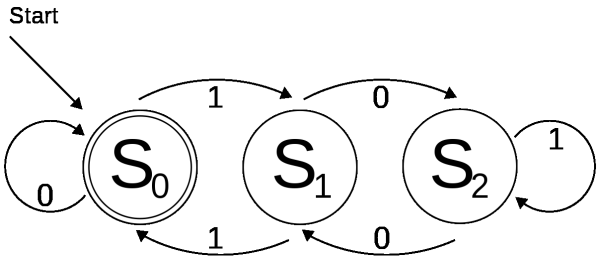
Autômatos finitos determinísticos

Predefinição:Use dmy dates



An example of a deterministic finite automaton that accepts only binary numbers that are multiples of 3. The state S_0 is both the start state and an accept state.

Na Teoria dos autômatos, um sub-tópico da Ciência da computação teórica, um **autômato finito determinístico** — também chamado **máquina de estados finita determinística** (**AFD**) — é uma Máquina de estados finita que aceita ou rejeita cadeias de símbolos gerando um único ramo de computação para cada cadeia de entrada.[1] 'Deterministica' refere-se à unicidade do processamento. O primeiro conceito similar ao de autômatos finitos foi apresentado por McCulloch e Pitts em 1943.[2][3] Modelo esse que foi produzido na busca por estruturas mais simples para a reprodução de máquinas de estado finitas.

A figura à direita representa um autômato finito determinístico através de um Diagrama de transição de estados. Nesse autômato há três estados: S0, S1 e S2 (representados graficamente por círculos). A entrada é constituída por uma sequência finita de caracteres 1s e 0s. Para cada estado da máquina, existe um arco de transição levando a um outro estado para ambos caracteres 0 e 1. Isso significa que, em um dado estado, após a leitura de cada símbolo a máquina determinística transita para um único estado referente à aresta associada ao símbolo. Por exemplo, esteja o autômato atualmente no estado S0 e o símbolo de entrada para aquela instância um '1', então ele salta deterministicamente para o estado S1. Todo AFD possui um *estado inicial* (denotado graficamente por uma seta de origem anônima) onde a sua computação começa e um conjunto de *estados de aceitação* (denotados graficamente por um círculo de borda dupla) o qual indica a aceitação da cadeia de entrada.

Um Autômato finito determinístico é normalmente definido como um conceito matemático abstrato, mas devido à seu fator determinístico, ele pode ser implementado através de Hardware e Software para resolver diversos problemas específicos. Por instância, DFAs são utilizados para modelar softwares que validam entradas de usuário tal como o seu e-mail em um servidor de correio eletrônico.

AFDs reconhecem exatamente o conjunto de Linguagens Regulares que são, dentre outras coisas, úteis para a realização de Análise lexica e reconhecimento de padrões. Uma AFD pode ser construído a partir de um Autômato finito não determinístico através de uma Construção do conjunto das partes.

Definição Formal

Um **Autômato Finito Determinístico A** é uma 5-tuple, $(Q, \Sigma, \delta, q_0, F)$ consistindo de:

- um conjunto finito de estados (Q)
- um conjunto finito de símbolos de entrada chamado Alfabeto (Σ)
- uma função de transição ($\delta : Q \times \Sigma \to Q$)
- um estado inicial ($q_0 \in Q$) e
- um conjunto de estados de aceitação ($F \subseteq Q$)

Seja $w = a_1 a_2 \ldots a_n$ uma cadeia de símbolos sobre o alfabeto Σ, O autômato M aceita a cadeia w se somente se existe uma sequência de estados, $r_0, r_1, \ldots, r_n$, em Q com as seguintes condições:

1. $r_0 = q_0$
2. $r_{i+1} = \delta(r_i, a_{i+1})$, para $i = 0, \ldots, n-1$
3. $r_n \in F$.

Em outras palavras, a primeira condição afirma que a máquina se inicia no estado inicial q_0. A segunda condição diz que, dado cada símbolo da entrada *w*, a máquina transita de estado em estadoo de acordo com a função de transição δ. A terceira e última condição diz que a máquina aceita *w* se somente se o último símbolo da entrada leva o autômato a parar em um estado *f* tal que f ∈ *F*. Caso contrário, diz-se que a máquina *rejeita* a entrada. O conjunto de cadeias que *M* aceita é chamado Linguagem *reconhecida* por *M* e é simbolicamente representado por *L(M)*.

Um autômato finito determinístico que não possui estado inicial ou estados de aceitação é conhecido como um *Sistema de Transições* ou *Semiautômato*.

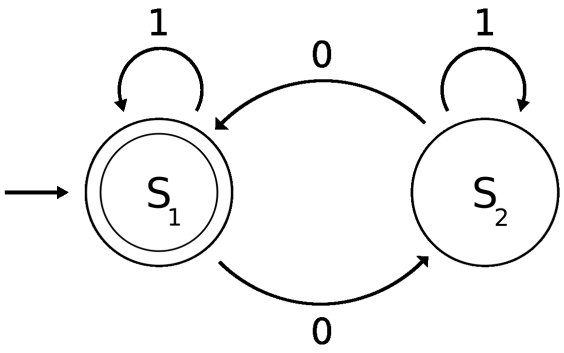
Para uma melhor compreensão introdutória à definição formal de autômatos, veja Teoria dos autômatos.

Exemplo

O exemplo a seguir representa um AFD *M*, com um alfabeto binário, que requer que sua entrada contenha um número par de 0s.

$M = (Q, \Sigma, \delta, q_0, F)$ onde

- $Q = \{S_1, S_2\}$,
- $\Sigma = \{0, 1\}$,
- $q_0 = S_1$,
- $F = \{S_1\}$, e
- δ é definido da pela seguinte Tabela de transição de estados:



The state diagram for *M*

	0	**1**
S_1	S_2	S_1
S_2	S_1	S_2

O estado S_1 indica que houve um número par de ocorrências do símbolo 0 até então na entrada, enquanto que S_2 significa que esse valor foi impar. Um 1 na entrada não altera o estado atual do autômato. Quando a entrada é completamente lida, o estado atual vai indicar se a cadeia de entrada possui um número par de 0s ou não. Caso a cadeia possua de fato uma quantidade par, *M* vai encerrar-se no estado S_1, um estado de aceitação, tal que a entrada vai ser aceita. Caso contrário, se o autômato encontrar-se no estado S_2, a máquina rejeita a cadeia.

A linguagem reconhecida por *M* é a Linguagem regular dada pela Expressão regular 1*(0 (1*) 0 (1*))*, tal que "*" é uma Kleene star, e.g., 1* indica que há uma ocorrência não-negativa de 1s nesse trecho da expressão.

Propriedades de fechamento

AFDs são fechadas sob as seguintes operações (o que implica que se uma linguagem for obtida através da aplicação de uma dessas funções sobre uma ou mais linguagens, existe uma AFD que reconhece a linguagem resultante):

- União
- Interceção
- Concatenação
- Negação
- Estrela
- Quociente
- Substituição
- Homomorfismo

Sendo AFDs equivalentes a Autômatos finitos não determinísticos (AFN), essas operações de fechamento podem ser provadas através do uso das propriedades dos AFNs.

Modos de Geração e Aceitação

Uma AFD que representa uma dada linguagem regular pode ser usada tanto em um modo de aceitação para validar que uma cadeia de entrada pertence a uma linguagem L como em Modo de geração para gerar uma lista de todas as linguagens que pertencem a L.

No modo de aceitação, o autômato processa uma cadeia de entrada do caracter mais à esquerda ao mais à direita, lendo um símbolo por vez. A computação da entrada se inicia no *estado inicial* e continua ao lerse o primeiro símbolo da cadeia de entrada, seguindo o estado de transição que corresponde àquele simbolo (de acordo com a função de transição do autômato). O sistema segue as transições até que todos os símbolos da entrada tenham sido lidos, o que indica o final da computação. Se após o processamento da entrada a máquina encontrar-se em um estado de aceitação, a cadeia é aceita e diz-se que ela pertence a L. Ao contrário, a cadeia não pertence a L e diz-se que ela foi rejeitada pelo autômato.

O modo de geração (criação) é similar, exceto que, em vez de validar uma cadeia de entrada, seu objetivo é produzir uma lista de todas as cadeias de caracteres dentro da linguagem. Em vez de seguir uma única transição de cada estado, o modo de criação segue todas elas. Na prática, isso pode ser alcançado por um paralelismo massivo (fazer com que o programa se ramifique em dois ou mais processos cada vez que este se deparar com uma decisão) ou então por Recursão. Como antes, a computação começa no estado inicial e procede a seguir cada transição disponível, sem perder de vista que processos foram tomados. Toda vez que o autômato se encontra em um estado de aceitação este sabe que a sequência de processos tomada forma uma cadeia de caracteres válida na linguagem e adiciona essa cadeia na lista que o programa está gerando. Se a linguagem que o automato descreve é infinita (exemplo: contém um número infinito de cadeias, como: “todas as cadeias binárias com um número ímpar de zeros”), então a computação nunca irá parar. Sabendo-se que linguagens regulares, em geral, são infinitas, autômatos no modo de criação tendem a ser muitas vezes uma construção teórica.

Equivalência com Expressões Regulares

Expressões regulares e autômatos finitos determinísticos (AFD) são equivalentes em poder de descrição. Ambas representações descrevem exatamente a classe de linguagens regulares. É possível, a partir de um autômato finito determinístico, obter a expressão regular que o autômato descreve.

De forma a provar a equivalência entre AFDs e expressões regulares, é preciso demonstrar que é possível converter um AFD em uma expressão regular e que também é possível converter uma expressão regular em um AFD.

O procedimento que converte autômatos finitos determinísticos utiliza outro tipo de autômato, denominado autômato finito não determinístico generalizado. O procedimento é constituído em duas etapas, a conversão de autômato finito determinístico para um autômato finito não determinístico generalizado (AFNG), e enfim, de um AFNG para uma expressão regular. Um AFNG é extremamente semelhante a um autômato finito não determinístico (AFN), no entanto, a diferença consiste principalmente em que o AFNG é capaz de ler **blocos de símbolos de entrada**, ao invés de apenas um único símbolo de entrada. A função de transição de um AFNG caracteriza-se da seguinte forma:

$\delta : (Q - \{q_{aceita}\}) \times (Q - \{q_{inicio}\}) \rightarrow R$

De tal forma que:

- Q é o conjunto de estados;
- q_{aceita}, q_{inicio} são estados de aceitação e iniciais, respectivamente
- R é a coleção de todas as expressões regulares sobre o alfabeto $\sum$do AFNG.

Conversão de AFD para expressão regular

O procedimento de conversão de um AFD em uma expressão regular consiste em duas etapas; 1. converter o AFD em um AFNG, e em seguida; 2. converter o AFNG em uma expressão regular.

```
Conversão AFD -> AFNG


Adicione um novo estado inicial com uma transição epsilon para o estado inicial original
Adicione um novo estado de aceitação
Para cada estado de aceitação antigo:
     Adicione uma transição epsilon desse estado ao novo estado de aceitação
Se quaisquer setas têm múltiplos rótulos
     Substitua cada uma dessas setas por uma única seta cujo rótulo é a união dos rótulos anteriores
Se não há setas entre qualquer par de estados
     Adicione uma seta com rótulo vazio entre esse par de estados
```

Esse procedimento transforma um AFD de q estados em um AFNG de $q + 2$estados, de forma que a mesma linguagem L é descrita por ambos autômatos, sendo assim os AFDs e AFNGs equivalentes em poder de descrição. Uma vez que o AFNG encontra-se nesse formato, podemos prosseguir com o procedimento recursivo que converte um AFNG em uma expressão regular. [4]

Conversão de expressão regular para AFD

Para concluir a prova de que expressões regulares e AFDs descrevem a mesma classe de linguagens (as linguagens regulares), uma vez mostrado que um AFD pode ser convertido em uma expressão regular, agora é mostrado que uma expressão regular pode ser convertida em um AFD. Fazemos isso convertendo uma expressão regular qualquer para um autômato finito não determinístico (AFN), uma vez que esse tipo de autômato também reconhece a classe de linguagens regulares (e por sua vez, descreve a mesma classe que os AFDs).

Seja uma expressão regular R que descreve uma determinada linguagem L. Descrevemos seis casos a serem tratados na conversão para um autômato determinístico.

Caso 1: $R = a$ para algum $a \in \Sigma$.

Então, $L(R) = a$. Descrevemos formalmente o seguinte AFN N tal que $L(N) = R$:

$N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, de forma que a função de transição é descrita com as transições: $\delta(q_1, a) = \{q_2\}$ e $\delta(r, b) = \emptyset$ para $r \neq q_1$ ou $b \neq a$.

Caso 2: $R = \epsilon$

Então, $L(R) = \{\epsilon\}$. Seja então a descrição formal do AFN N que reconhece essa expressão regular:

$N = (\{q_1\}, \Sigma, \delta, q_1, \{q_2\})$ tal que $\delta(r, b) = \emptyset$ para qualquer r, b.

Caso 3: $R = \emptyset$.

Então, $L(R) = \emptyset$. Seja então a descrição formal do AFN N que reconhece essa expressão regular:

$N = (\{q_1\}, \Sigma, \delta, q_1, \emptyset)$ tal que $\delta(r, b) = \emptyset$ para qualquer r, b.

Caso 4: $R = R_1 \cup R_2$

Sejam os AFN $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ que reconhecem respectivamente as linguagens descritas por R_1 e R_2. O seguinte AFN $N = (Q, \Sigma, \delta, q_0, F)$ reconhece $R_1 \cup R_2$:

- $Q = \{q_0\} \cup Q_1 \cup Q_2$
- $F = F_1 \cup F_2$
- $$\delta = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \wedge a = \epsilon \\ \emptyset & q = q_0 \wedge a \neq \epsilon \end{cases}$$

Os estados de N são compostos de todos os estados que estão contidos no conjunto de estados de N_2 ou N_1 em adição de um novo estado inicial denominado q_0. Além disso, os estados de aceitação de N são todos os estados de aceitação de N_2 ou N_1, de forma que N aceita uma cadeia que pertença tanto à linguagem descrita por N_1 como N_2. Essa construção também é a prova de que as linguagens regulares são fechadas sob a operação de união.

Caso 5: $R = R_1 \circ R_2$

Sejam os AFN $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ que reconhecem respectivamente as linguagens descritas por R_1 e R_2. O seguinte AFN $N = (Q, \Sigma, \delta, q_1, F_2)$ reconhece $R_1 \circ R_2$:

- $Q = Q_1 \cup Q_2$
- $$\delta = \begin{cases} \delta_1(q, a) & q \in Q_1 \wedge q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \wedge a \neq \epsilon \\ \delta_1(q, a) \cup \{q_2\} & q \in F_1 \wedge a = \epsilon \\ \delta_2(q, a) & q \in Q_2 \end{cases}$$

O estado inicial de N é o estado inicial de N_1. Os estados de N são compostos de todos os estados que estão contidos no conjunto de estados de N_2 ou N_1. Além disso, os estados de aceitação de N são apenas os estados de aceitação de N_2. Esse autômato construído aceita as cadeias que possam ser divididas em duas partes de forma que a primeira parte seja aceita por N_1 e a segunda por N_2. Essa construção também é a prova de que as linguagens regulares são fechadas sob a operação de concatenação.

Caso 6: $R = R_1^*$

Seja o AFN $N_1 = (Q_1, \Sigma, \delta_1, q_1 = 0, F_1)$ que reconhece a linguagem descrita por R_1. O seguinte AFN $N = (Q, \Sigma, \delta, q, F)$ reconhece R_1^*:

- $Q = \{q_0\} \cup Q_1$
- $F = F_1 \cup \{q_0\}$

- $\delta = \begin{cases} \delta_1(q,a) & q \notin F_1 \wedge q \in Q_1 \\ \delta_1(q,a) & q \in F_1 \wedge a \neq \epsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \wedge a = \epsilon \\ \{q_1\} & q = q_0 \wedge a = \epsilon \\ \emptyset & q = q_0 \wedge a \neq \epsilon \end{cases}$

Os estados de aceitação de N são todos os estados de aceitação de N_1 com adição do novo estado inicial q_0. Essa construção também é a prova de que as linguagens regulares são fechadas sob a operação do fecho de Kleene (ou estrela).

Uma vez que é possível construir um autômato finito não determinístico a partir de uma determinada expressão regular, é possível convertê-lo em um autômato finito determinístico correspondente, de forma que a expressão regular é convertida num AFD. Dessa forma é concluída a prova de que expressões regulares e AFDs são equivalentes em poder de descrição uma vez que é possível obter a expressão regular a partir do AFD e vice-versa.

Vantagens e Desvantagens

AFDs são equivalentes em poder de expressão a Autômatos finitos não determinísticos (NFDs). Isso se dá porque, primeiramente qualquer AFD é também um AFN, então AFNs podem expressar o que AFDs expressam. Além disso, dado um AFN, através de uma Construção do conjunto das partes é possível construir um AFD que reconhece a mesma linguagem que um determinado AFN, apesar de o AFD poder precisar de um número muito maior de estados que o AFN (mais precisamente, o número de estados de um AFD construído a partir de um AFN equivale ao conjunto das partes dos estados do AFN que o gerou).

Por outro lado, autômatos de estado finito de potência são estritamente limitados nas linguagens que podem reconhecer. Muitas linguagens simples, incluindo qualquer problema que requeira mais que espaço constante para resolver, não podem ser reconhecidos por um AFD. O exemplo clássico de uma linguagem simplesmente descrita que nenhum AFD reconhece é a linguagem de colchetes, ou seja, linguagem que consiste em colchetes devidamente emparelhados como a palavra "(()())". Nenhum DFA pode reconhecer a linguagem de colchetes porque não há um limite para a recursividade, ou seja, sempre se pode incorporar outro par de suportes dentro. Seria necessária uma quantidade infinita de estados para reconhecê-la. Outro exemplo mais simples é a linguagem consistindo de cadeias de símbolos da forma $a^n b^n$— um número finito de a's, seguido por um número igual de b's.

Veja também

- Autômato finito determinístico acíclico
- Autômato finito não deterministico
- Lógica de segunda ordem
- Turing machine
- Autômato com pilha
- Linguagem Regular
- Expressão regular

Notas

[1] Hopcroft 2001:
[2] McCulloch and Pitts (1943):
[3] Rabin and Scott (1959):
[4] https://pt.wikipedia.org/wiki/Aut%C3%B4mato_finito_n%C3%A3o_determin%C3%ADstico_generalizado#Convers.C3.A3o_de_AFNG_para_express.C3.A3o_regular

Referências

- Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D.. In: John E.. *Introduction to Automata Theory, Languages, and Computation* (http://www.pearsonhighered.com/educator/product/Introduction-to-Automata-Theory-Languages-and-Computation/9780201441246.page). 2 ed. [S.l.]: Addison Wesley, 2001. ISBN 0-201-44124-1 Página visitada em 19 November 2012.
- Lawson, Mark V.. *Finite automata*. [S.l.]: Chapman and Hall/CRC, 2004. ISBN 1-58488-255-7
- (1943) "A logical calculus of the ideas imminent in nervous activity". *Bulletin of Mathematical Biophysics*: 541–544.
- (1959) "Finite automata and their decision problems.". *IBM J. Res. Develop.*: 114–125.
- Sakarovitch, Jacques. *Elements of automata theory*. Cambridge: Cambridge University Press, 2009. ISBN 978-0-521-84425-3
- Sipser, Michael. *Introduction to the Theory of Computation*. Boston: PWS, 1997. ISBN 0-534-94728-X. Section 1.1: Finite Automata, pp. 31–47. Subsection "Decidable Problems Concerning Regular Languages" of section 4.1: Decidable Languages, pp. 152–155.4.4 DFA can accept only regular language

Links Externos

- DFA Simulator - an open source graphical editor and simulator of DFA (http://home.arcor.de/kai.w1986/dfasimulator/)

Predefinição:Formal languages and grammars

Fontes e Editores da Página

Autômatos finitos determinísticos *Fonte*: http://pt.wikipedia.org/w/index.php?oldid=38338312 *Contribuidores*: Diego Queiroz, Gpp40, LeonardoG, Mgtmc, Mobyduck, Njr, Olavom, SallesNeto BR, Santana-freitas, 14 edições anónimas

Fontes, Licenças e Editores da Imagem

File:DFA example multiplies of 3.svg *Fonte*: http://pt.wikipedia.org/w/index.php?title=Ficheiro:DFA_example_multiplies_of_3.svg *Licença*: Public Domain *Contribuidores*: Self-made

File:DFAexample.svg *Fonte*: http://pt.wikipedia.org/w/index.php?title=Ficheiro:DFAexample.svg *Licença*: Public Domain *Contribuidores*: Cepheus

Licença