LINGUAGENS FORMAIS E AUTÔMATOS

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Expressões regulares

Carlos Estevão Bastos Sousa

OBJETIVOS DE APRENDIZAGEM

- Definir expressões regulares e seus operadores.
- Exemplificar a construção de expressões regulares.
- Desenvolver o algoritmo de conversão de expressões regulares em autômatos finitos.

Introdução

Podemos afirmar que uma linguagem é regular se for possível refleti-la em um autômato finito. Entretanto, também é possível denotar uma linguagem regular por meio de uma expressão regular. Deste modo, considera-se que uma linguagem também é regular se existe uma expressão regular que a represente.

De acordo Stubblebine (2007), as expressões regulares são uma linguagem usada para análises e manipulação de caracteres. Elas são frequentemente aplicadas na realização de tarefas complexas em operações de pesquisa, substituição ou validação de dados.

Atualmente as expressões regulares estão presentes na maioria das linguagens de programação, de editores de textos, aplicativos, bancos de dados, entre outros.

Salienta-se que, conforme Lima (2020), atualmente vivemos a era da informação. Isto posto, poder possuir, difundir e gerenciar as informações representa um bem diferenciado e valiosíssimo em diversas esferas, como industriais, sociais e, inclusive, para o indivíduo comum.

Neste capítulo, você vai compreender o que são expressões regulares, quais operações podemos realizar com o uso delas, como construí-las e, por fim, como converter um autômato finito em uma expressão regular e vice-versa.

Expressões regulares e seus operadores

Conforme Jargas (2016), as expressões regulares são uma composição de símbolos, caracteres com funções especiais e caracteres literais que formam uma sequência, isto é, uma expressão que pode ser interpretada como uma regra que poderá indicar sucesso, se uma dada entrada pertencer à regra ou, insucesso, caso contrário. Normalmente, em linguagens de programação teremos como saída uma resposta semelhante, ou seja, *true* ou *false* para representar sucesso ou insucesso na aplicação de uma determinada cadeia a uma expressão regular.

Na matemática, mais precisamente na aritmética, podemos fazer uso de operações como adição e multiplicação a partir dos símbolos + e ×. Isto posto, a partir delas é possível elaborar equações como 5 × (97 + 32). De forma similar, também é possível representar as expressões regulares, por exemplo ***a* + b***.

As expressões regulares podem ser aplicadas com o intuito de testar se uma determinada cadeia corresponde a uma expressão de busca, localizar caracteres em uma cadeia, substituir cadeias em um texto, processar o formato de entradas, etc. Isto posto, podem ser aplicadas, por exemplo, a datas, os horários, números de IPs, endereços de *e-mail*, endereços de *websites*, entre outros.

Hopcroft, Ulman e Motwani (2003) afirmam que antes de conhecer as expressões regulares, faz-se necessário entender as três operações sob as linguagens regulares que os operadores das expressões regulares representam. Vejamos.

Para as operações apresentadas a seguir, considere o alfabeto $\Sigma = \{0,1\}$ e as linguagens $L = \{01, 11\}$ e $M = \{00, 01\}$.

- A união de duas linguagens L e M, também representada por $L \cup M$, tem como finalidade unir todas as cadeias presentes em ambas as linguagens. Deste modo, quando aplicada, $L \cup M = \{01, 11, 00\}$.
- Ainda considerando as linguagens L e M, a concatenação, representada por LM ou $L.M$, consiste em formar uma nova linguagem, na qual são obtidas as cadeias de L e concatenadas a qualquer cadeia pertencente a M, uma seguida da outra. Para este caso, temos $LM = \{0100, 0101, 1100, 1101\}$.
- O fecho estrela ou fechamento de Kleene é uma operação unária, assim, considerando somente a linguagem L. Esta operação, denotada por L^*, é uma concatenação sucessiva das cadeias de L, isto é, consiste em

tomar qualquer quantidade de cadeias de L, inclusive nenhuma, ou seja, $L^* = \{\varepsilon, 01, 11, 0101, 0111, 1111, 1101, 010111, ...\}$.

Saiba mais

Se você já utilizou algum *shell* do Linux ou fez alguma pesquisa em qualquer editor de texto que possibilite fazer buscas, há grandes chances de você ter utilizado alguma expressão regular. No Windows, por exemplo, ao entrar em qualquer diretório pelo Windows Explorer, é possível pesquisar por *.pdf. Perceba que neste caso você estará pesquisando todos os arquivos que terminam com a subcadeia .pdf, ou seja, possuem a extensão .pdf.

Em relação à precedência desses operadores, de início é utilizada a operação estrela, que possui precedência mais alta. Posteriormente, a concatenação, ou "ponto", na qual deve-se sempre agrupar os símbolos existentes da esquerda para a direita e, finalmente, a operação de união, em que a ordem é pouco importante, mas costumamos agrupar também da esquerda para a direita.

No Quadro 1 são apresentados os símbolos que costumam ser utilizados e a ordem de precedência dos operadores das expressões regulares.

Quadro 1. Precedência entre operações

Operação	Símbolo representante	Precedência
Fecho estrela	*	Alta
Concatenação	.	Média
União	+	Baixa

Ainda sobre as precedências, em casos em que não pretendemos seguir as apresentadas anteriormente, podemos fazer uso de parênteses, com o intuito de explicitar qual operação deverá ser executada primeiro.

Vejamos um exemplo.

Exemplo

Considere a expressão *ba + a + ba**.

Normalmente determinaríamos o início pela realização da operação de fecho estrela (*) sob *a* em ***a****; posteriormente a concatenação com ***b***, aqui representada apenas como **ba** e ***ba***, ou seja, ***b*** concatenado a ***a***; posteriormente, ***b*** concatenado a ***a****; e, finalmente, a operação de união (+), unindo ***ba*** com ***a*** e com ***ba***, assim, formando a expressão ***ba + a + ba****.

Diante da expressão formada, ao informarmos a precedência por parênteses poderíamos construir outra expressão, que apesar de similar, em muitos dos casos gera cadeias diferentes, como é o caso de (***b***(***a*** + ***a***) + ***b***)***a****.

Como visto, as expressões regulares são outra forma de definir linguagens regulares, inclusive, segundo Hopcroft, Ulman e Motwani (2003), elas podem ser consideradas uma "linguagem de programação", na qual podemos expressar algumas aplicações importantes como em pesquisas em textos, compiladores, entre outras.

Dentre os comandos de pesquisas do Unix, o `grep` utiliza uma notação semelhante às linguagens regulares. Outras notações podem ser utilizadas também em linguagens de programação como Java e Python, por exemplo. Estas são conhecidas também como RegEx (*regular expressions*).

Veja a expressão a seguir:

^*s*.....*e*$

Neste exemplo, considere:

- ^ como a definição de início da expressão;
- *s* como o uso do símbolo *s*;
- . representa a existência de qualquer símbolo;
- *e* representa a existência desse símbolo;
- $ representa o fim da cadeia, deste modo *e*$ caracteriza que a cadeia deverá terminar com *e*.

Vejamos o exemplo aplicado à linguagem de programação Python, com o intuito de verificar se uma determinada cadeia faz parte de uma linguagem.

Exemplo

```
import re

expressao = '^s.....e$'
cadeia = 'software'
resultado = re.match(expressao, cadeia)

if resultado:
  print("Cadeia reconhecida.")
else:
  print("Cadeia não reconhecida.")
```

Salienta-se que no exemplo estamos fazendo uso do módulo `re`, que possibilita o uso de expressões regulares na linguagem Python.

Quanto a saída do código, a mesma será `Cadeia reconhecida`, pois a cadeia *software* pertence à linguagem definida pela expressão ^*s*.....*e*$.

Construindo expressões regulares

Como apresentado anteriormente, as expressões regulares são constituídas pelas operações de concatenação, união e fecho estrela. A partir de um conjunto de símbolos advindos de um determinado alfabeto (Σ) e das operações citadas, torna-se possível construir expressões regulares; estas, por sua vez, representam linguagens regulares. Deste modo, possuem a mesma simplicidade e possibilidades de aplicações.

Conforme Hopcroft, Ulman e Motwani (2003), a álgebra das expressões regulares segue o mesmo padrão das álgebras de todos os tipos, isto é, inicia com algumas expressões elementares, normalmente fazendo uso de constantes e/ou variáveis, e aplica-as a um conjunto de operadores, geralmente fazendo uso de agrupadores como parênteses, por exemplo.

Considere o alfabeto Σ = {*a, b*}. Neste caso podemos afirmar que ***a*** e ***b*** são duas expressões regulares. Do mesmo modo, ***a*** **+** ***b***, representam uma expressão regular formada pela união das expressões ***a*** e ***b***. Nós utilizamos

as expressões regulares para definir uma linguagem regular tendo como base a própria expressão regular, deste modo são dadas algumas definições-base, a seguir.

Considerando a como um símbolo qualquer, logo $\boldsymbol{a}$ é uma expressão regular que denota a linguagem $\{a\}$. Deste modo $L(a) = \{a\}$.

$L(\varepsilon) = \{\varepsilon\}$ representa a linguagem que possui apenas o símbolo vazio e é denotada pela expressão $\boldsymbol{\varepsilon}$. Deste modo, considera-se também $\boldsymbol{\varepsilon}$ como a expressão que contém exclusivamente a cadeia vazia. O mesmo é válido para $L(\emptyset) = \emptyset$, isto é, a expressão $\emptyset$ não produz nenhuma cadeia.

Por fim, encerrando a base de construção, consideramos L, maiúscula e em itálico, como a variável que representa qualquer linguagem. Assim, considerando, por exemplo, uma expressão regular R_1, $L(R_1)$ é a linguagem denotada por R_1.

Entendida a base de uma expressão regular, conheceremos agora o processo de indução. Neste iremos apresentar quatro etapas indutivas, sendo três relacionadas ao uso dos operadores e uma para o uso dos parênteses. Vejamos a seguir.

Considere R_1 e R_2. Se estas são expressões regulares, logo $R_1 + R_2$ também é uma expressão regular, que consiste na união das primeiras, ou seja, a partir da união dessas expressões é possível fazer uso de qualquer cadeia existente em R_1 e R_2. Deste modo, $L(R_1 + R_2) = L(R_1)\, L(R_2)$.

Vejamos um exemplo.

Exemplo

Considere um alfabeto que contém os símbolos a e b, ou seja, $\Sigma = \{a, b\}$. Seja $\boldsymbol{a}$ uma expressão regular R_1 e $\boldsymbol{b}$ uma expressão regular R_2, o resultado da união entre elas consiste em todas as cadeias de ambas, ou seja, $\{a, b\}$.

De modo semelhante, considerando R_1 e R_2 como expressões regulares, então R_1R_2 também é uma expressão regular construída a partir da operação de concatenação, isto é, $L(R_1R_2) = L(R_1)L(R_2)$.

Vejamos um exemplo:

Exemplo

Para um $\Sigma = \{a, b\}$, considere ***a*** como a expressão R_1 e ***b*** como R_2, deste modo, R_1R_2 consiste na concatenação respectiva das expressões ***a*** e ***b***, ou seja, $L(R_1R_2) = \{ab\}$.

Salienta-se que na operação de concatenação torna-se facultativo o uso do ponto. Deste modo, determinada expressão pode ser escrita tanto como R_1R_2 quanto como $R_1.R_2$, e ambas possuirão o mesmo sentido. No entanto, normalmente utiliza-se a primeira expressão apresentada, ou seja, sem o uso de ponto.

Saiba mais

Normalmente optamos por representar a operação de concatenação sem a utilização do ponto (.), pois, como vimos, em diversas linguagens de programação este possui um significado diferente. Em Java, JavaScript e Python, por exemplo, o sinal de ponto pode representar a utilização de qualquer caractere.

Dada uma expressão regular R_1, a expressão R_1^* também será uma expressão regular composta a partir da operação fecho estrela ou fechamento de R_1, isto é, $L(R_1^*) = (L(R_1))^*$.

Vejamos um exemplo:

Exemplo

Para um $\Sigma = \{a, b\}$, considere ***a*** como a expressão que denota R_1, desse modo, ao aplicar a operação de fechamento em R_1, teremos $L(R_1^*) = \{\varepsilon, a, aa, aaa, ...\}$.

Por fim, para uma expressão regular R_1, (R_1), a R_1 entre parênteses também é uma expressão denotada pela mesma linguagem de R_1, isto é, $L((R_1)) = L(R_1)$.

Conforme vimos, a operação de fechamento tem precedência sobre a operação de concatenação. Deste modo, ao considerar a expressão ***aa****, sua resolução é dada como se segue:

- **Passo 1:** Consideramos inicialmente a expressão ***a****, pois ela possui precedência sob a concatenação, e tem como resultado {*ε, a, aa, aaa,* ...}, isto é, qualquer quantidade de *a*, inclusive nenhuma.
- **Passo 2:** Considere ainda ***aa****. O fragmento da expressão representada por ***a*** à esquerda representa, neste caso, uma unidade de *a* no início de qualquer cadeia formada por R_1. Assim, concatenamos uma unidade de *a* a qualquer quantidade de *a*, apresentado no Passo 1. Deste modo, é possível produzir cadeias como {*a, aa, aaa,* ...}.

De outro modo, ao fazer uso de parênteses na expressão R_2 como ***(aa)****, deverá ser resolvida primeiro a operação de concatenação, pois está entre parênteses, e, posteriormente, a de fechamento. Isto é, o resultado consiste em {*ε, aa, aaaa, aaaaaa,* ...}, ou seja, nenhuma ou uma quantidade par de símbolos *a*, o que torna $R_1 \neq R_2$. Vejamos:

- **Passo 1:** Considere a expressão R_2 = ***(aa)****. O primeiro passo é considerar os operadores ***a*** de forma individual, deste modo, ao efetuar a operação de concatenação sob eles, o resultado consistirá em ***aa***.
- **Passo 2:** Aprendemos que a operação de fechamento, quando aplicada a algum símbolo ou conjunto de símbolos, possui como resultado qualquer quantidade deste, inclusive nenhuma. Isto posto, ao aplicar a operação de estrela sob ***aa***, teremos nenhuma (*ε*) ou várias representações de *aa* concatenadas. Deste modo, $L(R_2)$ = {*ε, aa, aaaa, aaaaaa,* ...}.

Com base no exposto, e no $\Sigma = \{a, b\}$, vejamos alguns exemplos no Quadro 2.

Quadro 2. Exemplificação de expressões regulares e seus significados

Expressão regular	Linguagem
ab	Possibilita a leitura apenas da cadeia *ab* por meio da operação de concatenação.
ab*	Possibilita a leitura de qualquer cadeia iniciada pelo símbolo *a* seguido por qualquer quantidade de *b*, inclusive nenhum. Neste exemplo é feito uso da operação de concatenação e fechamento.

(Continua)

(Continuação)

Expressão regular	Linguagem
$(a + b)^*$	Possibilita a leitura vazia (ε) ou de qualquer cadeia que contenha os símbolos *a* ou *b*. Neste exemplo são utilizadas as operações de união e fechamento. De forma implícita, temos a operação de concatenação, pois a operação de fechamento consiste em uma generalização da concatenação.
$b(a + b)^* a$	Possibilita a leitura de todas as cadeias que iniciam com *b* e terminam com *a*.

Vejamos agora um exemplo de problema que consiste em, com base em uma linguagem regular, construir a expressão regular que a denote.

Exemplo

Considere L_1 como uma linguagem regular de forma que $\{w \in \Sigma^*$, com $\Sigma = \{0,1\} \mid |w|$ é múltiplo de 7$\}$.

Para a resolução deste problema, inicialmente, faz-se necessário entendê-lo. Desse modo, consideramos *w* como qualquer cadeia pertencente a $\Sigma = \{0, 1\}$, ou seja, qualquer cadeia que possui combinações dos símbolos 0 e 1. Por fim, solicita-se também que o comprimento de *w* seja múltiplo de 7.

Vejamos a resolução:

Ao considerar Σ como qualquer símbolo pertencente ao alfabeto, ou seja, 0 e 1, para construirmos a expressão regular, com cadeias de tamanho de comprimento múltiplo de sete, podemos começar pelo caso-base, que é uma expressão de comprimento sete. Assim temos:

$$\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma$$

Por fim, considerando que 0 é múltiplo de qualquer número, podemos inserir a operação de fecho estrela sobre a expressão elaborada para o caso base. Assim, temos qualquer quantidade da expressão inserida entre parênteses.

$$(\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma)^*$$

Note que nesta expressão regular é possível produzir cadeias como $\{\varepsilon$, 0000000, 1111111, 00000000000000, 00000001111111, ...$\}$.

Convertendo expressões regulares em autômatos finitos

Anteriormente vimos que toda linguagem regular pode ser representada por uma expressão regular. Dito isto, se *R* é uma expressão regular, então *L*(*R*) é uma linguagem regular; assim, se *L* é uma linguagem regular, então existe uma expressão *R* tal que *L*(*R*) = *L*.

Para Menezes (2000), as expressões regulares tratam-se de um formalismo denotacional e gerador, pois é possível tanto inferir como construir cadeias de uma dada linguagem. Sipser (2007) afirma que as expressões regulares e os autômatos finitos possuem equivalência. Vejamos.

> Expressões regulares e autômatos finitos são equivalentes em seu poder descritivo. Esse fato é surpreendente porque autômatos finitos e expressões regulares aparentam superficialmente ser bastante diferentes (SIPSER, 2007, p. 68).

Levando em consideração que tanto os autômatos finitos quanto as expressões regulares denotam linguagens regulares, é possível que haja a necessidade da conversão entre essas duas formas. Deste modo, a seguir vamos compreender como funciona a conversão de uma expressão regular para um autômato finito.

Convertendo expressões regulares para autômatos finitos

A realização da conversão de expressões regulares para autômatos finitos será apresentada a partir do algoritmo Thompson (1968), também conhecido como algoritmo de construção de Thompson. Para a conversão, faz-se necessário compreender os casos-base. Com esse conhecimento, é possível aplicar os casos-base a casos mais complexos e obter a solução desejada. Salienta-se que, como forma de tornar a conversão um processo mais simples, será explanada a conversão de uma expressão regular para um autômato finito não determinístico de movimentos vazios (AFNε), o qual pode ser facilmente convertido para um autômato finito não determinístico (AFN) ou para um autômato finito determinístico (AFD).

Iniciaremos entendendo o processo de conversão a partir das expressões-base, ou seja, do uso de símbolos isolados, ε e ∅. Posteriormente apresentaremos como fazer a combinação desses autômatos com o intuito de formar autômatos maiores e que aceitam as operações de união, concatenação e fechamento estrela. Vejamos.

Caso-base 1

Para o primeiro caso, considere *a* a expressão *R*, apresentada na Figura 1.

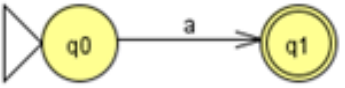

Figura 1. R = a.

Note que a linguagem regular é composta apenas por *a*, ou seja, $L(R) = \{a\}$, assim, podemos representá-la com um autômato com apenas uma transição do estado inicial para o estado final fazendo a leitura do símbolo *a*.

Caso-base 2

Para o segundo caso, considere ε como a expressão *R*, assim, podemos representá-la de acordo com a Figura 2.

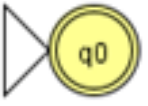

Figura 2. $R = \varepsilon$.

Neste caso, temos uma linguagem que aceita apenas a cadeia vazia, assim, deve possuir um estado inicial e estado de aceitação com ligação entre eles a partir de uma transição ε, podendo, ainda, como representando na Figura 2, fazer uso do mesmo estado como inicial e de aceitação.

Caso-base 3

Para o terceiro caso, a expressão $R = \emptyset$; podemos representá-la de acordo com a Figura 3.

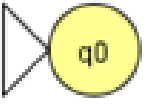

Figura 3. R = ∅.

Para uma expressão vazia, representamos um autômato que não aceita nenhum símbolo, ou seja, não há um estado de aceitação.

Como visto, há três partes que representam a base da construção de um autômato. Neste momento, baseados nas partes citadas, iremos conhecer as três partes da indução. Vejamos.

Caso indutivo 1: Operação de união

Considere duas expressões regulares R_1 e R_2, denotadas por **a** e **b,** nas quais $L(M_1) = L(R_1)$ e $L(M_2) = L(R_2)$, representadas pelos autômatos finitos apresentados na Figura 4.

Figura 4. $R_1 = a$ e $R_2 = b$.

Ao aplicar a operação de união sob as duas expressões, $R_1 + R_2$, temos como resultado um autômato como o apresentado na Figura 5.

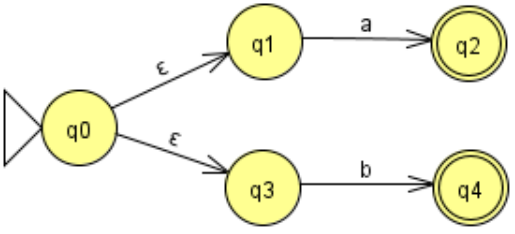

Figura 5. $R_1 + R_2$.

Atente-se ao fato de que é criado um novo estado inicial, o qual possui transições vazias para cada estado inicial anteriormente existente em R_1 e R_2.

Caso indutivo 2: Concatenação

Ainda sob as expressões R_1 e R_2, para concatená-las inserimos os autômatos, dispostos tal qual a expressão R_1R_2, e posteriormente os ligamos a partir de uma transição ε. O resultado dessa ação é apresentado na Figura 6.

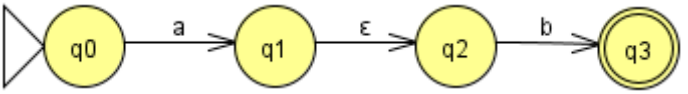

Figura 6. R_1R_2.

Note que para a concatenação é necessário seguir a ordem dos símbolos utilizados na expressão regular.

Caso indutivo 3: Fecho de Kleene (operação estrela) ou concatenação sucessiva

Nesta operação, ocorre uma transformação de modo a aceitar símbolos vazios ou uma concatenação encadeada do símbolo contido em R_1. Na Figura 7 é apresentada uma representação dessa operação.

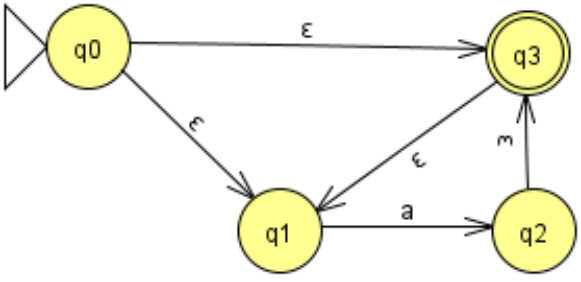

Figura 7. Fecho estrela em *a*.

Perceba que nesta operação há a leitura vazia (ε) ou de um ou mais símbolos *a*.

Após conhecidos os casos-base e os passos de indução, vejamos um exemplo de sua aplicação.

Considere a expressão regular ***(ab*) + a***. Para construirmos um autômato para esta expressão, devemos inicialmente criar um autômato finito que represente as expressões *a* e *b*, conforme apresentado anteriormente na Figura 4.

Posteriormente, como sabemos, o operador estrela possui precedência mais alta, assim, seguindo o caso 6, aplicamos a operação estrela em *b*, tendo como resultado o disposto na Figura 8.

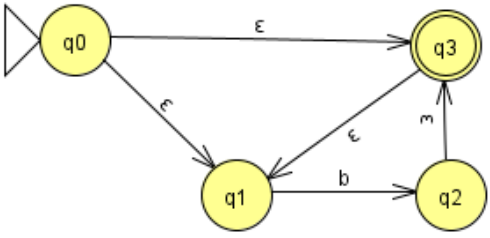

Figura 8. Fecho estrela em *b*.

Posteriormente aplicamos a operação de concatenação da expressão ***a*** com ***b****, formando a expressão ***ab****, dada pela Figura 9.

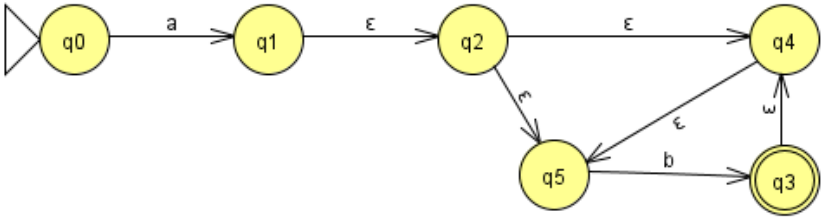

Figura 9. ab^*.

E, por fim, aplicamos a operação de união à expressão ***ab**** com ***a***, isto é, ***(ab***) + a***, representada na Figura 10.

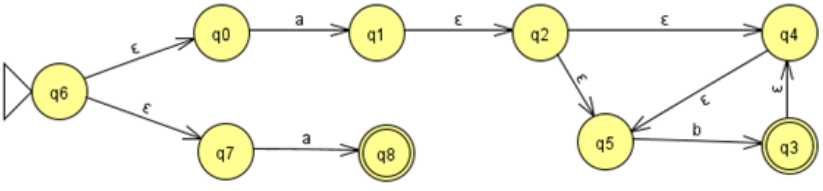

Figura 10. $(ab^*) + a$.

Conforme apresentado, as conversões de expressões regulares para autômatos finitos geram um AFNε, ou seja, não produzem um autômato finito

com o menor número de estados; para isto, após o processo de conversão, é possível converter o autômato resultante em um AFD ou AFN e, posteriormente, aplicá-lo a um processo de minimização de estados.

Convertendo autômatos finitos em expressões regulares

Conforme Sipser (2007), o processo de conversão de autômatos finitos para expressões regulares requer a aplicação de algumas transformações, de forma que, ao final, tenhamos apenas dois estados: um estado inicial e um estado de aceitação. Ao final, a transição desse autômato conterá a expressão regular referente ao autômato inicial.

Para esta conversão, utilizaremos o método por eliminação de estados. Antes de começarmos a utilizar esse método, vejamos alguns casos que precisamos conhecer para realizar o processo de conversão.

Caso 1

Se um autômato finito possuir alguma transição com destino ao estado inicial, este estado não deverá ser mais o inicial; deste modo, será criado outro estado, que possuirá uma transição ao estado inicial anterior, a partir de uma transição vazia. Essa transformação é apresentada na Figura 11.

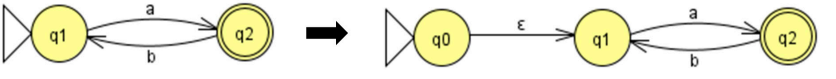

Figura 11. Inserção de um estado e alteração do estado inicial.

Caso 2

Se existir um estado final no qual existe uma transição saindo dele, deverá ser criado outro estado final, substituindo o anterior, com uma transição do estado final anterior para ele. Analise essa transformação na Figura 12.

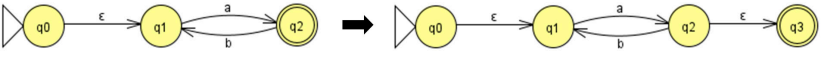

Figura 12. Inserção de um estado e alteração do estado final.

Caso 3

Se existir um autômato finito com mais de um estado final, estes não deverão ser considerados estados finais; assim, outro estado de aceitação será criado, e os estados finais anteriores possuirão uma transição vazia até ele. Essa transformação é ilustrada na Figura 13.

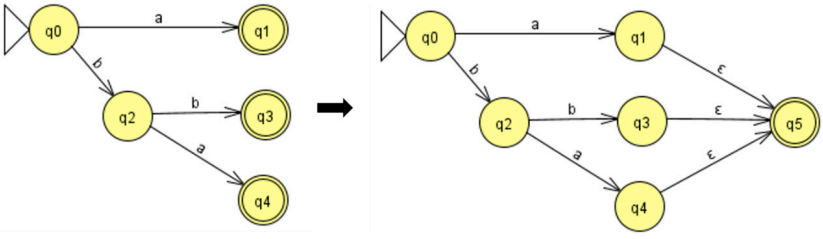

Figura 13. Inserção de um estado e alteração no conjunto de estados finais.

Caso 4

Por fim, o procedimento de conversão consiste em remover todos os estados, com exceção dos estados inicial e final, de modo que a linguagem produzida se mantenha a mesma.

Vejamos um exemplo apresentado na Figura 14.

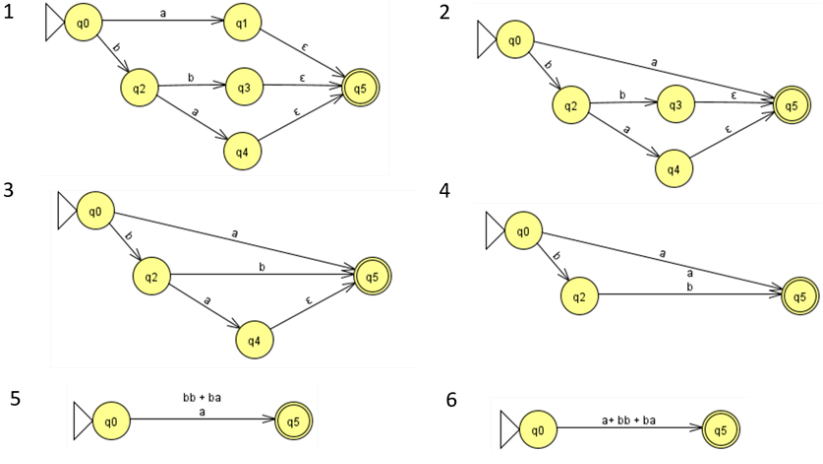

Figura 14. Transformação de um autômato finito para uma expressão regular.

Ao analisar a Figura 14, perceba que a conversão consiste em seis etapas.

- Etapa 1: temos o autômato original.
- Etapa 2: retiramos o estado q1, e a transição do símbolo *a* passa a ser direcionada ao estado final, q5.
- Etapa 3: removemos o estado q3, e a transição de q2, com o símbolo *b*, passa a ser direcionada ao estado q5.
- Etapa 4: retiramos o estado q4, e a transição de q2, com o símbolo *a*, passa a ser direcionada ao estado q5.
- Etapa 5: removemos o estado q2, isto é, temos uma união de ***bb*** com ***ba***, representada por ***bb + ba***.
- Etapa 6: por fim, unimos a transição ***a*** com a transição ***bb + ba***, finalizando a construção da expressão regular com ***a + bb + ba***.

Vejamos agora uma resolução, apresentada na Figura 15, que possui as operações de união, concatenação e fecho de Kleene.

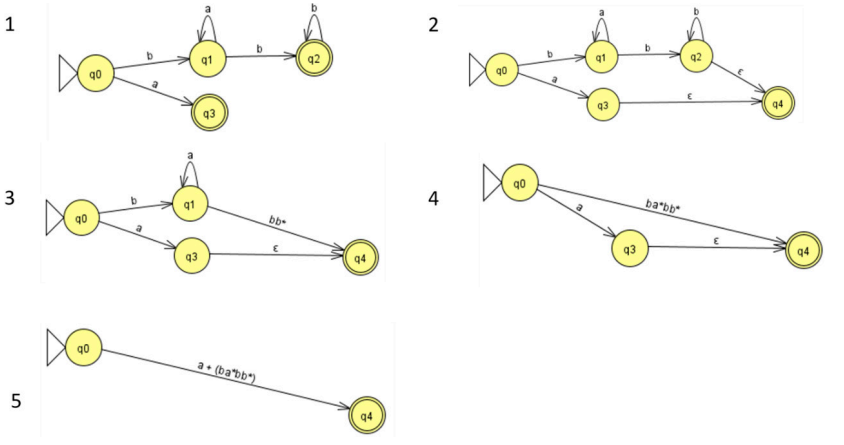

Figura 15. Transformação de um autômato finito para uma expressão regular.

Vejamos nas cinco etapas a seguir o processo de conversão apresentado na Figura 15.

- Etapa 1: o autômato é apresentado.

- Etapa 2: verificou-se que o autômato possui dois estados de aceitação, sendo que um deles (q2) possui transição para si mesmo. Deste modo, é introduzido outro estado de aceitação q4 e retirada a função de aceitação dos anteriores (q2 e q3).
- Etapa 3: é excluído o estado q2, assim, temos uma concatenação de ***b*** com ***b****, elaborando a expressão ***bb****.
- Etapa 4: é retirado o estado q1, neste sentido, as transições referentes a ***b***, ***a**** e ***bb**** são concatenadas, isto é ***ba*bb****.
- Etapa 5: por fim, é retirado o estado q3, gerando a expressão ***a* + (*ba*bb****)**.

Neste capítulo você aprendeu um pouco sobre expressões regulares, exemplos de aplicação, sua equivalência com autômatos finitos e como fazer a conversão de expressões regulares para autômatos finitos e vice-versa.

Referências

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. *Introdução à teoria de autômatos, linguagens e computação*. Rio de Janeiro: Campus; Elsevier, 2003. 560 p.

JARGAS, A. M. *Expressões regulares*: uma abordagem divertida. 5. ed. São Paulo: Novatec, 2016. 248 p.

LIMA, E. S. *Expressões regulares*: conceitos, usos e aplicações na web. Rio Branco: Edição do autor, 2020. 74 p. (Série Aprenda Sozinho, 1).

MENEZES, P. B. *Linguagens formais e autômatos*. 6. ed. Porto Alegre: Bookman, 2011. 256 p. (Série Livros Didáticos Informática UFRGS, 3).

SIPSER, M. *Introdução à teoria da computação*. São Paulo: Cengage Learning, 2007. 459 p.

STUBBLEBINE, T. *Regular expression pocket reference*: regular expressions for Perl, Ruby, PHP, Python, C, Java and .NET. 2. ed. Sebastopol: O'Reilly, 2007. 117 p.

THOMPSON, K. Programming techniques: regular expression search algorithm. *Communications of the ACM*, New York, v. 11, n. 6, p. 419–422, June 1968.

Leituras recomendadas

CRITCHLOW, C.; ECK, D. J. Regular expressions. *In*: CRITCHLOW, C.; ECK, D. J. *Foundations of computation*. Davis: LibreTexts; University of California, 2020. p. 100–101. Disponível em: https://eng.libretexts.org/Bookshelves/Computer_Science/Book%3A_Foundations_of_Computation_(Critchlow_and_Eck)/03%3A_Regular_Expressions_and_FSA's/3.02%3A_Regular_Expressions. Acesso em: 28 jan. 2021.

FITZGERALD, M. *Introdução às expressões regulares*: desvendando as expressões regulares, passo a passo. São Paulo: Novatec; O'Reilly, 2012. 160 p.

GOYVAERTS, J.; LEVITHAN, S. *Regular expressions cookbook*: detailed solutions in eight programming languages. 2. ed. Sebastopol: O'Reilly, 2012. 612 p.

MENEZES, P. B. Linguagens regulares: expressões regulares. *In*: MENEZES, P. B. *Linguagens formais e autômatos*. 6. ed. Porto Alegre: Bookman, 2011. p. 93–106.

Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS