcup) = [i, [a, D], [a, D]]$ para todo $a \in \Sigma$;
- $\delta(i, \sqcup, \sqcup) = [j, [\sqcup, E], [\sqcup, E]]$
- $\delta(j, a, a) = [j, [a, E], [a, E]]$, para todo $a \in \Sigma$;
- $\delta(j, \langle, \langle) = [[i_1, i_2], [\langle, D], [\langle, D]]$.

Logo após a cópia da palavra de entrada, a MT começa a operar no estado $[i_1, i_2]$. O restante de δ é construído a partir de δ_1 e δ_2 da seguinte maneira:

- Se $\delta_1(e_1, a_1) = [e_1', b_1, d_1]$ e $\delta_1(e_2, a_2) = [e_2', b_2, d_2]$ então, $\delta([e_1, e_2], a_1, a_2) = [[e_1', e_2'], [b_1, d_1], [b_2, d_2]]$;
- Se $\delta_1(e_1, a_1) = \delta_2(e_2, a_2) = \perp$ então $\delta([e_1, e_2], a_1, a_2) = \perp$.
- Se $\delta_1(e_1, a_1) = [e_1', b_1, d_1]$ e $\delta_2(e_2, a_2) = \perp$, então $\delta([e_1, e_2], a_1, a_2) = [[e_1', e_2], [b_1, d_1], [a_2, I]]$.
- Se $\delta_1(e_1, a_1) = \perp$ e $\delta_2(e_2, a_2) = [e_2', b_2, d_2]$, então $\delta([e_1, e_2], a_1, a_2) = [[e_1, e_2'], [a_1, I], [b_2, d_2]]$.

Para completar a construção para união basta fazer $F = (F_1 \times E_2) \cup (E_1 \times F_2)$ e para interseção $F = F_1 \times F_2$.

Finalmente, uma MT que aceita $\overline{L(M_1)}$ é $M = (E_1, \Sigma, \Gamma, \langle, \sqcup, \delta_1, i_1, E_1 - F_1)$. □

Teorema 30. *A classe das linguagens recursivamente enumeráveis é fechada sobre união e interseção.*

Demonstração. Construção análoga a de linguagens recursivas. □

Observe que as linguagens recursivamente enumeráveis não são fechadas sobre a complementação, visto que essa classe possui linguagens para as quais não existem MTs que param para toda entrada. Veremos um exemplo de tal linguagens em aulas posteriores.

O seguinte teorema mostra uma importante propriedade de linguagens recursivas.

Teorema 31. *Se L e $\overline{L}$ são linguagens recursivamente enumeráveis, então L é uma linguagem recursiva.*

Demonstração. Consequência do fechamento sobre união. □

20.5 Exercícios

1. Considere a seguinte linguagem $00(0+1)^*$. Apresente uma MT padrão que reconheça essa linguagem e construa a gramática irrestrita equivalente a sua MT.

21

Aula 21 - Conjuntos Enumeráveis e o Teorema de Cantor

Objetivos

- Apresentar os conceito de conjunto enumerável e alguns exemplos.
- Apresentar o Teorema de Cantor e sua aplicação para mostrar que alguns conjuntos não são enumeráveis.
- Discutir a relação entre o Teorema de Cantor e a decidibilidade de problemas.

21.1 Uma pequena revisão sobre funções

Nesta aula faremos uso de diversos conceitos vistos anteriormente por vocês. Apresentaremos as definições necessárias de maneira breve.

Definição 82. Sejam A, B conjuntos quaisquer. Dizemos que $f \subseteq A \times B$ é uma função se $\forall x.x \in A \rightarrow \exists! y.y \in B \wedge (x, y) \in f$. Usamos a notação $f : A \rightarrow B$ para representar o fato de que $f \subseteq A \times B$ é uma função. Usamos a notação $f(x) = y$ para representar $(x, y) \in f$. ■

Definição 83. Seja $f : A \rightarrow B$ uma função. Dizemos que f é uma função injetora se

$$\forall a_1.\forall a_2.a_1 \in A \wedge a_2 \in A \rightarrow f(a_1) = f(a_2) \rightarrow a_1 = a_2$$

■

Definição 84. Seja $f : A \rightarrow B$ uma função. Dizemos que f é uma função sobrejetora se

$$\forall y.y \in B \rightarrow \exists x.x \in A \wedge f(x) = y.$$

■

Definição 85. Seja $f : A \rightarrow B$ uma função. Dizemos que f é uma função bijetora se f for injetora e sobrejetora. ■

Fato 1. *Se $f : A \rightarrow B$ é uma função bijetora, então sua inversa $f^{-1} : B \rightarrow A$ também é uma função bijetora.*

Fato 2. *Sejam $f : A \rightarrow B$ e $g : B \rightarrow C$ duas funções bijetoras. Então, $g \circ f : A \rightarrow C$ também é uma função bijetora.*

21.2 Conjuntos enumeráveis

Definição 86 (Conjuntos de mesma cardinalidade)**.** Sejam A e B dois conjuntos quaisquer. Dizemos que A e B possuem a mesma cardinalidade, $A \sim B$, se existe uma função bijetora $f : A \to B$. ■

Usando a definição de conjuntos de mesma cardinalidade, podemos definir o conceito de conjunto infinito e finito.

Definição 87 (Conjuntos finitos e infinitos)**.** Seja $n \in \mathbb{N}$. Definimos $I_n = \{i \in \mathbb{N}^+ \mid i \leq n\}$. Dizemos que um conjunto A é finito se existe $n \in \mathbb{N}$, tal que $A \sim I_n$. Dizemos que um conjunto é infinito se não existe $n \in \mathbb{N}$ tal que $f : A \to I_n$. ■

As definições anteriores tem algumas consequências interessantes. Por exemplo, podemos mostrar que o conjunto dos números inteiros, $\mathbb{Z}$, possui a mesma cardinalidade dos números naturais, $\mathbb{N}$. Note que o bizarro deste fato é que $\mathbb{N} \subset \mathbb{Z}$! A seguinte função bijetora (prove este fato!) mostra que $\mathbb{N} \sim \mathbb{Z}$.

$$f(n) = \begin{cases} \frac{n}{2} & \text{se } n \text{ é par.} \\ \frac{1-n}{2} & \text{se } n \text{ é ímpar} \end{cases}$$

Pode-se mostrar que a relação $\sim$ é uma relação de equivalência, isto é, que esta é uma relação reflexiva, transitiva e simétrica.

Definição 88. Dizemos que um conjunto A é contável (enumerável) se este é finito ou se $A \sim \mathbb{N}$. Caso contrário, este é dito não contável. ■

Exemplo 90. Mostraremos que o conjunto $\mathbb{N} \times \mathbb{N}$ é contável. Para isso, vamos utilizar a seguinte função:

$$f(i,j) = \frac{(i+j-2)(i+j-1)}{2} + i$$

Pode-se mostrar que essa função $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ é uma função bijetora. ■

Teorema 32. *Suponha que A e B são conjuntos enumeráveis. Então, $A \times B$ é enumerável e $A \cup B$ é enumerável.*

Demonstração. Como A e B são enumeráveis, existem funções $f : A \to \mathbb{N}$ e $g : B \to \mathbb{N}$ bijetoras. As seguintes funções mostram que tanto o produto cartesiano quanto a união produzem conjuntos enumeráveis.

$$h(x,y) = (f(x), g(y))$$

$$h(x) = \begin{cases} f(x) & \text{se } x \in A \\ -g(x) & \text{se } x \in B \end{cases}$$

Observe que no caso da união, estamos definindo uma função $h : A \cup B \to \mathbb{Z}$. Porém, como apresentado anteriormente, $\mathbb{Z} \sim \mathbb{N}$. Como $\sim$ é uma relação transitiva, temos que $A \cup B \sim \mathbb{N}$ conforme requerido. □

Teorema 33. *Seja $\mathcal{F}$ uma família de conjuntos tal que $\mathcal{F}$ é enumerável e todo elemento de $\mathcal{F}$ é enumerável. Então $\bigcup \mathcal{F}$ é enumerável.*

Demonstração. Como $\mathcal{F}$ é contável, é possível indexar cada um dos conjuntos que compõe $\mathcal{F}$ por um número natural:

$$\mathcal{F} = \{A_1, A_2, ...\}$$

Porém, sabemos que cada elemento de $\mathcal{F}$ é também enumerável. Assim, temos que podemos indexá-los por número inteiro:

$$\begin{aligned} A_1 &= \{a_1^1, a_1^2, a_1^3 ...\} \\ A_2 &= \{a_2^1, a_2^2, a_2^3 ...\} \\ A_3 &= \{a_3^1, a_3^2, a_3^3 ...\} \\ &\vdots \end{aligned}$$

Note que $\bigcup \mathcal{F} = \{a_j^i \mid i, j \in \mathbb{N}\}$. Seja $f : \mathbb{N} \times \mathbb{N} \to \bigcup F$ definida como:

$$f(i,j) = a_j^i$$

Como sabemos que $\mathbb{N} \times \mathbb{N}$ é enumerável, temos que existe uma função bijetora $g : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$. Dessa forma $f \circ g : \mathbb{N} \to \bigcup \mathcal{F}$ é também uma função bijetora, concluindo a demonstração. □

Definição 89. Uma sequência de elementos sobre um conjunto A é uma função $f : I_n \to A$, em que $n \in \mathbb{N}$ é o tamanho da sequência f. ■

Intuitivamente, sequências são o equivalente matemática de listas presentes em linguagens de programação. Quando consideramos o conjunto A como sendo um alfabeto finito, temos que sequências nada mais são que strings sobre esse determinado alfabeto. A seguir apresentamos que o conjunto de sequências finitas sobre um conjunto A é enumerável, se A é um conjunto enumerável. Para mostrar esse fato, primeiramente provaremos um resultado auxiliar.

Lema 11. *Seja S_n o conjunto de todas as sequências de tamanho $n \in \mathbb{N}$ de um conjunto enumerável A. Então S_n é enumerável.*

Demonstração. No caso base, temos $n = 0$. Note que $I_0 = \emptyset$ e assim uma sequência de tamanho $n = 0$ é uma função $f : \emptyset \to A$, que é exatamente o conjunto $\emptyset$. Logo, $S_0 = \{\emptyset\}$ é enumerável. Para o passo indutivo, suponha $n \in \mathbb{N}$ arbitrário e que S_n é enumerável. Seja $F : A \times S_n \to S_{n+1}$ definida como:

$$F(a, f) = f \cup \{(n + 1, a)\}$$

Note que $F(a, f)$ é exatamente a operação de inserir a na sequência f. Logo, F é uma função bijetora. Dessa forma, temos que $A \times S_n \sim S_{n+1}$. Porém, como A e S_n são contáveis e o produto cartesiano de dois conjuntos contáveis é também contável, temos que S_{n+1} é contável. □

Teorema 34. *O conjunto de todas as sequências finitas de um conjunto contável A é também contável.*

Demonstração. Note que o conjunto de todas as sequências finitas é $\bigcup_{n\in\mathbb{N}} S_n$. Pelo Lema 1, temos que S_n é contável. Pelo teorema 2, temos que a união de famílias enumeráveis de conjuntos enumeráveis é também enumerável. □

Note que o teorema 3 implica que Σ^* é um conjunto enumerável.

21.3 O teorema de Cantor

A partir dos resultados apresentados acima pode-se pensar que todos os conjuntos são enumeráveis e que qualquer operação preserva a cardinalidade de conjuntos. Porém, não é esse o caso. A operação de conjunto potência não preserva a cardinalidade de um conjunto. Esse fato é conhecido como teorema de Cantor e é apresentado a seguir.

Teorema 35. *O conjunto $\mathcal{P}(\mathbb{N})$ não é enumerável.*

Demonstração. A demonstração consiste em mostrar que não existe uma função $f : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ bijetora. Para isso, mostraremos que $f : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ não pode ser sobrejetora o que é suficiente para mostrar que $\mathcal{P}(\mathbb{N})$ não é contável. Dizer que f não é sobrejetora pode ser descrito como:

$$\neg\forall D.D \in \mathcal{P}(\mathbb{N}) \to \exists n.n \in \mathbb{N} \wedge f(n) = D.$$

Usando álgebra Booleana, temos que a fórmula acima é equivalente a:

$$\exists D.D \in \mathcal{P}(\mathbb{N}) \wedge \forall n.n \in \mathbb{N} \to f(n) \neq D$$

Logo, o maior problema é encontrar um conjunto $D \subset \mathbb{N}$ tal que $f(n) \neq D$ para todo n. Basta fazer D ser o seguinte conjunto.

$$D = \{n \in \mathbb{N} \,|\, n \notin f(n)\}$$

Pela definição de D, temos que $D \subseteq \mathbb{N}$ e, portanto, $D \in \mathcal{P}(\mathbb{N})$. Suponha $x \in \mathbb{N}$ arbitrário e considere os seguintes casos.

1. $x \in f(x)$: Pela definição de D, podemos concluir que $x \notin D$. Como $x \in f(x)$ e $x \notin D$, temos que $f(x) \neq D$.
2. $x \notin f(x)$: Pela definição de D, podemos concluir que $x \in D$. Como $x \notin f(x)$ e $x \in D$, temos que $f(x) \neq D$.

Como $x \in \mathbb{N}$ é arbitrário temos que $\forall x.x \in \mathbb{N} \to D \neq f(x)$ e, portanto, f não pode ser sobrejetora. Dessa forma, temos que não existe função $f : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ bijetora, o que nos leva a concluir que $\mathcal{P}(\mathbb{N})$ não é enumerável. □

Observe que o conjunto de todas as linguagens sobre um determinado alfabeto não é enumerável.

21.4 Teorema de Cantor e Indecidibilidade

É fácil perceber que $|A| < \mathcal{P}(A)$, quando A é finito. Porém, uma consequência do teorema de Cantor, é que a mesma idéia vale para conjuntos infinitos. Se A é um conjunto enumerável infinito, então $A \not\sim \mathcal{P}(A)$. De maneira informal, isso quer dizer que o conjunto potência sempre terá maior cardinalidade que o conjunto que o originou.

Para entender a relação entre o teorema de Cantor e a decidibilidade de problemas, considere os seguintes fatos:

- Podemos considerar o código ASCII ou o Unicode como um alfabeto.
- Programas expressos em uma linguagem de programação qualquer são palavras sobre um alfabeto.

Como apresentado anteriormente, o conjunto palavras (sequências) sobre um alfabeto é enumerável. Dessa forma, o conjunto de todos os programas expressos em uma linguagem de programação é enumerável.

Se conseguimos expressar programas como sequências, podemos pensar que entradas para programas também podem ser representadas de maneira similar. Porém, cada programa possui seu próprio conjunto de entradas. Logo, podemos associar a um problema (a ser resolvido por um algoritmo) um conjunto que representa todas as suas possíveis entradas. Assim sendo, temos que para cada programa, existe uma linguagem correspondente as entradas por ele demandadas. Logo, o conjunto de todas as possíveis entradas para problemas é o conjunto de todas as possíveis linguagens sobre um dado alfabeto, isto é, $\mathcal{P}(\Sigma^*)$.

O que isso quer dizer? De maneira intuitiva, isso representa que o conjunto $\mathcal{P}(\Sigma^*)$ possui uma cardinalidade maior que Σ^*. Conforme discutido anteriormente, podemos representar programas como palavras em Σ^* e problemas como o conjunto de suas entradas, isto é, linguagens (elementos de $\mathcal{P}(\Sigma^*)$). Disso segue que **existem mais problemas que programas aptos a resolvê-los**. Logo, existem problemas para os quais não há solução.

21.5 Exercícios

1. Prove que a relação $\sim$ é uma relação de equivalência.
2. Prove que o conjunto de todos os números naturais ímpares é contável.

22

Aula 22 - A Tese de Church-Turing, codificação de problemas e a MT universal

Objetivos

- Apresentar a tese de Church-Turing.
- Discutir como instâncias de problemas podem ser representadas como uma linguagens sobre Σ.
- Apresentar a máquina de Turing universal.

22.1 A tese de Church-Turing

Definição 90. Se uma função é computável, então ela é computável por meio de uma máquina de Turing. ■

A definição anterior expressa que MTs são capazes de expressar **TODO** algoritmo. Em nosso estudo de decidibilidade, estaremos interessados em problemas de decisão, visto que esses possuem a mesma dificuldade que suas versões convencionais. Porém, para que uma MT solucione um problema decisão, é necessário que sua entrada seja expressa usando uma palavra sobre o alfabeto da MT em questão. Abordaremos o problema de codificação na próxima seção.

22.2 Codificação de Problemas

Definição 91. Um código para uma instância de um problema de decisão P sobre um alfabeto Σ consiste de uma palavra $w \in \Sigma^*$ que deve atender os seguintes requisitos:

- Para cada instância p do problema P deve existir pelo menos uma palavra de Σ^* que a represente.
- Cada palavra de Σ^* deve representar no máximo uma instância de P.
- Para cada palavra $w \in \Sigma^*$, deve ser possível determinar se w representa ou não uma instância de P.

■

Exemplo 91. Considere o seguinte problema de decisão: "Dado um número $n \in \mathbb{N}$, determinar se n é primo". Apresentaremos duas possíveis representações das instâncias deste problema:

- Para cada instância, "determinar se n é primo", existe uma palavra em $\{1\}^*$ que a representa: 1^n. A seguir apresentamos que requisitos anteriores são atendidos por essa representação.

- Para cada $n \in \mathbb{N}$, existe uma palavra que a representa: 1^n.
- Cada palavra 1^n representa uma instância do problema "determinar se n é primo".
- Para cada $w \in \{1\}^*$, é possível determinar se ela representa ou não uma instância: toda palavra representa uma instância do PD.

- Usando $\Sigma = \{0, 1\}$ podemos representar cada instância usando sua representação em binário. Com isso:
  - Cada instância é representada por inúmeras palavras, pois pode-se adicionar uma quantidade qualquer de 0s à esquerda da palavra e não alterar o número que ela representa.
  - Cada palavra $w \in \{0, 1\}^*$ representa, no máximo, uma instância. Note que $w = \lambda$ não representa instância alguma.
  - É possível determinar se uma palavra representa ou não uma instância.

■

Nos próximos exemplos, utilizaremos a notação $R\langle v_1, ..., v_n\rangle$ para representar uma instância de um problema de decisão com n parâmetros. Uma MT que resolve o problema em questão receberá $R\langle v_1, ..., v_n\rangle$ como entrada e para em estado final caso a resposta para o problema de decisão seja sim, e em estado não final (ou entra em loop) caso a resposta seja não. A figura abaixo ilustra esse fato.

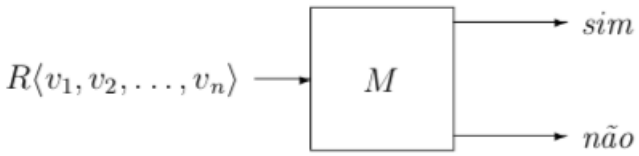


Figura 22.1: MT para problema de decisão.

A seguir, apresentamos um exemplo mais elaborado de representação de instâncias.

Exemplo 92. Considere o problema "determinar se uma gramática livre de contexto G gera uma palavra w, para G e w arbitrários". Para apresentarmos a codificação deste problema, vamos usar como exemplo a seguinte GLC:

$$\begin{array}{lcl} A & \to & aAa \,|\, B \\ B & \to & aB \,|\, CC \\ C & \to & b \,|\, \lambda \end{array}$$

Uma possível codificação é como se segue. Considere $G = (V, \Sigma_G, R, P)$ em que $V = \{A_1, ..., A_n\}$ e $\Sigma_G = \{a_1, ..., a_m\}$. Representaremos a entrada para o problema de decisão acima mencionado como uma palavra sobre $\{0, 1\}$. Para isso, precisamos encontrar uma maneira de representar cada um dos componentes de uma gramática G. Faremos isso da seguinte maneira:

- Representaremos uma variável A_i, $1 \leq i \leq n$, por 1^i. Na gramática anterior, temos que representaremos a variável A por 1, B por 11 e C por 111.
- Representaremos um símbolo do alfabeto a_j, $1 \leq j \leq m$, por 1^{n+j}. Dessa forma, temos que o símbolo a será representado por $1^{3+1} = 1^4$ e b por $1^{3+2} = 1^5$.
- Representaremos regras como uma sequência dos códigos dos simbolos que a compõe separados por 0. Como exemplo, considere a regra $A \to aAa$. Essa seria representada por $R\langle A\rangle 0 R\langle a\rangle 0 R\langle A\rangle 0 R\langle a\rangle = 101^4 0101^4$.
- Representamos o conjunto de regras de uma gramática separando o código correspondente a cada regra usando a palavra 00. Como exemplo, anteriormente apresentamos o código para a regra $A \to aAa$, que é:

$$R\langle A\rangle 0 R\langle a\rangle 0 R\langle A\rangle 0 R\langle a\rangle$$

Por sua vez, a regra $A \to B$ é representada como:

$$R\langle A\rangle 0 R\langle B\rangle$$

A sequência dessas duas regras é representada como:

$$R\langle A \to aAa\rangle 00 R\langle A \to B\rangle$$

- Finalmente, a representação completa da gramática é dada por: $R\langle(V, \Sigma_G, R, P)\rangle = 1^n 01^m 0 R\langle R\rangle$, em que $R\langle R\rangle$ é a representação do conjunto de regras de G, n é o número de variáveis e m o número de símbolos do alfabeto.

Com base na representação acima, podemos definir a instância do problema de decisão deste exemplo como $R\langle G, w\rangle = R\langle G\rangle 000 R\langle w\rangle$, em que $R\langle G\rangle$ é a codificação da gramática e $R\langle w\rangle$ é a representação da palavra de entrada. ■

22.3 Máquina de Turing Universal

Nessa seção discutiremos uma possível codificação de MTs que permita a construção de uma MT capaz de simular MTs recebidas como um código em sua fita de entrada. A MT capaz de executar outras MTs é denominada MT Universal. A definição seguinte apresenta essa possível codificação, seguida de um exemplo.

Definição 92 (Codificação de MTs)**.** Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ uma MT qualquer, em que $E = \{e_1, ..., e_n\}$, $\Gamma = \{a_1, ..., a_m\}$, $i = e_1$, $a_1 = \langle$, $a_2 = \sqcup$ e $F = \{f_1, ..., f_p\}$. Representaremos cada estado e_i por $R\langle e_i\rangle = 1^i$. De maneira similar, representaremos símbolos de Γ: $R\langle a_i\rangle = 1^i$. Representaremos a direção para mover o cabeçote da seguinte maneira: $R\langle E\rangle = 1$ e $R\langle D\rangle = 11$. Dado o exposto, a representação de M (supondo que M possui transições $t_1, ..., t_s$) é:

$$R\langle M\rangle = R\langle F\rangle 00 R\langle t_1\rangle 00...00 R\langle t_s\rangle$$

em que $R\langle F\rangle$ é a representação dos estados finais de M, que pode ser feita seperando o código de cada estado usando o símbolo 0:

$$R\langle F\rangle = R\langle f_1\rangle 0...0 R\langle f_p\rangle$$

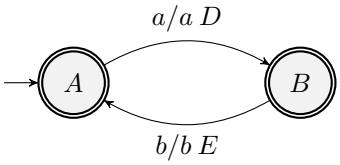
Cada transição t da forma $\delta(e, a) = [e', b, d]$ é representada como:

$$R\langle e\rangle 0 R\langle a\rangle 0 R\langle e'\rangle 0 R\langle b\rangle R\langle d\rangle$$

■

Apresentamos um exemplo da codificação apresentada.

Exemplo 93. Considere a seguinte MT $M = (\{A, B\}, \{a, b\}, \{\langle, \sqcup, a, b\}, \langle, \sqcup, \delta, A, \{A, B\})$.



Usando a codificação anterior, temos a seguinte representação dos estados e símbolos do alfabeto:

- $R\langle A\rangle = 1$, $R\langle B\rangle = 11$.
- $R\langle a\rangle = 1^3$, $R\langle b\rangle = 1^4$.

A transição t_1, $\delta(A, a) = [B, a, D]$, é representada por:

$$R\langle A\rangle 0R\langle a\rangle 0R\langle B\rangle 0R\langle a\rangle 0R\langle D\rangle$$

Por sua vez, a transição t_2, $\delta(B, b) = [A, b, E]$, é representada como:

$$R\langle B\rangle 0R\langle b\rangle 0R\langle A\rangle 0R\langle b\rangle 0R\langle E\rangle$$

Seguindo a codificação apresentada, temos que a MT acima possui a codificação:

$$R\langle A\rangle 0R\langle B\rangle 00R\langle t_1\rangle 00R\langle t_2\rangle$$

■

Após apresentarmos a codificação de MTs, podemos descrever uma MT que é capaz de simular outras MTs fornecidas como entrada em sua fita.

Definição 93 (Máquina de Turing Universal)**.** A MT universal é uma MT de 3 fitas que:

- A fita 1 possui a palavra de entrada $R\langle M, w\rangle$, que representa a máquina M a ser simulada e a palavra w a ser processada por M.
- A fita 2 é usada para simular a computação de M sobre a fita contendo w. Isto é, a fita 2 representa a fita de M.
- A fita 3 armazena o estado atual de M.

A MT universal é descrita pelo seguinte algoritmo:

Copie $R\langle w\rangle$ na fita 2 e posicione o cabeçote no início ;
Escreva $R\langle i\rangle$ na fita 3 e posicione o cabeçote no início;
while *true* **do**
  Seja $R\langle a\rangle$ a representação sobre o cabeçote na fita 2;
  Seja $R\langle e\rangle$ a representação sobre o cabeçote na fita 1;
  Procure $R\langle e\rangle 0R\langle a\rangle 0R\langle e'\rangle 0R\langle b\rangle 0R\langle d\rangle$ na fita 1 ;
  if *encontrou* **then**
    Substitua $R\langle e\rangle$ por $R\langle e'\rangle$ na fita 3 e volte seu cabeçote ao início ;
    Substitua $R\langle a\rangle$ por $R\langle b\rangle$ na fita 2 ;
    Mova o cabeçote da fita 2 na direção d ;
  else
    Pare ;
  end
end

■

Denominados a MT universal por U. Observe que a linguagem aceita por U é:

$$L(U) = \{R\langle M, w\rangle \mid M \textit{para com entrada } w\}.$$

Veremos na próxima aula que essa linguagem é um exemplo de linguagem recursivamente enumerável (existe uma MT que a aceita, U) que não é recursiva, pois existem entradas que fazem U não terminar sua computação.

22.4 Exercícios

1. Apresente uma codificação para os seguintes problemas de decisão:
   (a) Dado um AFD M e uma palavra w, determinar se $w \in L(M)$.
   (b) Dado um grafo G, determinar se G pode ser colorido usando k cores.
2. Mostre que a linguagem $\{R\langle M, w\rangle \mid M \textit{ passa duas vezes por algum estado de } M\}$ é recursiva.

23

Aula 23 - Indecidibilidade do problema da parada

Objetivos

- Apresentar a demonstração de que o problema da parada para MTs é indecidível.
- Apresentar a demonstração de que o problema da para linguagens de programação convencionais é indecidível.

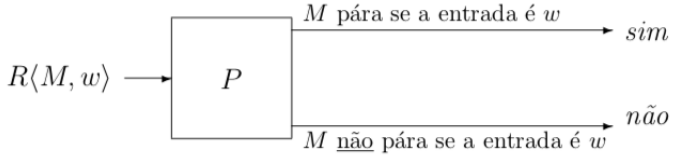
23.1 O problema da parada para MTs

Definição 94 (Problema da parada para MTs)**.** Dadas uma MT arbitrária M e uma palavra w, determinar se a computação de M para com entrada w. ■

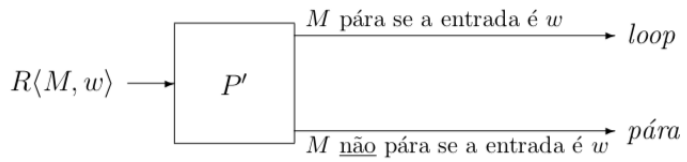
Vamos mostrar que o problema da parada é indecidível, isto é, não existe algoritmo capaz de dar a resposta correta (lembre-se: loop não é resposta!) para todas as possíveis instâncias deste problema.

Teorema 36. *O problema da parada para MTs é indecidível.*

Demonstração. Na demonstração, consideraremos que a palavra de entrada w é representada por ela própria, isto é, $R\langle w\rangle = w$. A prova será por contradição. Suponha que exista uma MT P que decida o problema da parada. Essa MT é apresentada na figura seguinte.



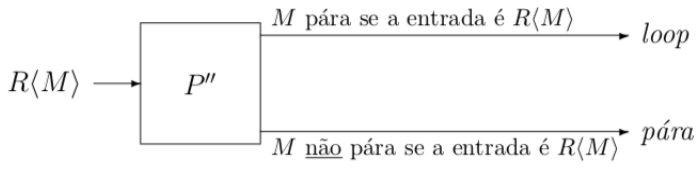
A partir de P, pode-se construir uma MT P' de forma que P' entra em loop exatamente nas situações em que P pára. Para isso, basta fazer para cada par (e, a) de P tal que $\delta(e, a) = \perp$, $\delta(e, a) = [l, a, D]$, em que l é um novo estado. Nesse novo estado basta adicionar as transições $\delta(l, a) = [l, a, D]$, para todo $a \in \Gamma$. A seguir mostramos o diagrama para a MT P'.



A partir de P' podemos construir outra máquina, chamada P''. Para construir P'' a partir de P' basta:

1. A partir da entrada $R\langle M\rangle$, gerar a palavra $R\langle M, R\langle M\rangle\rangle$, isto é, passaremos como entrada para a máquina M o código correspondente a M, $R\langle M\rangle$.
2. Agir como P' sobre essa entrada.

A seguir ilustramos a máquina P'':



De posse da MT P'', podemos encontrar a contradição desejada. Considere o que acontece quando submetemos $R\langle P''\rangle$ para a MT P'':

- Se P'' entra em loop com entrada $R\langle P''\rangle$, é porquê P'' para com entrada $R\langle P''\rangle$;
- P'' para com entrada $R\langle P''\rangle$, é porquê P'' entra em loop com entrada $R\langle P''\rangle$;

Ou seja, P'' para com entrada $R\langle P''\rangle$ se, e somente se, P'' não para com entrada $R\langle P''\rangle$, o que obviamente é uma contradição. Como podemos obter as MTs P' e P'', a suposição inválida é a da existência de P. Portanto não existe MT capaz de resolver o problema da parada. □

23.2 O Problema da Parada para Linguagens de Programação

Teorema 37. *O problema da parada para linguagens de programação é indecidível.*

Demonstração. Suponha que exista um procedimento `pare(x,w)` que dado um programa (expresso em texto) x e seus parâmetros de entrada (também expressos em texto) w, retorne verdadeiro se x para com entrada w. Podemos escrever a seguinte função.

```
procedimento D(x) {
   while (pare(x,x)) ;
}
```

Seja T o texto desse programa D. Então, se $D(T)$ para é porquê `pare(T,T)`, retorna falso, o que significa que $D(T)$ não para; assim, se $D(T)$ para, $D(T)$ não! Por outro lado, se $D(T)$ não para é porque $P(T, T)$ é verdadeiro, o que significa que $D(T)$ pára. Assim, se $D(T)$ não pára, $D(T)$ para. Contradição! Dessa forma, conclui-se que o problema da parada para linguagens de programação é indecidível. □

23.3 Consequência da Indecidibilidade do Problema da Parada

Considere a seguinte linguagem que representa as instâncias do problema da parada para MTs:

$$L_P = \{R\langle M, w\rangle \mid M \text{ para com entrada } w\}$$

A seguir apresentamos algumas consequências da indecidibilidade de L_P.

Teorema 38. *A linguagem L_P não é recursiva.*

Demonstração. Consequência da indecidibilidade do problema da parada. □

Teorema 39. *A linguagem L_p é recursivamente enumerável.*

Demonstração. Consequência da definição da MT universal, visto que L_P é a linguagem aceita pela MT universal. □

Teorema 40. *A linguagem $\overline{L_P}$ não é recursivamente enumerável.*

Demonstração. Ora, se $\overline{L_P}$ fosse recursivamente enumerável, temos que L_P seria recursiva, o que é uma contradição. □

23.4 Exercícios

1. Prove que o problema de determinar se um programa P, em uma linguagem de programação qualquer, imprime a string "Hello world!" é indecidível.

2. Mostre que se o problema da parada for decidível, então toda linguagem recursivamente enumerável seria recursiva.

24

Aula 24 - Redução de Problemas

Objetivos

- Apresentar o conceito de redução de problemas e mostrar como esse pode ser utilizado para demonstrar a decidibilidade de problemas.

24.1 Introdução

Existem duas técnicas principais para demonstração de indecidibilidade de problemas: *diagonalização* e *redução*. A diagonalização consiste em uma argumentação similar à feita na demonstração do teorema de Cantor e é exatamente o estilo de prova utilizado para mostrar a indecidibilidade do problema da parada.

Outra forma para demonstrar a indecidibilidade de problema é por redução. Na aula 23, mostrarmos que o problema da parada é indecidível. Podemos mostrar que um problema P é indecidível reduzindo o problema da parada a P. Intuitivamente, isso significa que podemos manipular entradas para o problema da parada para torná-las "similares" às entradas de P de forma que possamos usar P para resolver o problema da parada. Porém, como sabemos que o problema da parada é indecidível, podemos concluir por contradição que P também o deve ser.

24.2 Redução

Definição 95. Sejam $A \subseteq \Sigma^*$ e $B \subseteq \Delta^*$ duas linguagens sobre alfabetos Σ e Δ. Uma redução de A para B é uma função computável (MT):

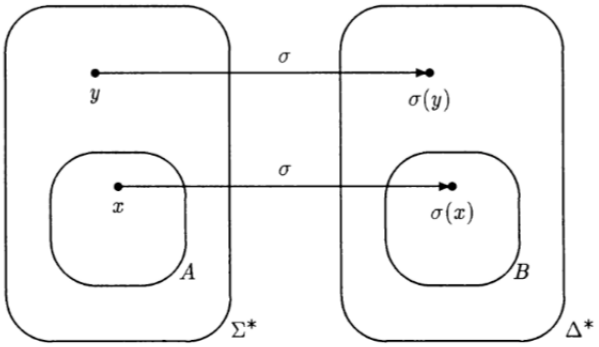
$$\sigma : \Sigma^* \to \Delta^*$$

tal que para todo $x \in \Sigma^*$, $x \in A \leftrightarrow \sigma(x) \in B$. ■

De maneira intuitiva, a condição

$$x \in A \leftrightarrow \sigma(x) \in B$$

expressa que palavras em A devem ser associadas a palavras em B pela função σ, e palavras que não estão em A devem ser associadas a palavras que não estão em B sobre σ. A figura seguinte ilustra essa situação.



É importante salientar que não há necessidade de σ ser uma função bijetora. A única exigência sobre σ é que esta seja uma função total e computável (i.e. existe uma MT que a calcule e pare para todas as palavras).

Porém, pode restar a pergunta: como definir uma MT que calcula uma função? A idéia é que a MT receba uma palavra $w \in A$ e pare com a fita contendo a palavra $\sigma(w) \in B$. Expressamos que A é redutível a B por uma função σ como $A \leq_\sigma B$. É fácil ver que a relação $\leq_\sigma$ é reflexiva e transitiva.

24.3 Provas por redução

Conforme discutido anteriormente, provar a indecidibilidade de um problema por redução consiste em "transformar" entradas de um problema sabidamente indecidível para entradas do problema cuja indecibilidade desejamos provar (P) de forma que seja possível obter a resposta do problema indecidível usando P. Porém, como mostrar a decidibilidade de problemas? Podemos usar redução para esse fim. Para isso, basta reduzir P a um problema que sabemos ser decidível.

Resumindo:

- Para demonstrar que um problema P é indecidível, basta reduzir um problema indecidível a P;
- Para demonstrar que um problema P é decidível, basta reduzir P a um problema decidível.

Exemplo 94. Vamos mostrar que o problema de "determinar se $L(M)$ é finita, em que M é um AFD" é decidível. Para isso, vamos reduzir o problema $L(M)$ é finita ao problema decidível de determinar se um grafo possui ciclos. Observe que a entrada para uma MT que resolve a questão $L(M)$ é finita é $R\langle M\rangle$, que consiste de uma representação de um AFD como uma palavra sobre um alfabeto Σ. A função de redução será uma MT que converte a representação de AFD para uma representação de grafos.

Como sabemos que $L(M)$ é infinita se o grafo de seu AFD possui ciclos, e pode-se converter entre representações de AFD em uma representação de grafo, temos que o problema em questão é decidível, visto que determinar se um grafo possui ciclos também o é. ■

O próximo exemplo mostra o uso de redução para demonstrar a indecidibilidade de um problema.

Exemplo 95. O problema da fita em branco pode ser enunciado da seguinte forma: dada uma MT M, determinar se M para com entrada λ. Mostraremos que o problema da fita em branco é indecidível reduzindo-o ao problema da parada. Para isso, vamos definir uma função σ que a partir de uma entrada $R\langle M, w\rangle$ para o problema da parada produz uma palavra $R\langle M'\rangle$ que denota o problema da fita em branco de forma que M' pare com entrada λ se, e somente se, M para com entrada w.

Porém, como especificar a função σ? A idéia é especificar a transformação que σ realiza sobre a entrada em "alto-nível". Apesar de usarmos uma descrição de alto-nível, os passos realizados devem concluir em tempo finito, garantindo a computabilidade de σ. Para o caso do problema da fita em branco, a função σ deve produzir $R\langle M'\rangle$ a partir de $R\langle M, w\rangle$ de forma que M' possua o seguinte comportamento:

1. M' apaga a entrada. Esse passo é simples, basta adicionar um novo estados l_1, l_2 a M com transições que apagam o conteúdo da fita até alcançar a primeiro símbolo de $\sqcup$. As transições incluídas seriam:
   - $\delta(l_1, a) = [l_1, \sqcup, D]$, para todo $a \in \Gamma - \{\langle\}$

- $\delta(l_1, \sqcup) = [l_2, \sqcup, E]$
- $\delta(l_2, \sqcup) = [l_2, \sqcup, E]$
- $\delta(l_2, \langle) = [i, \langle, D]$, em que i é o estado inicial de M

2. M' escreve w. Se $|w| = n$, então basta incluir $n + 2$ estados a MT resultante do passo anterior contendo transições para escrever cada um dos símbolos de w, que demanda $n + 1$ estados, e um último estado para voltar a fita ao início.

3. A partir desse passo, M' comporta-se exatamente como M. Isto é, o restando dos estados e transições de M' são exatamente os originais de M.

Com isso, temos que a MT M' pára com a fita em branco se, e somente se, a MT M para com a entrada w. Dessa forma, temos que o problema da fita em branco é indecidível. ■

Dizer que um problema é indecidível resume-se a mostrar que a linguagem de códigos de MTs que o solucionam não é uma linguagem recursiva. No caso do problema da fita em branco, a sua prova de indecidibilidade representa o fato de que a linguagem

$$\{R\langle M\rangle \mid \lambda \in L(M)\}$$

não é recursiva.

A seguir mostramos outro exemplo de prova de indecidibilidade.

Exemplo 96. Considere o seguinte problema: "determinar se a linguagem de uma MT M não é $\emptyset$". Vamos mostrar que esse problema é indecidível reduzindo o problema da parada a este. Para isso, vamos definir uma MT σ que transforma a entrada do problema da parada, $R\langle M, w\rangle$, na entrada do referido problema, $R\langle M'\rangle$. A função σ construirá $R\langle M'\rangle$ de forma que a MT M' se comporte da seguinte maneira:

1. M' apaga a palavra de entrada. Esse passo pode ser realizado conforme descrito na demonstração de indecidibilidade do problema da fita em branco.

2. M' escreve w e volta o cabeçote ao início da fita.

3. M' se comporta como M.

Dessa forma, temos que M' para com alguma palavra (e portanto $L(M') \neq \emptyset$) se, e somente se, M para com entrada w. ■

24.4 Exercícios

1. Apresente uma codificação para AFDs e outra para grafos. Explique, usando um pseudo-código, como converter sua codificação de AFD para codificação de grafos.

2. Para cada um dos problemas seguintes, mostre que esse é decidível ou não fornecendo uma prova por redução.

   (a) Dada uma MT M e um número $n \in \mathbb{N}$, determinar se M pára em n passos.

   (b) Dada uma MT M e uma palavra w, determinar se M move o cabeçote à esquerda durante o processamento de w.

   (c) Dada uma MT M, uma palavra w e um estado e, determinar se a computação de w por M atinge o estado e.

25

Aula 25 - O Teorema de Rice

Objetivos

- Apresentar e demonstrar o teorema de Rice.
- Apresentar como o teorema de Rice pode ser utilizado para provar que problemas são indecidíveis.

25.1 Introdução

Na aula 24, vimos alguns problemas indecidíveis e suas respectivas provas de indecidibilidade. Porém, de maneira geral, todos os problemas apresentados possuem a seguinte estrutura, quando expressos como uma linguagem de códigos de MTs:

$$\{R\langle M\rangle \mid L(M) \textit{ satisfaz a propriedade } P\}$$

Como exemplo, para o problema da fita em branco, temos que a propriedade P é $\lambda \in L(M)$. Nesta aula, apresentaremos o teorema de Rice, que generaliza a indecibilidade de problemas com essa estrutura. O teorema mostra que quase todas as propriedades sobre códigos de MTs resultam em linguagens que não são recursivas.

25.2 Teorema de Rice

O enunciado do teorema de Rice faz menção a propriedades não triviais, cuja definição é dada a seguir.

Definição 96. Dizemos que uma propriedade P sobre linguagens recursivamente enumeráveis é trivial se ela é verdadeira para todas ou nenhuma linguagem recursivamente enumerável. ■

Exemplo 97. Primeramente, note que existem apenas duas propriedades triviais: as funções constantes sobre valores booleanos. Quaisquer outras propriedades são não triviais. A seguir, apresentamos exemplos de propriedades não triviais de linguagens recursivamente enumeráveis.

- Determinar se $L(M) = \emptyset$.
- Determinar se $L(M) = \Sigma^*$.
- Determinar se $L(M)$ é finita.
- Determinar se $L(M)$ é regular.

■

A seguir apresentamos o enunciado e a demonstração do teorema de Rice.

Teorema 41. *Toda propriedade não trivial de linguagens recursivamente enumeráveis é indecidível.*

Demonstração. Suponha que P seja uma propriedade não trivial de linguagens recursivamente enumeráveis. Suponha, adicionalmente, que $P(\emptyset) = \bot$. Uma vez que P é não trivial, deve existir uma linguagem L tal que $P(L) = \top$. Seja M_L uma MT que aceita a linguagem L. Agora, apresentaremos uma função σ que reduz o problema da parada ao conjunto $\{R\langle M\rangle \mid P(L(M)) = \top\}$ mostrando que esse último não é uma linguagem recursiva. A função σ cria uma MT M' que sobre a entrada y, realiza os seguintes passos:

1. Escreve a palavra w, parte da entrada do problema da parada, depois de y, isto é, se a fita possui a forma $\langle y$ ela passará a ter o formato $\langle y[w$, em que $[$ é um novo símbolo que representará o início da fita de M, a MT fornecida como parte da entrada do problema da parada.
2. M' se comporta como M sobre a fita $[w$.
3. Se M para com entrada w, então M' executa a MT M_L sobre a entrada y e para se M_L parar com y.

Agora, consideramos os seguintes casos:

1. M para com entrada w. Nessa situação, temos que M' executa a MT M_L sobre a entrada y e, portanto irá parar se M_L parar com entrada y. Dessa forma, podemos dizer que $L(M') = L(M_L) = L$ e, foi suposto que $P(L(M')) = P(L) = \top$.
2. M não para com entrada w. Nessa situação, M' entra em loop junto com M e não aceita a entrada y e, dessa forma, temos que $L(M') = \emptyset$. Logo, $P(L(M')) = P(\emptyset) = \bot$.

Dessa forma, $P(L(M)) = \top$ se, e somente se, M para com entrada w, o que conclui a indecidibilidade deste problema. □

Dado o teorema de Rice, o questão de mostrar a indecidibilidade de um problema resume-se em provar que a propriedade que descreve esse problema é uma propriedade não trivial. Para isso, basta mostrarmos exemplos que tornam a propriedade verdadeira e exemplos que tornam a propriedade falsa. O exemplo seguinte prova a indecidibilidade de um problema usando o teorema de Rice.

Exemplo 98. O problema da fita em branco pode ser descrito pela seguinte linguagem:

$$\{R\langle M\rangle \mid \lambda \in L(M)\}$$

Para essa linguagem, temos que a propriedade $P(L(M)) = \lambda \in L(M)$. Note que a seguinte linguagem L_1 é recursivamente enumerável e $\lambda \in L_1$: $L_1 = \{0\}^*$. Além disso, a linguagem L_2 é também recursivamente enumerável: $L_2 = \{0\}^+$. Note que $\lambda \notin L_2$. Dessa forma, a propriedade $P(L(M)) = \lambda \notin L(M)$ é não trivial e, pelo teorema de Rice, a linguagem $\{R\langle M\rangle \mid \lambda \in L(M)\}$ não é recursiva, o que mostra que o problema da fita em branco é indecidível. ■

25.3 Exercícios

1. Considere os problemas seguintes. Caso seja possível, demonstre a sua indecibilidade utilizando o teorema de Rice.
   (a) Dada uma MT M e uma palavra w, determinar se M move o cabeçote à esquerda durante o processamento de w.
   (b) Dada uma MT M, uma palavra w e um estado e, determinar se a computação de w por M atinge o estado e.
   (c) Dada uma MT M, determinar se M aceita alguma palavra iniciada com 0.
   (d) Data uma MT M, determinar se M não aceita palavras de tamanho par.

26

Aula 26 - O Problema de Correspondência de Post

Objetivos

- Apresentar a prova de indecidibilidade do problema de correspondência de Post (PCP).

26.1 Introdução

O objetivo desta aula é apresentar a prova de indecibilidade do PCP, que é um problema simples sobre palavras de um certo alfabeto. A utilidade do PCP vem do fato que este pode ser utilizado para demonstrar a indecibilidade de diversos problemas de cunho "mais prático", como por exemplo, problemas relativos a gramáticas livres de contexto - que são utilizadas para descrever a sintaxe de linguagens de programação - e também a inexistência de estratégias para sempre vencer no jogo Tetris.

26.2 O Problema de Correspondência de Post

Definição 97. Um sistema de correspondência de Post (SCP) é um par (Σ, P), em que P é uma sequência finita de pares (x, y) tais que $x, y \in \Sigma^+$. ■

Uma solução para um SCP $S = (\Sigma, P)$ é uma sequência de pares de P tal que a palavra formada pela concatenação dos primeiros elementos dos pares é igual a palavra formada pela concatenação dos segundos elementos dos mesmos pares. A seguir definimos esse conceito formalmente.

Definição 98. Seja um SCP $S = (\Sigma, [(x_1, y_1), ..., (x_n, y_n)])$. Uma solução para S é uma sequência $i_1, ..., i_k$ tal que

$$x_{i1}x_{i2}...x_{ik} = y_{i1}y_{i2}...y_{ik}$$

para $1 \leq i_j \leq n$ para $1 \leq j \leq k$. ■

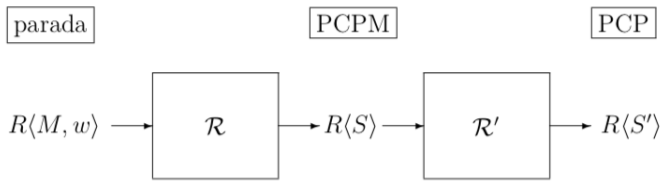
Antes de apresentar a prova de indecidibilidade do PCP, vamos considerar um exemplo.

Exemplo 99. Seja o SCP $(\{0, 1\}, [(10, 0), (0, 010), (01, 11)])$. Esse SCP possui três pares: $(x_1 = 10, y_1 = 0)$, $(x_2 = 0, y_2 = 010)$ e $(x_3 = 01, y_3 = 11)$. Uma possível solução para esse SCP seria a sequência 2131. Fica mais fácil visualizar a solução representando pares como frações. Por exemplo, representamos o par $(10, 0)$ por $\frac{10}{0}$. Dessa forma, usando os índices 2131 obtemos:

$$\frac{0}{010} \quad \frac{10}{0} \quad \frac{01}{11} \quad \frac{10}{0}$$

Evidentemente, repetições da sequência 2131 também formam soluções. Exemplos: 21312131, 2131213121312131, etc. ■

O problema de decisão a ser mostrado indecidível consiste em determinar se um SCP qualquer possui solução. Para isso, vamos introduzir um problema intermediário para facilitar a prova de indecidibilidade a partir do problema da parada. A figura a seguir ilustra as reduções a serem feitas.



O problema intermediário é conhecido como problema de correspondência de Post modificado e é definido a seguir.

Definição 99. O problema de correspondência de Post modificado (PCPM) consiste em determinar se um PCP arbitrário tem solução iniciada com o índice 1. ■

Exemplo 100. Seja o SCP $(\{0,1\}, [(0,010),(10,0),(01,11)])$. Esse SCP possui solução para o PCPM e esta é: 1232. ■

Agora vamos demonstrar a primeira parte da redução: que o PCPM é redutível ao PCP.

Teorema 42. *O PCPM é redutível ao PCP*

Demonstração. Seja um SCP $S = (\Sigma, P)$ com $P = [(x_1, y_1), ..., (x_n, y_n)]$. Sejam "#" e "*" símbolos não pertencentes a Σ. Considere:

- x'_i o resultado de colocar um "*" depois de cada símbolo de x_i. Por exemplo, se $x_i = 010$ então $x'_i = 0*1*0*$.
- y'_i o resultado de colocar um "*" antes de cada símbolo de y_i. Por exemplo, se $y_i = 100$ então $y'_i = *1*0*0$.

Seja o SCP $S' = (\Sigma \cup \{*, \#\}, P')$ em que P' é formado pelos pares:

- $(*x'_1, y_1)$, o primeiro par da solução.
- (x'_i, y'_i), para $1 \leq i \leq n$.
- $(\#, *\#)$.

Agora, basta mostrar que se S' tem solução então S também tem. Suponha que S' possui solução. Note que o único par de S' que começa com o mesmo símbolo é $(*x'_1, y'_1)$ e o único que termina com o mesmo símbolo é $(\#, *\#)$. Dessa forma, a solução para S' deve ser da forma:

$$\frac{*x'_1}{y'_1} \quad ... \quad \frac{x'_{i_k}}{y'_{i_k}} \quad \frac{\#}{*\#}$$

Ora, dessa forma, removendo o último par e todos os "*" presentes em cada x'_i, y'_i obtemos uma solução para o PCP a partir de uma do PCPM. □

Para concluir a indecidibilidade do PCP, vamos mostrar como reduzir a entrada do problema da parada para o PCPM. A idéia da redução é construir pares do PCPM de forma que cada par represente as possíveis configurações que a MT passa durante sua computação com a palavra w. Os pares são construídos de forma que o PCPM possui solução se, e somente se, a MT M para com a palavra w.

Teorema 43. *O PCPM é indecidível.*

Demonstração. Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ uma MT e $w \in \Sigma^*$ uma palavra. A partir destas construiremos um SCP $S = (\Delta, P)$, em que:

- $\Delta = \Sigma \cup \{*, \#\}$;

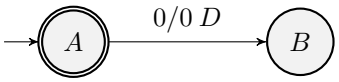
- O primeiro elemento de P é $(*, *\langle iw*)$
- Os demais pares de P são:
  - (c, c) para cada $c \in \Gamma$;
  - $(*, *)$;
  - Para cada $a, b \in \Gamma$ e $e, e' \in E$:

$$(ea, be'), \textit{ se } \delta(e, a) = [e', b, D]$$
$$(e*, be'*), \textit{ se } \delta(e, \sqcup) = [e', b, D]$$
$$(cea, e'cb), \textit{ se } \delta(e, a) = [e', b, E], \textit{ para cada } c \in \Gamma$$
$$(ce*, e'cb*), \textit{ se } \delta(e, \sqcup) = [e', b, E], \textit{ para cada } c \in \Gamma.$$

  - $(ea, \#)$, se $\delta(e, a) = \perp$, para cada $e \in E$ e $a \in \Gamma$;
  - $(e*, \#*)$, se $\delta(e, \sqcup) = \perp$, para cada $e \in E$.
  - $(c\#, \#)$, para cada $c \in \Gamma$.
  - $(\#c, \#)$, para cada $c \in \Gamma$.
  - $(*\#*, *)$.

Pode-se mostrar que M para com entrada w, se e somente se, o SCP S tiver solução iniciada com $(*, *\langle iw*)$. □

Exemplo 101. Considere a seguinte MT que aceita $1(0+1)^*$:



O SCP correspondente a M e a palavra 11 é formado pelos seguintes pares:

- $(*, *\langle A11*)$, que corresponde ao primeiro par.
- $(\langle, \langle)$, $(\sqcup, \sqcup)$, $(0, 0)$, $(1, 1)$ e $(*, *)$.
- A transição $\delta(A, 0) = [B, 0, D]$ produz o par: $(A0, 0B)$.
- Como $\delta(A, 1) = \delta(A, \langle) = \delta(A, \sqcup) = \perp$, temos os seguintes pares: $(A1, \#)$, $(A\langle, \#)$ e $(A\sqcup, \#)$.
- Como $\delta(A, \sqcup) = \perp$, temos o par $(A*, \#*)$.
- Como $\delta(B, 0) = \delta(B, 1) = \delta(B, \langle) = \delta(B, \sqcup) = \perp$, temos os seguintes pares: $(B0, \#)$, $(B1, \#)$, $(B\langle, \#)$, $(B\sqcup, \#)$.
- Como $\delta(B, \sqcup) = \perp$, temos: $(B*, \#*)$.
- Incluir os pares para cada símbolo de Γ: $(\langle\#, \#)$, $(\sqcup\#, \#)$, $(0\#, \#)$ e $(1\#, \#)$.
- Incluir os pares $(\#\langle, \#)$, $(\#\sqcup, \#)$, $(\#0, \#)$ e $(\#1, \#)$.
- Incluir o par $(*\#*, *)$.

■

A partir do PCP, prodemos provar a indecidibilidade de diversos problemas envolvendo gramáticas livres de contexto. Esse será o assunto da próxima aula.

27

Aula 27 - Problemas Indecidíveis sobre GLCs

Objetivos

- Apresentar como reduzir o PCP a diversos problemas envolvendo GLCs.

27.1 Introdução

O objetivo desta aula é apresentar a prova de indecibilidade de alguns problemas envolvendo GLCs. Mostraremos que o problema de determinar se uma gramática arbitrária é ambígua é indecidível. A prova de que esse problema é indecidível consiste em uma redução ao problema de correspondência de Post.

27.2 Reduzindo o SCP a GLCs

Dado um SCP $S = (\Sigma, P)$, em que $P = [(x_1, y_1), ..., (x_n, y_n)]$ podemos construir duas GLCs, que chamaremos de G_x e G_y, que representarão os componentes de um SCP. A construção seguinte ilustra como definir essas gramáticas.

Definição 100. Seja um SCP $S = (\Sigma, P)$, em que $P = [(x_1, y_1), ..., (x_n, y_n)]$ e $\{s_1, ..., s_n\}$ símbolos distintos tais que $\forall i. s_i \notin \Sigma$. O objetivo dos símbolos s_i é que esses são utilizados para indexar pares da solução do SCP. Usando o SCP, podemos construir duas GLCs sobre o alfabeto $\Sigma' = \Sigma \cup \{s_1, ..., s_n\}$:

- $G_x = (\{P_x\}, \Sigma', R_x, P_x)$, em que as regras são da forma:
  - $P_x \to x_i P_x s_i$, para $1 \leq i \leq n$;
  - $P_x \to x_i s_i$, para $1 \leq i \leq n$.
- $G_y = (\{P_y\}, \Sigma', R_y, P_y)$, em que as regras são da forma:
  - $P_y \to y_i P_y s_i$, para $1 \leq i \leq n$;
  - $P_y \to y_i s_i$, para $1 \leq i \leq n$.

■

A seguir ilustramos essa construção para um SCP de exemplo.

Exemplo 102. Seja o SCP $(\{0, 1\}, [(10, 0), (0, 010), (01, 11)])$. Como esse SCP possui 3 pares, usaremos os símbolos $\{s_1, s_2, s_3\}$ para indexar estes pares. As gramáticas produzidas para o SCP acima são:

- Gramática P_x

$$\begin{array}{rcl} P_x & \to & 10P_x s_1 \,|\, 0P_x s_2 \,|\, 01P_x s_3 \\ & | & 10s_1 \,|\, 0s_2 \,|\, 01s_3 \end{array}$$

- Gramática P_y

$$\begin{array}{rcl} P_y & \rightarrow & 0P_ys_1 \,|\, 010P_ys_2 \,|\, 11P_ys_3 \\ & | & 0s_1 \,|\, 010s_2 \,|\, 11s_3 \end{array}$$

Conforme visto na aula 25, temos que o SCP acima possui solução 2131. Logo, podemos mostrar uma derivações nessas gramáticas que refletem a solução do SCP. Primeiramente, a derivação em P_x:

$$\begin{array}{rcll} P_x & \Rightarrow & 0P_xs_2 & \{P_x \rightarrow 0P_xs_2\} \\ & \Rightarrow & 010P_xs_1s_2 & \{P_x \rightarrow 10P_xs_1\} \\ & \Rightarrow & 01001P_xs_3s_1s_2 & \{P_x \rightarrow 01P_xs_3\} \\ & \Rightarrow & 0100110s_1s_3s_1s_2 & \{P_x \rightarrow 10s_1\} \end{array}$$

A mesma derivação na gramática P_y gera:

$$\begin{array}{rcll} P_y & \Rightarrow & 010P_ys_2 & \{P_y \rightarrow 010P_ys_2\} \\ & \Rightarrow & 0100P_ys_1s_2 & \{P_y \rightarrow 0P_ys_1\} \\ & \Rightarrow & 010011P_ys_3s_1s_2 & \{P_y \rightarrow 11P_ys_3\} \\ & \Rightarrow & 0100110s_1s_3s_1s_2 & \{P_y \rightarrow 0s_1\} \end{array}$$

Observe que $P_x \Rightarrow^* 100110s_1s_3s_1s_2$ e $P_y \Rightarrow^* 100110s_1s_3s_1s_2$. ■

Do exemplo anterior, podemos perceber que se $i_1i_2...i_k$ é uma solução para um SCP, existem derivações nas gramáticas G_x e G_y tais que $P_x \Rightarrow x_{i_1}...x_{i_k}s_{i1}...s_{ik}$, $P_y \Rightarrow y_{i_1}...y_{i_k}s_{i1}...s_{ik}$. e $x_{i_1}...x_{i_k} = y_{i_1}...y_{i_k}$.

Desse fato, podemos concluir que o SCP possui solução se, e somente se, $L(G_x) \cap L(G_y) \neq \emptyset$ (pois ambas derivam a mesma palavra que corresponde a solução do SCP).

Teorema 44. *O problema de determinar se duas GLCs são disjuntas é indecidível.*

Demonstração. Consequência da construção apresentada para as gramáticas G_x e G_y. □

Conforme apresentado em aulas anteriores, as LLCs não são fechadas sobre a complementação. Porém, as gramáticas G_x e G_y possuem GLCs que geram o seu complemento.

Definição 101 (Gramática para o complemento de G_x e G_y)**.** Seja um SCP $S = (\Sigma, P)$, em que $P = [(x_1, y_1), ..., (x_n, y_n)]$ e $\{s_1, ..., s_n\}$ símbolos distintos tais que $\forall i.s_i \notin \Sigma$. A gramática que gera $\overline{G_x}$ é $(\{P, X\}, \Sigma \cup \{s_1, ..., s_n\}, R, P)$, em que o conjunto de regras R é dado por:

$$\begin{array}{lcll} P & \rightarrow & x_iPs_i & \textit{para } 1 \leq i \leq n \\ P & \rightarrow & Xa & a \in \Sigma \\ P & \rightarrow & s_iX & \textit{para } 1 \leq i \leq n \\ P & \rightarrow & yaXs_i & \textit{para } 1 \leq i \leq n \textit{ y é prefixo de } x_i, |ya| \leq |x_i|, \\ & & & ya \neq x_i, a \in \Sigma \cup \{s_1, ..., s_n\} \\ X & \rightarrow aX & a \in \Sigma \cup \{s_1, ..., s_n\} & \\ X & \rightarrow \lambda & & \end{array}$$

■

Usando a construção acima, podemos mostrar a indecidibilidade de outro problema envolvendo GLCs.

Teorema 45. *O problema de determinar se $L(G) = \Sigma^*$ é indecidível para uma GLC G arbitrária.*

Demonstração. Sejam G_x e G_y as gramáticas correspondentes a um SCP S. Construiremos uma gramática G tal que $L(G) = \Sigma^*$ se, e somente se, o SCP S não possui solução. Seja $L(G) = \overline{L(G_x)} \cup \overline{L(G_y)}$. Temos:

$$\begin{array}{ll} \overline{L(G_x)} \cup \overline{L(G_y)} = \Sigma^* & \leftrightarrow \\ \overline{\overline{L(G_x)} \cup \overline{L(G_y)}} = \overline{\Sigma^*} & \leftrightarrow \\ \overline{\overline{L(G_x)}} \cap \overline{\overline{L(G_y)}} = \emptyset & \leftrightarrow \\ L(G_x) \cap L(G_y) = \emptyset & \end{array}$$

Porém, conforme provado anteriormente, se um SCP S possui solução, então $L(G_x) \cap L(G_y) \neq \emptyset$. Dessa forma, temos que o SCP S não possui solução se $L(G_x) \cap L(G_y) = \emptyset$. Dessa forma, temos que o problema de determinar se $L(G) = \Sigma^*$ é indecidível. □

Encerramos essa aula apresentando a demonstração de que o problema de determinar se uma GLC é ambígua é também indecidível.

Teorema 46. *O problema de determinar se uma GLC arbitrária G é ambígua é indecidível.*

Demonstração. Sejam G_x e G_y as gramáticas geradas a partir de um SCP S e P_x e P_y os símbolos de partida de G_x e G_y, respectivamente. Considere a seguinte gramática $G = (\{P, P_x, P_y\}, \Sigma \cup \{s_1, ..., s_n\}, R_x \cup R_y \cup \{P \to P_x \mid P_y\}, P)$. Note que G será ambígua exatamente nas situações em que $L(G_x) \cap L(G_y) \neq \emptyset$, ou seja, quando o PCP possuir solução. □

Exercícios

1. Construa as gramáticas G_x e G_y para o seguinte SCP $S = (\{0, 1\}, P)$, em que $P = [(01, 011), (001, 01), (10, 00)]$.
2. Prove que os problemas seguintes são decidíveis ou indecidíveis.
   (a) Determinar se $L(G_1) \cap L(G_2) = \emptyset$ quando G_1 é uma gramática livre de contexto e G_2 uma gramática regular.
   (b) Determinar se $L(G_1) \cap L(G_2)$ é finito, quando G_1 e G_2 são gramáticas livres de contexto.
   (c) Determinar se $L(G_1) \cup L(G_2)$ é finito, quando G_1 e G_2 são gramáticas livres de contexto.
   (d) Determinar se $L(G_1) = L(G_2)$ é finito, quando G_1 e G_2 são gramáticas livres de contexto.