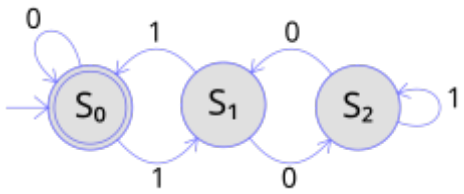
Introdução

A teoria dos autômatos é uma área fundamental da ciência da computação teórica e da matemática computacional, sendo base para o entendimento de linguagens formais, compiladores e reconhecimento de padrões. Neste trabalho, abordaremos dois dos modelos mais importantes: o Autômato Finito Determinístico (AFD) e o Autômato Finito Não Determinístico (AFND). Ambos são utilizados para modelar sistemas de reconhecimento de linguagem e possuem características que os tornam ferramentas essenciais para o estudo de linguagens regulares. Neste trabalho, exploraremos suas definições formais, funcionamento, aplicações práticas e faremos uma comparação entre os dois modelos, além de propor exercícios para fixação dos conceitos.

Autômato Finito Determinístico (AFD)

Um Autômato Finito Determinístico (AFD) é um modelo matemático de computação que consiste em um conjunto finito de estados, um alfabeto de entrada, uma função de transição, um estado inicial e um conjunto de estados de aceitação. Cada entrada leva a um único estado sucessor.

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] "Determinística" 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.



Um exemplo de autômato finito determinístico que aceita apenas números binários múltiplos de 3. O estado *S*0 é tanto o estado de início quanto um estado de aceitação.

A figura à cima 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, AFDs 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[1] que são, dentre outras coisas, úteis para a realização de Análise léxica 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.

Um Autômato Finito Determinístico A é uma 5-tuple (ou quíntupla), (Q, Σ, δ, q_0, F) consistindo de:

um conjunto finito de símbolos de entrada chamado Alfabeto (Σ)

- um conjunto finito de estados (Q)
- uma função de transição (δ : $Q \times \Sigma \to Q$)
- 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 estado 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*.

Exemplos:

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:

	0	
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 "zero ou mais" ocorrências de 1s nesse trecho da expressão.

Máquina de estados finitos determinística

Uma **máquina de estados finita determinística** ou **autômato finito determinístico (AFD)** é uma máquina de estados finitos onde, dado uma configuração e um símbolo da cadeia de entrada existe somente um estado para o qual a máquina pode transitar. Em outras palavras, em um AFD, o estado sucessor é definido unicamente pelo seu estado atual e pela condição de transição.

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
- Interseção
- 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 caractere mais à esquerda ao mais à direita, lendo um símbolo por vez. A computação da entrada se inicia no *estado inicial* e continua ao ler-se o primeiro símbolo da cadeia de entrada, seguindo o estado de transição que corresponde àquele símbolo (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 maciço (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:

De tal forma que:

- é o conjunto de estados;
- são estados de aceitação e iniciais, respectivamente
- é a coleção de todas as expressões regulares sobre o alfabeto 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.

Esse procedimento transforma um AFD de estados em um AFNG de 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.

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 que descreve uma determinada linguagem . Descrevemos seis casos a serem tratados na conversão para um autômato determinístico.

Caso 1: para algum .

Então, . Descrevemos formalmente o seguinte AFN tal que :

, de forma que a função de transição é descrita com as transições: e para ou .

Caso 2:

Então, . Seja então a descrição formal do AFN que reconhece essa expressão regular:

tal que para qualquer .

Caso 3: .

Então, . Seja então a descrição formal do AFN que reconhece essa expressão regular:

tal que para qualquer .

Caso 4:

Sejam os AFN e que reconhecem respectivamente as linguagens descritas por e . O seguinte AFN reconhece :

-
-
-

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

Caso 5:

Sejam os AFN e que reconhecem respectivamente as linguagens descritas por e . O seguinte AFN reconhece :

-
-

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

Caso 6:

Seja o AFN que reconhece a linguagem descrita por . O seguinte AFN reconhece :

-
-
-

Os estados de aceitação de são todos os estados de aceitação de com adição do novo estado inicial . 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.

Automato finito não deterministico

Na teoria da computação, uma **máquina de estados finita não-determinística** ou um **autômato finito não-determinístico** (**AFND**) é uma máquina de estados finita onde para cada par de estado e símbolo de entrada pode haver vários próximos estados possíveis. Isso o distingue do autômato finito determinístico (AFD), onde o próximo estado possível é univocamente determinado. Embora AFD e AFND possuam definições distintas, pode ser mostrado na teoria formal que eles são equivalentes e, deste modo, para qualquer AFND dado, pode-se construir um AFD equivalente e vice-versa: essa é a construção do conjunto das partes. Ambos os tipos de autômatos reconhecem apenas linguagens regulares. Máquinas de estados finitas não-determinísticas são às vezes estudadas com o nome de subshifts de tipo finito.

Máquinas de estados finitas não-determinísticas são generalizadas pelo autômato probabilístico, o qual atribui uma probabilidade para cada transição de estado, e por várias outras maneiras, tais como autômatos com pilha e AFNs com transições ε.

Autômatos finitos não-determinísticos foram introduzidos em 1959 por Michael O. Rabin e Dana Scott,[1] que também mostraram sua equivalência com autômatos finitos determinísticos.

Um AFND, similarmente a um AFD, lê uma cadeia de símbolos de entrada. Para cada símbolo da entrada há uma transição para um novo estado, até que todos os símbolos de entrada sejam lidos.

Ao contrário de um AFD, há um não-determinismo onde, para cada símbolo de entrada, seu próximo estado pode ser um de dois ou mais estados possíveis. Assim, na definição formal, o próximo estado é um elemento do conjunto das partes dos estados. Este elemento, um conjunto, representa algum subconjunto de todos os possíveis estados que podem ser considerados.

Uma extensão do AFND é o **AFND-lambda** (também conhecido como **AFND-epsilon** ou como **AFND com transições epsilon**), o qual permite uma transição para um novo estado sem ler nenhum símbolo de entrada. Por exemplo, se está no estado 1, com o próximo símbolo de entrada um *a*, pode-se se mover para o estado 2 sem ler nenhum símbolo de entrada. Assim, tem-se uma ambiguidade: o sistema está no estado 1 ou no estado 2 antes de ler a entrada *a*? Por causa desta ambiguidade, é mais conveniente falar de um conjunto de possíveis estados que o sistema pode estar. Assim, antes de ler um símbolo de entrada *a*, o AFND-epsilon pode estar em qualquer um dos estados do conjunto {1,2}. Equivalentemente, pode-se imaginar que o AFND está nos estados 1 e 2 'ao mesmo tempo': isso dá uma ideia informal da construção do conjunto das partes: o AFD equivalente a um AFND é definido como aquele que está no estado *q={1,2}. Transições para novo estados sem ler símbolos de entrada são chamadas de* ***transições lambda*** *ou* ***transições epsilon****. Elas são comumente rotuladas com as letras gregas λ ou ε.*

A noção de aceitação de uma entrada é similar àquela em um AFD. Quando o último símbolo de entrada é lido, diz-se que o AFND aceita se e somente se existe *algum* conjunto de transições que irá levá-lo a um estado de aceitação. Do mesmo modo, ele rejeita, se, não importando quais transições sejam aplicadas, ele não terminar num estado de aceitação.

Definição formal

Dois tipos similares de AFNDs são comumente definidos: o AFND e o AFND com transições ε. O tipo comum é definido como uma 5-tupla, , consistindo de

- um conjunto finito de estados ;
- um conjunto finito de símbolos de entrada , denominando o *alfabeto*;
- uma função de transição ;
- um estado *inicial* ;
- um conjunto distinto de estados como *estados de aceitação* (ou *estados finais*), .

Aqui, denota o conjunto das partes de

O AFND com transições ε (também conhecido como *AFN-epsilon* ou *AFN-lambda*) é definido por uma 5-upla extremamente semelhante, porém, a função de transição é substituída por uma que permite a cadeia vazia ε como uma entrada possível. Desse modo, sua função de transição é dada por:

Pode-se mostrar que AFND comum e um AFND com transições ε são equivalentes e, assim, dado qualquer AFND comum, pode-se construir um AFND com transições ε que reconhece a mesma linguagem e vice-versa.

Propriedades dos AFNDs

A máquina começa no estado inicial especificado e lê uma cadeia de símbolos do seu alfabeto. O autômato usa a função de transição de estado para determinar o próximo estado usando o estado atual e o símbolo que acabou de ser lido, o que inclui a cadeia vazia, no caso dos AFNDs com transições ε. Entretanto, "o próximo estado de um AFND não depende somente do evento atual, mas também de um número arbitrário de eventos subsequentes de entradas. Até esses eventos subsequentes ocorrerem, não é possível determinar onde a máquina de estados está".[2]

O conjunto de todas as cadeias que são aceitas por um AFND é a linguagem que o AFND aceita. Esta linguagem deve ser uma linguagem regular.

Para todo AFND pode ser encontrada um AFD que aceite a mesma linguagem. Portanto, é possível converter um AFND existente em um AFD com o propósito de implementar (talvez) uma máquina mais simples. Isto pode ser realizado usando a construção do conjunto das partes, o que pode levar a um aumento exponencial no número de estados necessários. Uma prova formal desta construção do conjunto das partes pode ser encontrada aqui.

Muitas vezes, pode ser mais fácil construir um AFND do que um AFD equivalente, tanto pelo entendimento como pelo tamanho da construção. AFNDs são comumente introduzidos no início de cursos de informática teórica para desenvolver a ideia de não-determinismo em modelos computacionais mais poderosos, como as máquina de Turing.

Definição formal de computação dos AFNDs

Seja um AFND e uma cadeia do alfabeto.

Diz-se que **aceita** se podemos escrever como onde cada e existe uma sequência de estados em seguindo as três condições a seguir:

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

Implementação

Existem várias maneiras de implementar um AFND:

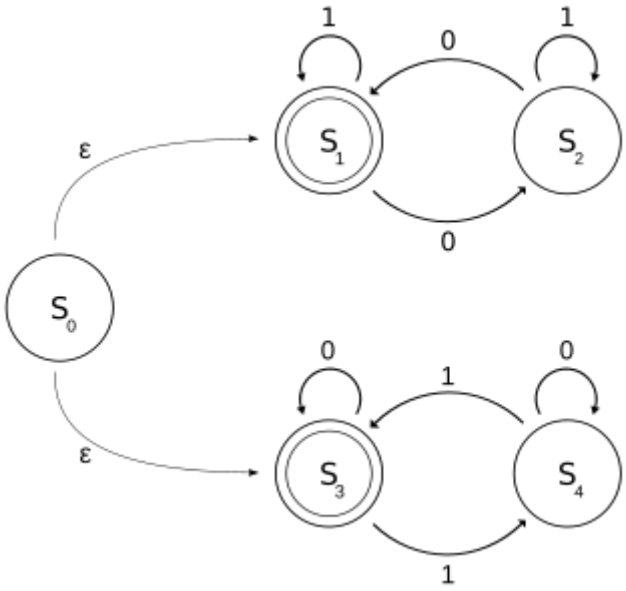
- Converter para o AFD equivalente. Em alguns casos isso pode causar um aumento exponencial no tamanho do autômato e, assim, espaço proporcional auxiliar ao número de estados do AFND (como o armazenamento do estado exige no máximo um bit para cada estado no AFND).
- Manter uma estrutura do conjunto de dados de todos os estados em que a máquina poderia estar. Na leitura do ultimo simbolo de entrada, se um desses estados é um estado final, a máquina aceita a cadeia. No pior dos casos, isso pode exigir espaço auxiliar proporcional ao número de estados do AFND; se a estrutura de conjunto usa um bit por estado do AFND, então esta solução é exatamente equivalente a anterior.
- Criar múltiplas cópias. Para cada n maneiras de decidir, o AFND cria até cópias da máquina. Durante sua computação, caso o AFND se depare com uma situação na qual existam múltiplas maneiras de se prosseguir, isto é, a função de transição leve a mais de um estado, pode-se considerar que o AFND divide-se em múltiplas cópias e cada uma dessas cópias fica responsável pela computação de uma das transições possíveis. Quando alguma cópia da máquina chega a uma situação na qual não há transição possível, diz-se que essa cópia do AFND "morre", não mais fazendo parte da computação. Se forma semelhante, caso **alguma** das cópias do AFND esteja em um estado de aceitação ao final da computação da cadeia de entrada, diz-se que o AFND **aceita** a dada cadeira. Isso também requer armazenamento linear com respeito ao número de estados do AFND, como também pode haver uma máquina para cada estado do AFND. Dessa forma, pode-se pensar no não-determinismo dos AFNDs como uma computação paralela, na qual o AFND (análogo a um processo) pode se *bifurcar* (de forma análoga à chamada de sistema *Fork*).
- Árvore de possibilidades. Derivada da ideia de *ramos de computação*, nessa forma, representamos a computação a partir de uma **árvore de possibilidades** cuja raiz corresponde ao início da computação. A cada símbolo de entrada lido, analisa-se a transição e se criam nós filhos adequadamente, isto é, caso haja 2 estados possíveis em uma dada transição, a árvore ramifica no nível atual e criam-se 2 novos filhos. Analogamente, se houver apenas uma transição possível, a árvore não é ramificada no dado momento e se não houver transição possível, o ramo "morre" e não é mais levado em conta na computação.
- Explicitamente propague tokens através da estrutura de transição do AFND e combine sempre que um token alcançar um estado final. Isto, às vezes, é útil quando o AFND pode codificar contextos adicionais sobre os eventos que desencadearam da transição. (Para uma implementação que usa esta técnica acompanhe as referências e olhe os Tracematches.[3])

Exemplo

- Exemplo 1: O seguinte exemplo explica um AFND *M*, com um alfabeto binário, que determina se uma entrada contém um número par de 0s ou um número par de 1s (Note que 0 ocorrências é um número par de ocorrências). Seja *M= (*Q, *Σ,* T, s0, F*) onde*
- Σ = {0, 1},
- *Q= {*s*0,* s*1,* s*2,* s*3,* s*4},*
- E({*s*0}) = { s0, s1, s3 }
- *F= {*s*1,* s*3}, and*
- A função de transição *T* pode ser definida por esta tabela de transição de estados:

	0	**1**	**ε**
S_0	{}	{}	{S_1, S_3}
S_1	{S_2}	{S_1}	{}
S_2	{S_1}	{S_2}	{}
S_3	{S_3}	{S_4}	{}
S_4	{S_4}	{S_3}	{}

O diagrama de estados para *M* é:



M pode ser visto como a união de dois AFDs: um com os estados {S_1, S_2} e o outro com os estados {S_3, S_4}.

Aplicação de AFND-ε

AFNDs e AFDs são equivalentes tal que se uma linguagem é reconhecida por um AFND, ela também é reconhecida por um AFD e vice versa. O estabelecimento de tal equivalência é importante e útil. É útil porque construir um AFND que reconhece uma dada linguagem é algumas vezes mais fácil do que construir um AFD para aquela linguagem. Isso é importante porque AFNDs

podem ser usadas para reduzir a complexidade do trabalho matemático necessário para estabelecer muitas propriedades importantes na teoria da computação. Por exemplo, é muito mais fácil provar as propriedades que seguem usando AFNDs do que AFDs:

- A união de duas linguagens regulares é regular:
A ideia da prova parte do princípio de que, tendo duas linguagens regulares e os AFNDs que as reconhecem, podemos tomar os dois AFNDs e combiná-los com um terceiro AFND que aceita sua entrada de qualquer um dos dois AFNDs prévios a aceita. A prova formal pode ser encontrada aqui.

- A concatenação de duas linguagens regulares é regular:
Sendo dois AFNDs iniciais e , a ideia da prova consiste em construir um novo AFND usando a estrutura de e na construção. O novo AFND possui como estado inicial, o estado inicial do AFND e possui toda a estrutura de , com os estados de aceitação de transformados em estados de não-aceitação possuidores de transições para o estado inicial de . O novo AFND, por conseguinte, também possui toda a estrutura de , e seus estados de aceitação são os mesmos de . Dessa forma, a entrada sempre poderá ser *quebrada* em duas partes não-deterministicamente, e só será aceita se puder ter a primeira parte aceita por e a segunda por .

- O Fecho de Kleene de uma linguagem regular é regular:
Sendo o AFND inicial , é construído um novo AFND que modifica a estrutura de para fazer com que os estados de aceitação deste possuam, cada um, uma transição para o estado inicial de . Adicionalmente, cria-se um novo estado de aceitação que passa a ser o estado inicial de , este possuindo uma transição para o estado inicial *antigo* (estado inicial de).

É importante notar que a construção de a partir da mera adição do estado inicial de ao conjunto dos estados de aceitação não provaria o que é proposto, pois há casos nos quais certas cadeias indesejadas seriam aceitas. Um exemplo de linguagem cuja computação nessa construção leva a uma falha é a linguagem .

Comparação entre AFD e AFND

- Determinismo: AFD possui exatamente uma transição para cada par (estado, símbolo); AFND pode ter múltiplas ou nenhuma. - Facilidade de construção: AFNDs são mais fáceis de construir para linguagens complexas. - Implementação: AFDs são mais fáceis de implementar diretamente. - Equivalência: Para toda linguagem reconhecida por um AFND, existe um AFD que reconhece a mesma linguagem. Em resumo, embora os AFNDs possam parecer mais poderosos, eles são equivalentes em termos de capacidade de reconhecimento às AFDs.

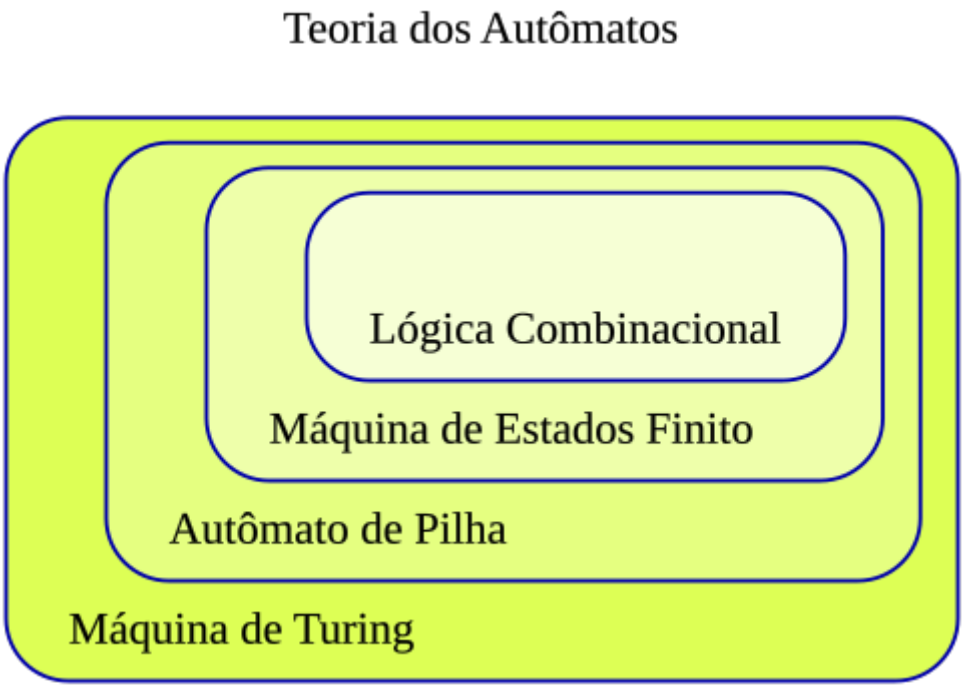
Conclusão

O estudo de autômatos finitos é essencial para compreender o funcionamento de linguagens formais e autômatos computacionais. Os modelos determinísticos e não determinísticos, embora distintos em sua estrutura e abordagem, são equivalentes em poder de reconhecimento. A compreensão desses modelos auxilia no desenvolvimento de compiladores, interpretadores e outros sistemas computacionais fundamentais.

Referências - HOPCROFT, John E.; ULLMAN, Jeffrey D. Introdução à teoria de autômatos, linguagens e computação. LTC, 2002. - SIPSER, Michael. Introdução à Teoria da Computação. Cengage Learning, 2011. - KOZEN, Dexter C. Automata and Computability. Springer, 1997. - AHO, Alfred V.; SETHI, Ravi; ULLMAN, Jeffrey D. Compilers: Principles, Techniques, and Tools. Addison-Wesley, 1986.

Teoria dos Autómatos

Teoria dos autômatos é o estudo das máquinas abstratas ou autômatos, bem como problemas computacionais que podem ser resolvidos usando esses objetos. É objeto de estudo tanto da Ciência da Computação Teórica como da Matemática Discreta. A palavra a*utômato* vem da palavra grega αὐτόματα que significa “autuação” (em tradução livre), isto é, sem influência externa.



A figura a cima ilustra uma máquina de estados finito, que pertence a uma variedade bem conhecida de autômato. Este autômato consiste em estados (representados na figura por círculos), e transições (representadas por setas).

Quando o autômato recebe um símbolo de entrada, ele faz uma transição (ou salto) para outro estado, de acordo com sua função de transição (que tem como entradas o estado atual e o símbolo recente). Teoria dos autômatos também está profundamente relacionada à teoria das linguagens formais. Um autômato é uma representação finita de uma linguagem formal que pode ser um conjunto infinito. Autômatos são frequentemente classificados pela classe das linguagens formais que são capazes de reconhecer, tipicamente ilustrado pela hierarquia de Chomsky, que descreve as relações entre várias línguas e tipos de lógica formalizada.

Autômatos desempenham um papel importante em teoria da computação, elaboração de compiladores, inteligência artificial, análise sintática e verificação formal.

Autômatos

Descrição Muito Informal

Um autômato é uma construção feita de *estados*, projetado para determinar se a entrada pode ser aceita ou rejeitada. Parece muito com um jogo de tabuleiro básico em que cada espaço no tabuleiro representa um estado. Cada estado tem informações sobre o que fazer quando uma entrada é recebida pela máquina (novamente, parece como o que fazer quando você cai num dos espaços em um popular jogo de tabuleiro). Quando a máquina recebe um nova entrada, ela analisa o estado e escolhe um novo local baseada na informações sobre o que fazer ao receber essa entrada nesse estado. Quando não há mais entradas, o autômato para e o espaço onde está quando conclui determina se o autômato aceita ou rejeita esse conjunto específico de entradas.

Descrição Informal

Um autômato *executa* (ou *roda*) quando lhe é dada uma sequência de *entradas* (individualmente) em passos de tempo discretos. Um autômato processa uma entrada que é obtida a partir de um conjunto de *símbolos* ou *letras*, que é chamado de alfabeto. Os símbolos recebidos pelo autômato como entrada, em qualquer etapa (ou passo) são uma sequência de símbolos chamados de *palavras*. Um autômato contém um conjunto finito de *estados*. Para cada instante durante a execução, o autômato está *em* um de seus estados. Ao receber uma nova entrada, ele se move para outro estado (ou faz a transição) baseado numa função que tem o estado atual e o símbolo como parâmetros. Esta função é chamada de *função de transição*. O autômato *lê* os símbolos da palavra de entrada, um após o outro, e faz a transição de estado para estado, de acordo com a função de transição, até a palavra ser totalmente lida. Uma vez que a palavra de entrada foi lida, diz-se que o autômato deve *parar*. O estado no qual o autômato para é chamado de *estado final*. Dependendo do estado final, o autômato pode *aceitar* ou *rejeitar* uma palavra de entrada. Existe um subconjunto de

estados do autômato, que é definido como o conjunto de *estados de aceitação*. Se o estado final é um estado de aceitação, então o autômato *aceita* a palavra. Caso contrário, a palavra é *rejeitada*. O conjunto de todas as palavras aceitas por um autômato é chamado de *linguagem reconhecida pelo autômato*. Qualquer subconjunto da linguagem de um autômato é uma *linguagem reconhecida pelo autômato*.

Em suma, um autômato é um objeto matemático que toma uma palavra como entrada e decide se a aceita ou rejeita. Como todos os problemas computacionais são redutíveis para o problema de aceitação/rejeição de palavras, (todas as instâncias do problema podem ser representadas por um tamanho finito de símbolos)[*carece de fontes*], a teoria dos autômatos desempenha um importante papel na teoria computacional.