notas7_240419_145940.pdf

19 de abril de 2024 15:01

Teoria da Computação **FCT-UNL, 2º semestre 2023-2024**

Notas 7: Expressões regulares

Autor: João Ribeiro

Introdução

Nestas notas introduzimos o conceito de expressão regular – uma maneira diferente de representar uma linguagem sobre um alfabeto. Em particular, vamos estudar a relação entre expressões regulares e linguagens regulares.

7.1 Expressões regulares

Podemos interpretar um AFD ou AFN como uma maneira de descrever a linguagem que este aceita. Expressões regulares são outra forma de representar linguagens, originalmente introduzidas por Kleene [Kle56]. Informalmente, uma expressão regular sobre um alfabeto Σ consiste na combinação de elementos base, que representam as linguagens mais simples como $\varnothing$, $\{\varepsilon\}$, e $\{a\}$ para $a \in \Sigma$, através de um pequeno número de operações – união, concatenação, e fecho de Kleene. Além do seu interesse teórico, as expressões regulares são ferramentas importantes no desenho de compiladores e em algoritmos para processamento de texto. Apresentamos agora uma definição mais formal, por indução.

Definição 7.1 (Expressão regular) *O conjunto das expressões regulares sobre um alfabeto finito Σ, denotado por $\mathsf{RegExp}(\Sigma)$, é definido indutivamente da seguinte forma:*

- ***(Base 1)*** $\varnothing \in \mathsf{RegExp}(\Sigma)$;
- ***(Base 2)*** $\varepsilon \in \mathsf{RegExp}(\Sigma)$;
- ***(Base 3)*** *Se* $a \in \Sigma$, *então* $a \in \mathsf{RegExp}(\Sigma)$;
- ***(União)*** *Se* $E, F \in \mathsf{RegExp}(\Sigma)$, *então* $E + F \in \mathsf{RegExp}(\Sigma)$;
- ***(Concatenação)*** *Se* $E, F \in \mathsf{RegExp}(\Sigma)$, *então* $E \circ F \in \mathsf{RegExp}(\Sigma)$;
- ***(Fecho de Kleene)*** *Se* $E \in \mathsf{RegExp}(\Sigma)$, *então* $E^* \in \mathsf{RegExp}(\Sigma)$.

Como mencionámos acima, uma expressão regular representa uma linguagem. A seguinte definição especifica as regras que usamos construir esta linguagem a partir da expressão regular.

Definição 7.2 (Linguagem representada por expressão regular) *Dada uma expressão regular $Z \in \mathsf{RegExp}(\Sigma)$, a linguagem representada por Z, denotada por $L(Z)$, é definida indutivamente da seguinte forma:*

- ***(Base 1)*** $L(\varnothing) = \varnothing$;
- ***(Base 2)*** $L(\varepsilon) = \{\varepsilon\}$;
- ***(Base 3)*** *Se* $a \in \Sigma$, *então* $L(a) = \{a\}$;
- ***(União)*** *Se* $E, F \in \mathsf{RegExp}(\Sigma)$, *então* $L(E + F) = L(E) \cup L(F)$;
- ***(Concatenação)*** *Se* $E, F \in \mathsf{RegExp}(\Sigma)$, *então* $L(E \circ F) = L(E) \circ L(F)$;
- ***(Fecho de Kleene)*** *Se* $E \in \mathsf{RegExp}(\Sigma)$, *então* $L(E^*) = L(E)^*$.

Para simplificar a escrita de expressões regulares, em muitos casos podemos omitir os símbolos $\circ$ de concatenação entre duas expressões regulares. A interpretação desta expressão regular será clara pelo contexto. Por exemplo, quando trabalhamos sobre $\Sigma = \{0, 1\}$, em vez de escrevermos

$$Z = (1^* \circ (0 \circ 0)^*)^*$$

podemos simplesmente escrever

$$Z = (1^*(00)^*)^*.$$

Usando as regras acima, podemos calcular a linguagem $L(Z)$ representada por Z como

$$\begin{aligned} L(Z) &= L((1^*(00)^*)^*) \\ &= L(1^*(00)^*)^* \\ &= (L(1^*) \circ L((00)^*))^* \\ &= (L(1)^* \circ L(00)^*)^* \\ &= (L(1)^* \circ (L(0) \circ L(0))^*)^* \\ &= (\{1\}^* \circ \{00\}^*)^*. \end{aligned}$$

Concluímos, então, que $L(Z)$ é a linguagem das strings binárias que contêm um número par de 0s.

Podemos ter $L(Z) = L(Z')$ para expressões regulares Z e Z' diferentes. Neste caso dizemos que Z e Z' são *equivalentes*, e podemos escrever $Z = Z'$.

Podemos aplicar várias propriedades, como associatividade, distributividade, e comutatividade, para simplificar ou manipular uma expressão regular. Por exemplo, dadas expressões regulares E, F, e G, temos

$$\begin{aligned} &\varepsilon E = E\varepsilon = E, \\ &E + \varnothing = E, \\ &E + F = F + E, \\ &G(E + F) = GE + GF, \\ &(E + F)G = EG + FG. \end{aligned}$$

Para verificarmos que $E + \varnothing = E$, aplicamos as regras da Definição 7.2 para obtermos

$$L(E + \varnothing) = L(E) \cup L(\varnothing) = L(E) \cup \varnothing = L(E).$$

O leitor deverá convencer-se da veracidade das outras equações baseando-se nas definições acima.

O principal resultado destas notas é o seguinte teorema, que diz que as linguagens representadas por expressões regulares correspondem exactamente às linguagens regulares. Portanto, AFDs e expressões regulares têm o mesmo poder expressivo.

Teorema 7.1 *Uma linguagem L é regular se e só se existe uma expressão regular Z tal que $L = L(Z)$.*

Tendo em conta o Teorema 7.1 e os resultados das notas anteriores, podemos mostrar que uma linguagem L é regular das seguintes maneiras: Construir um AFD que aceita L; Construir um AFN que aceita L (mais fácil que a primeira opção); Construir uma expressão regular Z que gera L, i.e., $L(Z) = L$. Existem situações em que construir uma expressão regular para uma determinada linguagem é mais fácil do que construir um AFN, e vice-versa.

Ambas a direcções do Teorema 7.1 podem ser estabelecidas através do desenho de algoritmos que convertem uma expressão regular Z num AFN que aceita $L(Z)$ (o que mostra que $L(Z)$ é regular), e que convertem um AFD M numa expressão regular Z tal que $L(Z) = L(M)$ (o que mostra que cada linguagem regular é representada por uma expressão regular).

O Teorema 7.1 providencia uma caracterização algébrica das linguagens regulares. Este teorema mostra que as linguagens regulares são exactamente as linguagens obtidas a partir dos conjuntos base (também chamados de ***singletons***) $\varnothing$, $\{\varepsilon\}$, e $\{a\}$ para cada $a \in \Sigma$ através de operações de união, concatenação, e fecho de Kleene.

7.1.1 Exemplos

Exemplo 7.1 A expressão regular Z que representa a linguagem das strings binárias que começam com um 0 e acabam com um 1 é $Z = 0(0+1)^*1$. Intuitivamente, a expressão $(0+1)^*$ significa que podemos repetir o padrão $0+1$ um número arbitrário de vezes – em cada repetição de $0+1$, podemos escolher colocar um 0 ou um 1. Podemos usar as regras acima para determinar $L(Z)$ como

$$\begin{aligned} L(0(0+1)^*1) &= L(0) \circ L((0+1)^*) \circ L(1) \\ &= L(0) \circ L(0+1)^* \circ L(1) \\ &= L(0) \circ (L(0) \cup L(1))^* \circ L(1) \\ &= \{0\} \circ \{0,1\}^* \circ \{1\}, \end{aligned}$$

que corresponde à linguagem desejada.

Exemplo 7.2 A expressão regular Z que representa a linguagem das strings binárias em que cada 0 é seguido imediatamente por um número par de 1s consecutivos é $Z = 1^*(0(11)^*)^*$. Intuitivamente, o primeiro termo 1^* simboliza que podemos ter um número arbitrário de 1s antes do primeiro 0. O termo $(0(11)^*)^*$ diz que podemos repetir o padrão $0(11)^*$ um número arbitrário de vezes (que corresponde ao número de 0s na string). Em cada repetição deste padrão podemos escolher o número de 11s que adicionamos a seguir ao 0. Como adicionamos dois 1s de cada vez, garantimos que há sempre um número par de 1s imediatamente a seguir a cada 0.

Exemplo 7.3 A expressão regular Z que representa a linguagem das strings binárias que começam e acabam com o mesmo elemento é $Z = 0(0+1)^*0 + 1(0+1)^*1$. Intuitivamente, a expressão $0(0+1)^*0$ representa todas as strings que começam e acabam com 0, enquanto que $1(0+1)^*1$ representa as strings que começam e acabam com 1. Ao combiná-las com um $+$ representamos as strings que caem pelo menos num destes casos.

Exemplo 7.4 A expressão regular Z que representa a linguagem das strings sobre o alfabeto $\{a, b, c\}$ que têm sempre pelo menos um b ou c imediatamente depois de cada a é $Z = (b+c)^*(a(b+c)^+)^*$, onde usamos a expressão $(b+c)^+$ como abreviatura para $(b+c)^+ = (b+c)(b+c)^*$, ou seja, o padrão tem de ser incluído pelo menos uma vez. Intuitivamente, a expressão inicial $(b+c)^*$ indica que pode existir uma sequência arbitrária de bs e cs antes do primeiro a. A expressão $(a(b+c)^+)^*$ diz que podemos repetir o padrão $a(b+c)^+$ um número arbitrário de vezes (que corresponde ao número de ocorrências de a na string). Por cada ocorrência do padrão, este força-nos (através de $(a+b)^+$) a incluir ou um b ou um c imediatamente a seguir ao a.

Exemplo 7.5 Consideremos a expressão regular $Z = (a+b)^*(ad+bc)^*$. Como podemos argumentar que $ababbc \in L(Z)$? E que $bcbcab \notin L(Z)$?

Para vermos que $ababbc \in L(Z)$ podemos argumentar da seguinte forma. Temos que

$$
\begin{aligned}
L(Z) &= L((a+b)^*) \circ L((ad+bc)^*) \\
&= L(a+b)^* \circ L(ad+bc)^* \\
&= \{a, b\}^* \circ \{ad, bc\}^*.
\end{aligned}
$$

Por palavras, $L(Z)$ é a linguagem das strings sobre $\{a, b, c, d\}$ compostas pela concatenação de uma string de a's e b's com uma string de ad's e bc's. Temos de argumentar que $ababbc$ tem este formato, o que é verdade pois temos $ababbc = abab \circ bc$, e $abab \in \{a, b\}^*$ e $bc \in \{ad, bc\}^*$.

Para vermos que $w = bcbcab \notin L(Z)$ temos de argumentar que não existe nenhuma maneira de dividir esta string na concatenação de uma string de $\{a, b, \}^*$ com uma string de $\{ad, bc\}^*$. Suponhamos, com vista a uma contradição, que $w = w' \circ w''$ com $w' \in \{a, b\}^*$ e $w'' \in \{ad, bc\}^*$. Como $w_2 = c$, as únicas escolhas possíveis para w' são $w' = \varepsilon$ ou $w' = b$. No primeiro caso teríamos $w'' = (bc)(bc)(ab) \notin \{ad, bc\}^*$. No segundo caso teríamos $w'' = cbcab \notin \{ad, bc\}^*$, pois nenhuma string de $\{ad, bc\}^*$ começa com c. Concluímos que $w \notin L(Z)$.

7.2 Conversão de expressões regulares em AFNs

Começamos por ver como converter uma expressão regular Z num AFN que aceita a linguagem $L(Z)$. Esta tarefa usa ideias que já explorámos no contexto do estudo das propriedades das linguagens regulares. Intuitivamente, as Definições 7.1 and 7.2 especificam que uma expressão regular Z e a linguagem $L(Z)$ que esta representa são construídas a partir de objectos base (as expressões regulares $\varnothing$, ε, e a para $a \in \Sigma$, que representam as linguagens $\varnothing$, $\{\varepsilon\}$, e $\{a\}$, respectivamente) através das operações de união, concatenação, e fecho de Kleene. Quando estudámos as propriedades das linguagens regulares, vimos como, dados AFNs M_1 e M_2 que aceitam linguagens L_1 e L_2, construir AFNs que aceitam a união $L_1 \cup L_2$, a concatenação $L_1 \circ L_2$, e o fecho de Kleene L_1^*. A ideia agora consiste em combinar estas construções com AFNs que reconhecem as linguagens representadas pelas expressões regulares base $\varnothing$, ε, e a para $a \in \Sigma$, que ainda precisamos de construir.

No caso da expressão regular $\varnothing$ e da linguagem $L(\varnothing) = \varnothing$, podemos usar o AFN descrito na Figura 7.1.

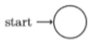


Figure 7.1: Diagrama de estados de um AFN que aceita a linguagem $L(\varnothing) = \varnothing$.

No caso da expressão regular ε e da linguagem $L(\varepsilon) = \{\varepsilon\}$, podemos usar o AFN descrito na Figura 7.2.

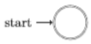


Figure 7.2: Diagrama de estados de um AFN que aceita a linguagem $L(\varepsilon) = \{\varepsilon\}$.

Finalmente, no caso da expressão regular a com $a \in \Sigma$ e da respectiva linguagem $L(a) = \{a\}$, podemos usar o AFN descrito na Figura 7.3.

Figure 7.3: Diagrama de estados de um AFN que aceita a linguagem $L(a) = \{a\}$.

Agora que obtivemos AFNs que aceitam todas as linguagens "base", lidamos com as operações de união, concatenação, e fecho de Kleene usando as ideias das notas anteriores.

Dadas duas expressões regulares E e F e AFNs M_E e M_F que aceitam $L(E)$ e $L(F)$, respectivamente, construímos um AFN para $E + F$, que representa a linguagem $L(E + F) = L(E) \cup L(F)$, através do procedimento que reproduzimos na Figura 7.4.

Dadas duas expressões regulares E e F e AFNs M_E e M_F que aceitam $L(E)$ e $L(F)$, respectivamente, construímos um AFN para $E \circ F$, que representa a linguagem $L(E \circ F) = L(E) \circ L(F)$, através do procedimento que reproduzimos na Figura 7.5.

Dadas uma expressão regular E e um AFN M_E que aceita $L(E)$, construímos um AFN para E^*, que representa a linguagem $L(E^*) = L(E)^*$, através do procedimento que reproduzimos na Figura 7.6.

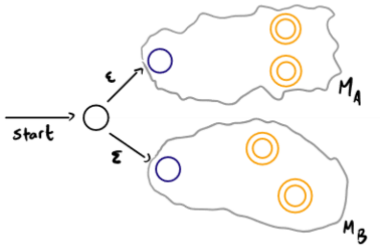
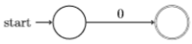


Figure 7.4: Ilustração do AFN que reconhece a união $A \cup B$ de linguagens regulares A e B. Os AFDs M_A e M_B reconhecem A e B, respectivamente. Os círculos azuis representam os estados iniciais originais de cada AFD, e os círculos laranjas os estados finais originais de cada AFD.

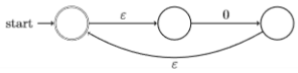
7.2.0.1 Exemplos de conversão de expressões regulares em AFNs

Exemplo 7.6 Queremos converter a expressão regular $0^* + 1^*$ no seu AFN correspondente. Procedemos passo-a-passo seguindo os procedimentos detalhados acima:

- AFN para 0:



- AFN para 0^*:



- AFN para 1:

1^*

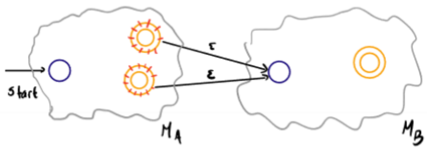
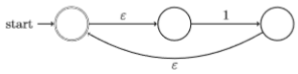
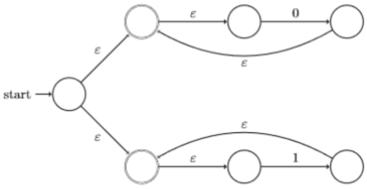


Figure 7.5: Ilustração do AFN que reconhece a concatenação $A \circ B$ de linguagens regulares A e B. Os AFDs M_A e M_B reconhecem A e B, respectivamente. Os círculos azuis representam os estados iniciais originais de cada AFD, e os círculos laranjas os estados finais originais de cada AFD. As riscas vermelhas mostram que estados deixam de ser finais no AFN.

- AFN para 1^*:

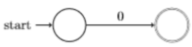


- AFN para $0^* + 1^*$:



Exemplo 7.7 Queremos converter a expressão regular $(00)^*$ no seu AFN correspondente. Procedemos passo-a-passo seguindo os procedimentos detalhados acima:

- AFN para 0:



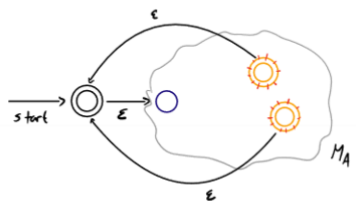
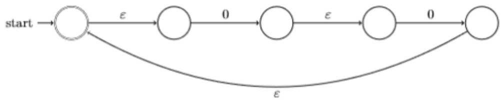


Figure 7.6: Ilustração do AFN que reconhece o fecho de Kleene A^* de uma linguagem regular A. O AFD M_A reconhece A. O círculo azul representa o estado inicial de M_A, e os círculos laranjas os estados finais de M_A. As riscas vermelhas mostram que estados deixam de ser finais no AFN.

- AFN para 00:

- AFN para $(00)^*$:



7.3 Conversão de AFDs em expressões regulares

Vimos acima como podemos converter uma expressão regular Z num AFN que aceita a linguagem representada por Z, usando observações de notas anteriores. Isto mostra que todas as linguagens representadas por expressões regulares são regulares.

Resta-nos mostrar que cada linguagem regular é representada por uma expressão regular. Existem várias maneiras de demonstrar isto. Usamos um método algébrico que nos permite converter um AFD numa expressão regular que gera a linguagem aceite pelo AFD através da resolução de um sistema de equações lineares sobre linguagens!

Começamos por descrever este método a partir de um AFD M concreto descrito na Figura 7.7.

lema (Arden): $E, F \in RegExp(\Sigma)$
Consideremos a eq linear $X = EX + F$
Então $X = E^*F$ é a menor solução desta eq
se $Z = EZ + F$ então $L(E^*F) \subseteq L(Z)$
Mais ainda, se $\varepsilon \notin L(E)$, então $X = E^*F$ é a única solução.

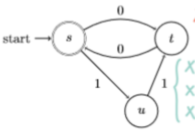
Dem. $E(E^*F) + F = E^+F + F = (E^+ + \varepsilon)F = E^*F$

$E^+ = EE^*$

$F = F \cdot \varepsilon$

concatenação

$X_s \to$ linguagem das seq. aceites por M a partir de s
$X_t \to$ linguagem das seq. aceites por M a partir de t
$X_u \to$ linguagem das seq. aceites por M a partir de u



$$\begin{cases} X_s = 0X_t + 1X_u + \varepsilon \\ X_t = 0X_s \\ X_u = 1X_t \end{cases}$$

Figure 7.7: Diagrama de estados de um AFD.

$$\begin{cases} X_s = 00X_s + 110X_s + \varepsilon = \\ = (00+110)X_s + \varepsilon = \\ = (00+110)^*\varepsilon = \\ = (00+110)^* \end{cases}$$

Ao estado inicial s associamos uma incógnita X_s, que representa a linguagem das strings aceites por M começando no estado s (X_s representa a linguagem $L(M)$ aceite por M). Analogamente, definimos X_t e X_u como incógnitas que representam as linguagem das strings aceites por M começando nos estados t e u, respectivamente. Podemos agora notar várias relações entre as linguagens X_s, X_t, e X_u que desconhecemos. Para começar, usando a notação das expressões regulares, temos a equação

$$X_u = 1X_t,$$

pois podemos ver X_u como o conjunto das strings que começam por 1 e o resto pertence a X_t (o conjunto das strings aceites por M começando no estado t, que é acessível de u por uma transição-1). De forma semelhante, temos também

$$X_t = 0X_s,$$

pois podemos ver as strings de X_t como começando por 0 e o resto pertencendo a X_s (o conjunto das strings aceites por M começando no estado s, que é acessível de t por uma transição-0). Finalmente, temos

$$X_s = 0X_t + 1X_u + \varepsilon,$$

pois podemos ver X_s como o conjunto das strings que, ou são a string vazia (porque s é estado final de M, e portanto M aceita a string vazia quando começa em s), ou começam por 0 e o resto pertence a X_t, ou começam por 1 e o resto pertence a X_u.

Em suma, temos o sistema de equações

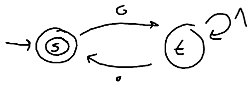
$$\begin{cases} X_s = 0X_t + 1X_u + \varepsilon \\ X_t = 0X_s \\ X_u = 1X_t, \end{cases}$$

e gostaríamos de determinar a incógnita X_s, que seria a expressão regular que representa $L(M)$. Substituindo a segunda e terceira equações na primeira, temos

$$\begin{aligned} X_s &= 0X_t + 1X_u + \varepsilon \\ &= 00X_s + 11X_t + \varepsilon \\ &= 00X_s + 110X_s + \varepsilon \\ &= (00+110)X_s + \varepsilon. \end{aligned} \tag{7.1}$$

AFD's $\Rightarrow$ RegExp

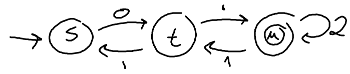
M



$$\begin{cases} X_s = 0X_t + \varepsilon \\ X_t = \underbrace{1}_{E}X_t + \underbrace{0X_s}_{F} \end{cases} \quad \begin{cases} - \\ X_t = 1^*0X_s \end{cases} \quad \begin{cases} X_s = \underbrace{01^*0}_{E}X_s + \varepsilon \\ - \end{cases}$$

$$\begin{cases} X_s = (01^*0)^*\varepsilon = (01^*0)^* \\ - \end{cases}$$

J



$$\begin{cases} X_s = 0X_t \\ X_t = 1X_s + 0X_u \\ X_u = \underbrace{2}_{E}X_u + \underbrace{1X_t + \varepsilon}_{F} \end{cases}$$

arden

$$\begin{cases} - \\ - \\ X_u = 2^*(1X_t + \varepsilon) \end{cases}$$

$$\begin{cases} - \\ X_t = 02^*(1X_t + \varepsilon) + 1X_s \\ - \end{cases} \quad \begin{cases} - \\ X_t = 02^*1X_t + 02^*\varepsilon + 1X_s \\ - \end{cases}$$

$$\begin{cases} - \\ X_t = 02^*1X_t + 02^* + 10X_t \\ - \end{cases} \quad \begin{cases} - \\ X_t = (\underbrace{02^*1 + 10}_{E})X_t + \underbrace{02^*}_{F} \\ - \end{cases}$$

arden

$$\begin{cases} - \\ X_t = (02^*1 + 10)^*02^* \\ - \end{cases} \Rightarrow X_s = 0X_t = 0(02^*1+10)^*02^*$$

Mas agora deparamo-nos com uma equação em que X_s aparece em ambos os lados. Como podemos simplificá-la?

Isto não é tão directo quanto fazer manipulações algébricas sobre os números reais e complexos, como em álgebra linear. O resultado chave que permite simplificar a equação acima é o seguinte, devido a Arden [Ard61].

Lema 7.1 (Lema de Arden) *Sejam E e F expressões regulares quaisquer e considere-se a equação linear $X = EX + F$. A menor solução para esta equação é $X = E^*F$ (no sentido em que se Z é qualquer outra solução para esta equação, então $L(E^*F) \subseteq L(Z)$). Mais ainda, se $\varepsilon \notin L(E)$ então esta solução é única.*

Antes de apresentarmos a demonstração do lema de Arden, mostramos como este é útil na resolução do sistema linear acima. Pela Equação (7.1), temos

$$X_s = (00+110)X_s + \varepsilon.$$

Aplicamos o lema de Arden com $E = 00+110$ e $F = \varepsilon$. Como $\varepsilon \notin L(E)$, obtemos a implicação

$$X_s = (00+110)X_s + \varepsilon = EX_s + F \implies X_s = E^*F = (00+110)^*\varepsilon = (00+110)^*.$$

Concluímos, então, que a linguagem $L(M)$ do AFD M da Figura 7.7 é representada pela expressão regular $(00+110)^*$.

Resta demonstrar o lema de Arden.

Demonstração:[Lema de Arden] Começamos por verificar que E^*F é uma solução desta equação. Temos que

$$E(E^*F) + F = E^+F + F = (E^+ + \varepsilon)F = E^*F,$$

como desejado.

Para vermos que E^*F é a menor solução desta equação (com respeito à inclusão de linguagens), seja Z uma expressão regular qualquer tal que $Z = EZ + F$. Temos de mostrar que $L(E^*F) \subseteq L(Z)$. Seja $k \in \mathbb{N}^+$ qualquer. Aplicando a equação $Z = EZ + F$ recursivamente k vezes, temos que

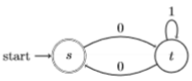
$$Z = EZ + F = E(EZ+F) + F = E^2Z + (E+\varepsilon)F = \cdots = E^kZ + (E^{k-1} + \cdots + E + \varepsilon)F \tag{7.2}$$

para qualquer $k \in \mathbb{N}^+$. Em particular, segue que $L(E^{k-1}F) \subseteq L(Z)$. Como isto se verifica para qualquer $k \in \mathbb{N}^+$, temos que $L(E^*F) \subseteq L(Z)$.

Resta mostrar que E^*F é a única solução quando $\varepsilon \notin L(E)$. Seja Z tal que $Z = EZ + F$ qualquer. Basta argumentar que $L(Z) \subseteq L(E^*F)$, pois já mostrámos acima que $L(E^*F) \subseteq L(Z)$. Seja $x \in L(Z)$ qualquer e $k = |x| + 1$. Como $\varepsilon \notin L(E)$, a linguagem $L(E^kZ)$ só contém strings de tamanho pelo menos $k > |x|$. Em particular, temos que $x \notin L(E^kZ)$. Como $x \in L(Z)$ por hipótese, segue pela Equação (7.2) que $x \in L((E^{k-1} + \cdots + E + \varepsilon)F) \subseteq L(E^*F)$. Como $x \in L(Z)$ era arbitrário, concluímos que $L(Z) \subseteq L(E^*F)$, e portanto $L(Z) = L(E^*F)$. ■

7.3.1 Exemplos de conversão de AFDs em expressões regulares

Exemplo 7.8 Vamos converter o seguinte AFD numa expressão regular correspondente através do método das equações lineares:



Este AFD gera o sistema de equações

$$\begin{cases} X_s = 0X_t + \varepsilon \\ X_t = 0X_s + 1X_t. \end{cases}$$

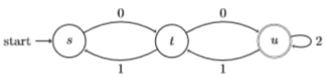
Como o estado inicial do AFD é s, o nosso objectivo é determinar a expressão regular representada pela incógnita X_s. Temos que

$$\begin{cases} X_s = 0X_t + \varepsilon \\ X_t = 0X_s + 1X_t \end{cases} \underset{\text{Lema de Arden}}{\Longrightarrow} \begin{cases} X_s = 0X_t + \varepsilon \\ X_t = 1^*0X_s \end{cases} \Longrightarrow \begin{cases} X_s = 01^*0X_s + \varepsilon \\ - \end{cases}$$

$$\underset{\text{Lema de Arden}}{\Longrightarrow} \begin{cases} X_s = (01^*0)^*\varepsilon = (01^*0)^* \\ - \end{cases}$$

Concluímos que uma expressão regular correspondente é $(01^*0)^*$. Existem várias maneiras de resolver o mesmo sistema de equações lineares, portanto é normal que duas pessoas a resolver o mesmo sistema cheguem a duas expressões regulares diferentes (mas sempre equivalentes!).

Exemplo 7.9 Vamos converter o seguinte AFD numa expressão regular correspondente através do método das equações lineares:



Este AFD gera o sistema de equações

$$\begin{cases} X_s = 0X_t \\ X_t = 1X_s + 0X_u \\ X_u = 1X_t + 2X_u + \varepsilon. \end{cases}$$

Como o estado inicial do AFD é s, o nosso objectivo é determinar a expressão regular representada pela incógnita X_s. Temos que

$$\begin{cases} X_s = 0X_t \\ X_t = 1X_s + 0X_u \\ X_u = 1X_t + 2X_u + \varepsilon \end{cases} \underset{\text{Lema de Arden}}{\Longrightarrow} \begin{cases} - \\ - \\ X_u = 2^*(1X_t + \varepsilon) \end{cases} \Longrightarrow \begin{cases} - \\ X_t = 1X_s + 02^*(1X_t + \varepsilon) \\ - \end{cases}$$

$$\Longrightarrow \begin{cases} - \\ X_t = 1X_s + 02^*1X_t + 02^* \\ - \end{cases} \underset{\text{Lema de Arden}}{\Longrightarrow} \begin{cases} - \\ X_t = (02^*1)^*(1X_s + 02^*) \\ - \end{cases}$$

$$\Longrightarrow \begin{cases} X_s = 0((02^*1)^*(1X_s + 02^*)) = 0(02^*1)^*1X_s + 0(02^*1)^*02^* \\ - \\ - \end{cases}$$

$$\underset{\text{Lema de Arden}}{\Longrightarrow} \begin{cases} X_s = (0(02^*1)^*1)^*0(02^*1)^*02^* \\ - \\ - \end{cases}$$

Concluímos que uma expressão regular correspondente é $(0(02^*1)^*1)^*0(02^*1)^*02^*$.

7.4 Representação de linguagens regulares: AFDs vs. expressões regulares

Vimos que os AFDs/AFNs e expressões regulares têm o mesmo poder expressivo, pois capturam exactamente a classe das linguagens regulares, apesar de serem bastante diferentes à primeira vista. Tendo em conta as várias representações de linguagens regulares existentes, é natural estudar a ***complexidade*** da descrição destas linguagens a partir de objectos como AFDs, AFNs, expressões regulares, e outros que não estudámos aqui. Esta é uma questão que já é estudada independentemente pelo menos desde trabalho de Meyer e Fischer [MF71] há mais de 50 anos atrás.

No que toca à comparação entre AFDs e AFNs, já vimos nas notas anteriores que AFNs permitem, por vezes, uma descrição muito mais simples de uma linguagem regular (em termos do número de estados do autómato que a aceita). Mais concretamente, existem linguagens regulares para as quais o menor AFD tem um número de estados exponencialmente maior do que o menor AFN. Relativamente à comparação entre autómatos e expressões regulares, tanto existem linguagens regulares cujo menor AFD tem um número de estados exponencialmente maior do que o tamanho da respectiva menor expressão regular [GN08], bem como o contrário [EZ74]. Este artigo da Wikipedia discute alguns destes aspectos.

7.5 Para explorar

Ainda existem muitas coisas que não sabemos sobre expressões regulares! Este artigo [EKSW04] colecciona alguns direcções de investigação interessantes. Para complementar a discussão nestas notas, sugerimos a leitura de [Sip13, Section 1.4].

Além do seu interesse teórico, expressões regulares têm muitas aplicações importantes no processamento de texto e em compiladores. Este link contém alguns exemplos simples da utilidade de expressões regulares. Se quiserem saber mais sobre as muitas aplicações práticas de expressões regulares, podem consultar o livro de Friedl [Fri06]. Naturalmente, também existem limitações!

References

[Ard61] Dean N. Arden. Delayed-logic and finite-state machines. In *2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*, pages 133–151, 1961.

[EKSW04] Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming-wei Wang. Regular expressions: new results and open problems. *Journal of Automata, Languages and Combinatorics*, 9(2-3):233–256, 2004.

[EZ74] Andrzej Ehrenfeucht and Paul Zeiger. Complexity measures for regular expressions. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing (STOC 1974)*, pages 75–79, 1974.

[Fri06] Jeffrey Friedl. *Mastering Regular Expressions*. O'Reilly Media, Inc., 2006.

[GN08] Wouter Gelade and Frank Neven. Succinctness of the complement and intersection of regular expressions. In Susanne Albers and Pascal Weil, editors, *25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, volume 1 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 325–336, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Kle56] Stephen C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, (34):3, 1956. Available at `https://www.rand.org/content/dam/rand/pubs/research_memoranda/2008/RM704.pdf`.

[MF71] Albert R. Meyer and Michael J. Fischer. Economy of description by automata, grammars, and formal systems. In *12th Annual Symposium on Switching and Automata Theory (SWAT 1971)*, pages 188–191, 1971.

[Sip13] Michael Sipser. *Introduction to the Theory of Computation*. CEngage Learning, 3rd edition, 2013.

$\Sigma = \{a, b\}$

PAR a's

2a ⇒ ímpares b's