

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/341327371>

Introduction to computer theory – formal languages – autonomous

Article · January 2017

CITATIONS

0

READS

31

1 author:



[Alvaro F. Pinheiro](#)

University of Pernambuco

29 PUBLICATIONS 5 CITATIONS

SEE PROFILE

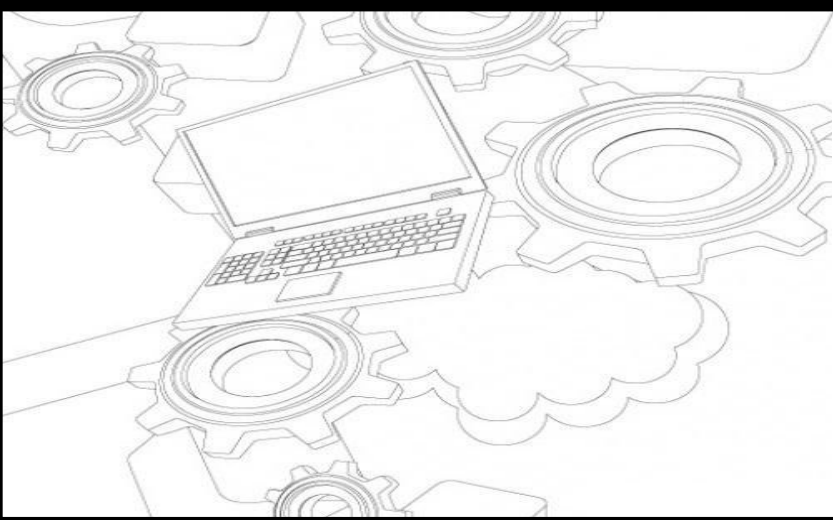
Álvaro Farias Pinheiro

Revista Fundamentos da Engenharia de Software

Teoria da Computação

Autômatos

Ano II – Número 5
Maio de 2017



Série Fundamentos da Engenharia de Software

Autômatos

PINHEIRO, Álvaro Farias
Autor

Série Fundamentos da Engenharia de Software

Autômatos

Publicação 2017

O autor acredita que todas as informações aqui apresentadas estão corretas e podem ser utilizadas para qualquer fim legal. Entretanto, não existe qualquer garantia explícita ou implícita, de que o uso de tais informações conduzirá sempre ao resultado desejado. Os nomes de sites e empresas, por ventura, mencionados, foram utilizados apenas para ilustrar os exemplos, não tendo vínculo nenhum com o livro, não garantindo a sua existência nem divulgação. Eventuais erratas estarão disponíveis para download no site de publicação.

As imagens utilizadas neste livro foram obtidas na Internet.

Dados da Publicação

Pinheiro, Álvaro Farias

Série Fundamentos da Engenharia de Software: Autômatos

Ano II – Número 5 – Recife, Maio de 2017

Selo Editorial: Publicação Independente

1. Introdução
2. Expressão Regular
3. Autômatos
4. Linguagem Formal

Série Fundamentos da Engenharia de Software Autômatos

Publicação Independente
Revista em português com o título
Autômatos
Série Fundamentos da Engenharia de Software
Ano II – Número 5
Recife – Pernambuco – Brasil
Maio de 2017

Introdução

A Engenharia de Software é um segmento da Computação que existe decorrente da necessidade de se construir *software* com qualidade, buscando planejar e organizar o desenvolvimento e dentro deste campo a disciplina Teoria da Computação tem a finalidade de conceituar o funcionamento dos compiladores, entendendo como as linguagens de programação são construídas e classificadas.

A finalidade dessa disciplina é fomentar os conceitos básicos para o conhecimento de compiladores, no qual o seu entendimento é essencial para a construção de um programa de computador que, a partir de um código fonte escrito em uma linguagem compilada, cria um programa semanticamente equivalente, porém escrito em outra linguagem, chamado de código final. E para entender um pouco mais é necessário a elucidação das gramáticas: léxica, sintática e semântica.

Análises de um compilador:

- Léxica - é o processo de analisar a entrada de linhas de caracteres e produzir uma sequência de símbolos;
- Sintática - é a análise das regras que regem a construção de frases; e
- Semântica - é a terceira fase da compilação onde se verifica os erros semânticos, isto é, de lógica.

1.1 Definições

O que é computação? Uma forma de defini-la é a forma de se estudar a solução de um problema, isto é, analisar e projetar solução a um problema.

O que é Teoria da Computação? É um subcampo da ciência da computação, que tem por objetivo descobrir quais problemas podem ser computados, fazendo o uso de um modelo.

1.2 Modelos

Foi no século XX que teve início aos estudos do que iria ser a Teoria da Computação. O objetivo era (e é ainda), descobrir quais problemas matemáticos poderiam ser resolvidos por um método simples.

Diversos modelos a partir desse momento foram propostos, como a Máquina de Turing, modelo Recursivo, Cálculo Lambda, Regras Gramaticais, Cadeias de Markov, Sistemas de Post, etc. Todos os formalismos propostos acima são equivalentes em termos computacionais, assim o objetivo atual é verificar a possibilidade de realizar certos tipos de computação em determinados tipos de máquina.

1.3 Finalidade

A teoria da computação serve para estudar os modelos computacionais, como também os limites dessa computação. Buscando respostas para questões do tipo: Independente do *hardware*, quais problemas jamais poderão ser resolvidos por uma máquina? Problema da parada; Problema da Correspondência de Post.

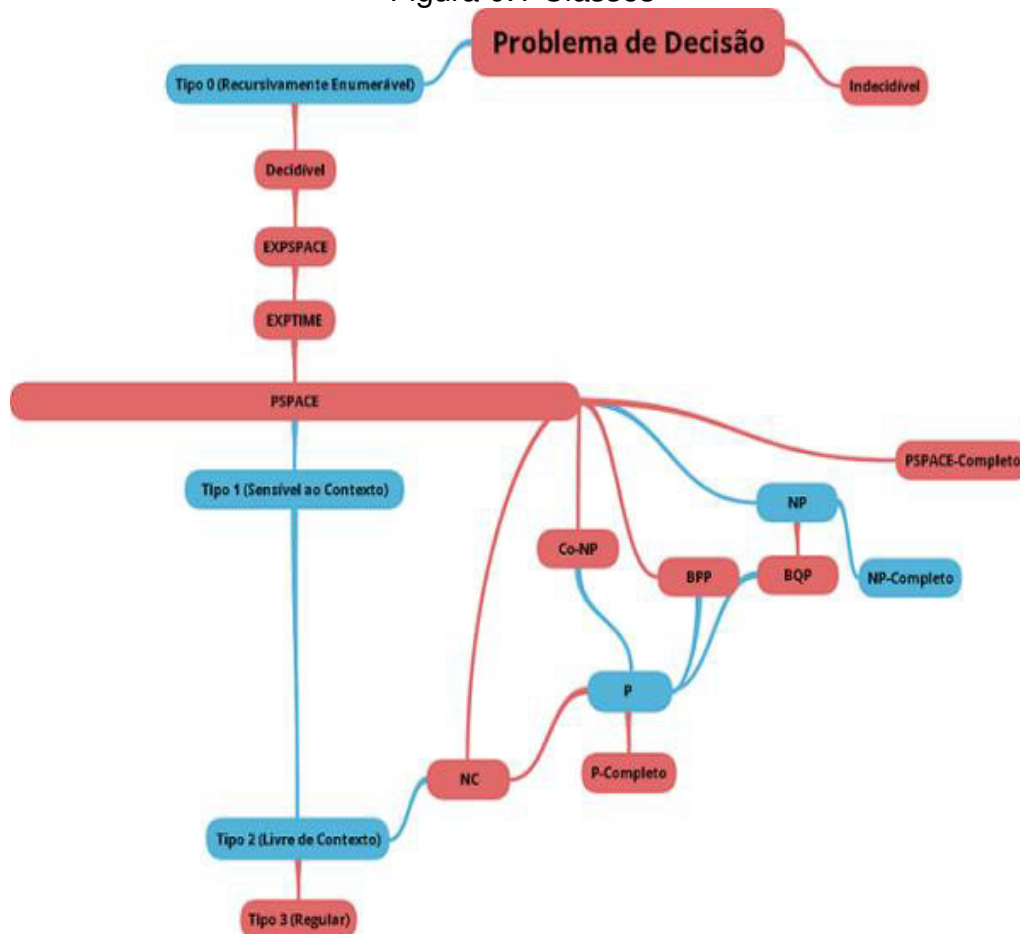
Quais problemas podem ser resolvidos, mas necessitam de um longo período que se torna impraticável? (Aritmética de Presburger) Em que situações podem ser mais difíceis a resolução de um problema do que verificar cada uma das soluções manualmente? Classes P e NP.

1.4 Classes

A teoria da computação se divide em duas classes. Observando a figura 1.1 se pode observar as classes azuis que pertencem à teoria da computabilidade e as em vermelho a teoria da complexidade. Assim, com o estudo das Expressões Regulares, podem se identificar e especificar padrões de cadeias de caracteres, que são utilizados em linguagens de programação.

Outro formalismo são os Autômatos Finitos, utilizados em desenho de circuitos e em alguns sistemas de resolução de problemas. As gramáticas livres de contexto são utilizadas para especificar a sintaxe das linguagens de programação; um formalismo equivalente são os autômatos com pilha, ou *pushdown* autômato.

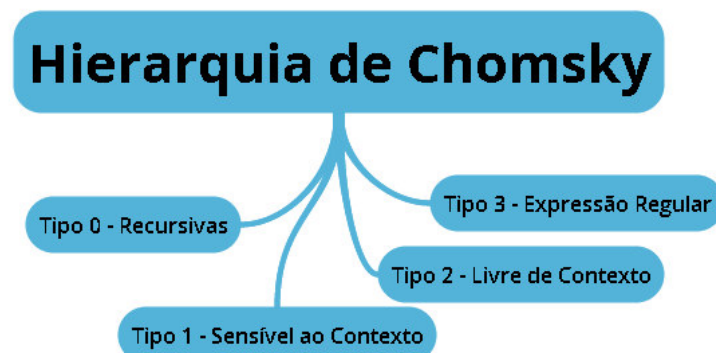
Figura 0.1 Classes



Fonte: Próprio Autor

Modelos de computação diferentes podem realizar tarefas distintas. Assim, uma forma de estudar o poder de um modelo computacional é estudar a classe das linguagens formais que o modelo pode gerar.

Figura 1.2 Hierarquia Chomsky



Fonte: Internet

1.5 Expressão Regular

Uma *regex* é uma forma flexível de identificar cadeias de caracteres. Essas expressões podem ser escritas numa linguagem formal e interpretadas por programas que servem como analisadores sintáticos, examinando o texto e identificando partes que casam com a especificação dada.

O matemático Kleene desenvolveu as *regex* como uma notação ao que ele chamava de álgebra de conjuntos regulares, servindo como base para os primeiros algoritmos computacionais de busca e depois para ferramentas de tratamento de texto.

Atualmente as *regex* servem para procura e substituição de texto em editores de texto e linguagens de programação, validação de formatos de texto, validação de protocolos, validação de formatos digitais, realce de sintaxe e filtragem de informação.

Como uma *regex* serve para descrever um conjunto de cadeias de caracteres sem precisar listar todos os elementos do conjunto, existem alguns caracteres coringa para isso.

O formalismo padrão usa três operações básicas:

- (+) união
- (.) concatenação
- (*) kleene

Essas construções podem ser combinadas arbitrariamente para formar expressões complexas, assim como expressões aritméticas com números e operações de adição, subtração, multiplicação e divisão.

1.5.1 União

$$L(E + F) = L(E) \cup L(F)$$

$$E = \{001, 110\} \quad F = \{\epsilon, 11, 110\}$$

$$L = \{001, 110, \epsilon, 11\}$$

1.5.2 Concatenação

$$L(E \cdot F) = L(E) \cdot L(F)$$

$$E = \{001, 110\} \quad F = \{\epsilon, 11, 110\}$$

$L=\{001,00111,001110,110,11011,110110\}$

1.5.3 Kleene

$L(E^*) = (L(E^*))^*$

$E=\{001,110\}$

$L=\{\epsilon,001,110,001001,001110,\dots\}$

$F=\{"ab", "c"\}^*$

$L=\{\epsilon, "ab", "c", "abab", "abc", "cab", "cc", "ababab", "abcab", \dots\}$

1.6 Teoria dos Autômatos

É um subcampo da ciência da computação que estuda as máquinas de estados finitas, considerando suas representações matemáticas.

Um autômato é um modelo matemático de uma máquina de estados que funciona como um reconhecedor de uma determinada linguagem. Baseado na Teoria dos Sistemas, os estados estão sempre em transição, por exemplo: uma TV pode estar ligada e transitar para desligada, tem então um sistema com dois estados.

Uma forma de utilização dos autômatos é no reconhecimento de padrões, a exemplo, editores de textos.

1.7 Classes dos Autômatos

A seguir serão apresentados três tipos de autômatos finitos:

1.7.1 Autômatos Finitos Determinísticos

Cada estado de um autômato desse tipo tem uma transição para cada símbolo do alfabeto;

1.7.2 Autômatos Finitos Não Determinísticos

Os estados de um autômato desse tipo podem ou não ter uma transição para cada símbolo do alfabeto, ou podem ter até mesmo múltiplas transições para cada símbolo; e

1.7.3 Autômatos Finitos Não Determinísticos com Transições ϵ (Espontâneas)

Além de ser capaz de pular para mais (ou nenhum) estados com quaisquer símbolos, esse tipo de autômato pode pular para outros estados sem que haja nenhum símbolo (*String* vazia).

1.8 Outros Tipos de Autômatos

Extensões de Autômatos Finitos. A família de linguagens aceitas pelos autômatos descritos acima é chamada de família de Linguagens regulares.

Autômatos mais poderosos podem aceitar linguagens mais complexas. Tais autômatos incluem:

1.8.1 Autômatos à Pilha (ou *pushdown*)

Tais máquinas são idênticas aos autômatos finitos determinísticos, exceto pelo fato deles adicionalmente carregarem a memória sob a forma de pilhas. A função de transição δ irá depender agora dos símbolos que estão no topo da pilha, e especificarão como a pilha deve ser modificada a cada transição. Os autômatos à pilha aceitam linguagens livres de contexto.

1.8.2 Máquina de Turing

Essas são as mais poderosas máquinas computacionais. Elas possuem uma memória infinita sob a forma de uma fita, uma cabeça que pode realizar a leitura ou alteração da fita, além de poder movimentar-se ao longo da fita. As máquinas de Turing são equivalentes a algoritmos e compõem a base teórica da computação moderna. Máquinas de Turing aceitam linguagens recursivamente enumeráveis.

1.8.3 Autômato Linearmente Limitado ou Linear *Bounded*

É uma máquina de Turing limitada, ao invés de ser uma fita infinita, a fita tem uma quantidade de espaço proporcional ao tamanho da *String* de entrada. Aceitam linguagens sensíveis ao contexto.

1.9 Projetos de Linguagens de Programação

Agora que entendemos o básico, vamos entender o que são linguagens de programação e como elas são projetadas.

Existem algumas características básicas necessárias a serem trabalhadas para projetarmos uma linguagem. Essas são: a sua arquitetura, a configuração técnica, os seus padrões e a compatibilidade com o legado.

Então vamos entender o que é arquitetura de uma linguagem. Basicamente podemos dizer que, projetar uma linguagem é levar em consideração seu *hardware*, e o modelo que o define, isto é, o modelo de von Neumann Eckert, lembrando, trata-se do modelo, entrada-processamento-saída. Então, boa parte, das linguagens, como C, é projetada para essa arquitetura de máquina, mas existem outras que não o são a exemplo o Lisp.

Detalhando um pouco mais a questão de arquitetura, temos que as linguagens são projetadas para atendê-las, e nos dias atuais, são dois os projetos básicos de arquiteturas:

Complex Instruction Set Computer (CISC), traduzindo Computador com um Conjunto Complexo de Instruções.

Explicando basicamente o que é, trata-se de um padrão de construção de processadores capazes de executar centenas de instruções complexas. Mas, o que quer dizer isso? Uma forma simples de explicar, que dizer que, no processador, são armazenadas instruções chamadas de microcódigo, que se não o existissem, teriam que ser codificadas pelos programadores nas linguagens. Esse armazenamento de instruções nos processadores minimiza e simplifica o trabalho de desenvolvimento, já que muito código comum a qualquer tipo de programa já está embutido no processador.

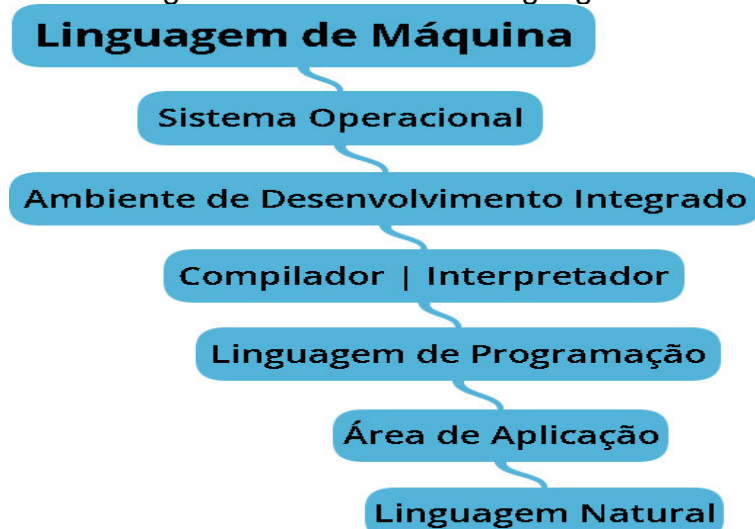
Como consequência direta as linguagens concebidas para essa arquitetura, tendem a ter um código executável bem menor. Mas, a sua desvantagem é que como a microprogramação realizada visa prever uma grande quantidade de funcionalidades comuns aos aplicativos, torna esse tipo de processador se torna pesado em relação ao tempo de resposta. E para completar com o passar do tempo, notou-se que muitas dessas funcionalidades embutidas nos microcódigos nunca eram usadas na maioria das aplicações desenvolvidas. Resumindo temos um melhor código com pior desempenho.

Reduced Instruction Set Computer (RISC), traduzindo Computador com um Conjunto Reduzido de Instruções.

Trata-se de outro padrão de construção de processadores, que como a própria sigla já diz, oferece um conjunto simples e pequeno de instruções armazenadas no processador. Simplificando, os processadores desse padrão não possuem ou possuem muito pouco microcódigo. Consequência direta é que o esforço de desenvolvimento torna-se maior. Porém, como os processadores não possuem microcódigo desnecessário, prevendo funcionalidades comuns a diversos tipos de programas, que normalmente não são usadas, o tempo de resposta é muito superior. Grandes partes dos projetos de processadores atuais seguem esse modelo de arquitetura. Resumindo temos um melhor desempenho com pior código.

Outra preocupação no projeto de uma linguagem de programação é em relação ao nicho que ela pretende atender, para resolver os seus problemas. Esse objetivo gera uma série de restrições, de natureza técnica ou de modelo, que são normalmente observados, durante o projeto de uma linguagem de programação. Essas restrições tecnológicas estão relacionadas aos seguintes níveis de abstração, observadas na figura 1.3.

Figura 1.3 Camadas de Linguagens



Fonte: Próprio Autor

Mais uma preocupação no projeto de desenvolvimento de uma nova linguagem é a sua compatibilidade com os seus projetos anteriores, caso existam. Isso se deve pela necessidade de se poderem manter os códigos produzidos pelas versões anteriores com a nova linguagem. Uma tarefa muito complicada. Mas, a preocupação com o legado na área de TIC é uma constante.

Depois, que uma linguagem se torna bem aceita pelo mercado, mais uma preocupação se passa a ter que é a sua padronização. Essa padronização é necessária, para uniformizar o uso e a evolução da linguagem. Normalmente as entidades que formalizam a padronização são a ANSI e o ISO, veja os *links* para saber mais.

Uma categorização importantíssima nos projetos de linguagens de programação é a Hierarquia de Chomsky, que classifica as linguagens em tipos, os quais podem ser vistos abaixo.

Tipo 3: Gramática Regular > Linguagem Regular > Autômato Finito

Tipo 2: Gramática Livre de Contexto > Linguagem LC > Autômato com Pilha

Tipo 1: Gramática Sensível ao Contexto > Linguagem SC > Autômato Linear

Tipo 0: Gramática Irrestrita > Linguagem Recursiva > Máquina de Turing

Mas para se ter uma ideia dos mesmos, vamos falar sucintamente sobre eles. Começando com a explicação de Linguagens Formais e Autômatos. Basicamente é o estudo de modelos matemáticos que permitam definir e reconhecer linguagens, possibilitando a sua classificação, estruturação e caracterização. É conhecida como Teoria da Ciência da Computação. A finalidade do estudo dessas teorias é fornecer fundamentos para a Ciência da Computação em relação à decidibilidade, computabilidade e complexidade computacional. Essas teorias fundamentam várias áreas da computação, como processamento de linguagens, reconhecimento de padrões, modelagem de sistemas.

1.10 Linguagem Formal

Sabemos que linguagem é uma forma de comunicação e que com a evolução do homem, essas ganharam especializações, do tipo léxico, sintático e semântico. Essas especializações servem tanto para linguagens naturais ou idiomas, como também para linguagens de programação e dos protocolos de comunicação. Assim, podemos dizer que as Linguagens Formais consistem de regras para representação e especificação de linguagens.

As representações são feitas por reconhecedores e geradores de linguagens, que nada mais é, que um recurso que verificam se uma sentença pertence ou não a uma determinada linguagem. Esses recursos são chamados de autômatos, que podem ser: Autômatos Finitos Determinísticos; Autômatos Finitos Não Determinísticos; Autômatos de Pilha; e Máquina de Turing.

Já os geradores são recursos que permitem a geração sistemática de todas as sentenças de uma linguagem. Esses recursos são as gramáticas, a qual se destaca a de Chomsky. Resumindo, as linguagens formais podem ser representadas de maneira finita e precisa através de teorias matemáticas, o que caracteriza a Ciência da Computação.

1.10.1 Tipo 3

Das linguagens regulares, que podem ser representados pelos autômatos finitos determinísticos e não determinísticos. Essas linguagens se baseiam nas Expressões Regulares. Essas expressões são representadas por três formas básicas: União, representada pelo símbolo (+) permitindo unir expressões; Concatenação, representada pelo símbolo (.) permitindo concatenar

expressões; e Klenee, representada pelo símbolo (*) permitindo potencializar expressões.

Seguem exemplos das *Regular Expression (RegEx)*:

Para $L(E + F) = L(E) \cup L(F)$, como $E=\{001,110\}$ e $F=\{\epsilon,11,110\}$, temos $L=\{001,110,\epsilon,11\}$

Para $L(E \cdot F) = L(E) \cdot L(F)$, como $E=\{001,110\}$ e $F=\{\epsilon,11,110\}$, temos $L=\{001,00111,001110,110,11011,110110\}$

Para $L(E^*) = (L(E))^*$, como $E=\{001,110\}$, temos $L=\{\epsilon,001,110,001001,001110,\dots\}$

Essas expressões podem ser representadas por Autômatos. E o que são? São máquinas de estados finitos, isto é, um autômato finito. Trata de uma modelagem de um comportamento, composto por estados, transições e ações. Um estado armazena informações sobre o passado, isto é, ele reflete as mudanças desde a entrada num estado, no início do sistema, até o momento presente. Uma transição indica uma mudança de estado e é descrita por uma condição que precisa ser realizada para que a transição ocorra. Uma ação é a descrição de uma atividade que deve ser realizada num determinado momento. Máquinas de estados finitos podem ser representadas por meio de um diagrama de estados ou diagrama de transição de estados.

A quintupla dos autômatos finitos é: $M = (Q, \Sigma, \delta, i, F)$ onde:

Q é um conjunto finito de estados;

Σ é o alfabeto de entrada;

δ é a função de transição de estados;

i é o estado inicial; e

F é um conjunto de estados finais ou de aceitação.

Segue um exemplo de autômato finito para definição de uma linguagem:

$L = \{p \in \{0,1\}^* \mid 111 \text{ seja uma subcadeia de } p\}$

Lê-se: Para uma linguagem, dado uma palavra que pertence ao dialeto 0 e 1 baseado na expressão de klenee, tal que, 111 seja uma subcadeia da palavra.

Temos:

Observe que os valores da quintupla para esse exemplo são:

Para Sigma temos 0 e 1;

Para Q temos q_0, q_1, q_2, q_3, q_4 e q_5 ;

Para Delta temos a tabela de transições acima;

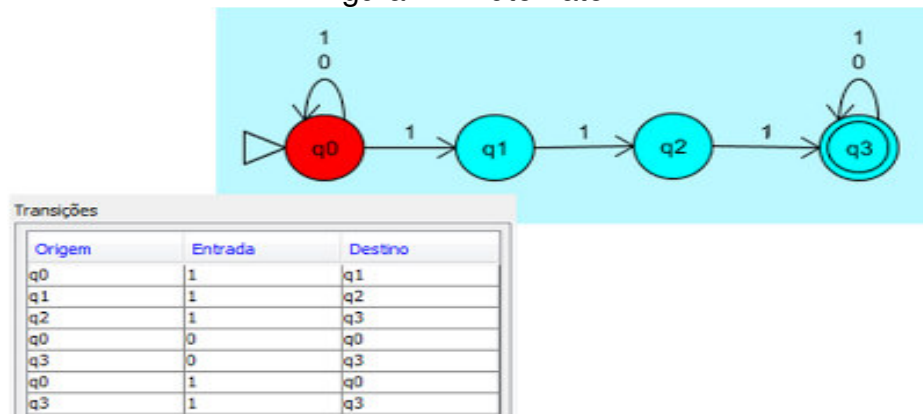
Para i temos q_0 ; e

Para F temos q_3 e q_4 .

Um detalhe importante para a definição dos autômatos finitos é a pequena diferença entre os determinísticos (AFD) e não determinísticos (AFND). O primeiro só permite a transição de um estado para outro, já o segundo é mais flexível, permitindo transições no mesmo estado, até mesmo transições espontâneas, isto é, independente de uma leitura. Veja o exemplo, para a mesma linguagem, sendo o primeiro atendido por um AFD e o segundo por um

AFND. Para a mesma linguagem $L=\{p \in \{0,1\}^* | 111 \text{ seja uma subcadeia de } p\}$ temos uma representação mais limpa, observe a figura 1.4.

Figura 1.4 Autômato



Fonte: Próprio Autor

1.10.2 Tipo 2

Das linguagens livre de contexto, essas são representadas por autômatos com pilhas. Que são idênticos aos autômatos finitos determinísticos, exceto pelo fato eles adicionalmente usam o recurso de memória sob a forma de pilhas. A função de transição δ irá depender agora dos símbolos que estão no topo da pilha, e especificarão como a pilha deve ser modificada a cada transição.

Os autômatos com pilha determinísticos são definidos com a sêxtupla dos autômatos com pilha é: $M = (Q, \Sigma, G, \delta, i, I, F)$ onde:

Q é o conjunto finito de estado;

Σ é o alfabeto de entrada;

δ é a função de transição;

i é o estado inicial;

I é o símbolo inicial da pilha; e

F é o conjunto de estados finais.

Os autômatos com pilha não determinísticos são definidos com a sétupla dos autômatos com pilha é: $M = (Q, \Sigma, G, \delta, i, I, F)$ onde:

Q é o conjunto finito de estado;

Σ é o alfabeto de entrada;

G é o alfabeto da pilha;

δ é a função de transição;

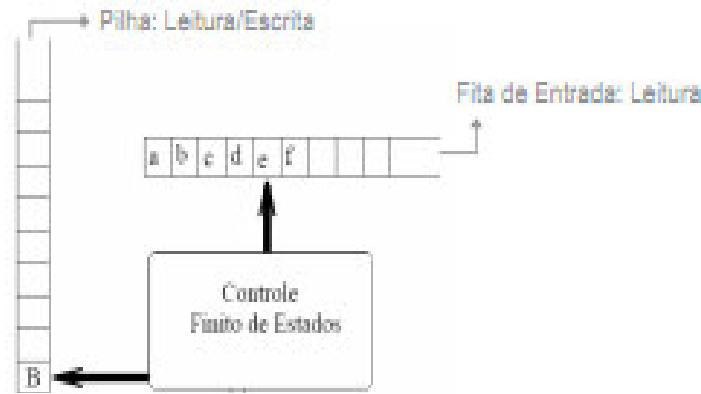
i é o estado inicial;

I é o símbolo inicial da pilha; e

F é o conjunto de estados finais.

Análogo às Linguagens Regulares (LR), as Linguagens Livres de Contexto (LCC) podem ser utilizadas como reconhecedoras. O seu funcionamento e os componentes de um Autômato com Pilha (AP) são: a) uma fita de entrada, b) uma cabeça de leitura, c) um indicador de estado e d) uma pilha. Veja a figura 1.5.

Figura 1.5 Autômato



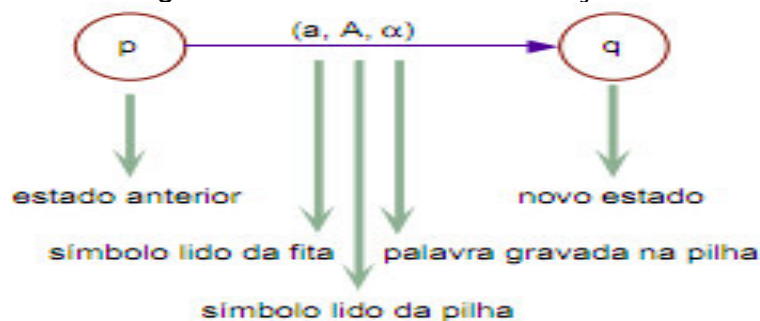
Fonte: Internet

O funcionamento de um AP consiste em posicionar a cabeça de leitura em um símbolo de entrada e em seguida posicionar a outra cabeça de leitura no topo da pilha obtendo os valores contidos na fita e na pilha e, com base nesses valores os seguintes passos devem ser realizados:

- Remover o item no topo da Pilha;
- Colocar um novo item na Pilha;
- Transitar para outro estado; e
- Avançar para o próximo símbolo na entrada.

Para entender melhor esse mecanismo, veja a figura 1.6 e observe os passos para reconhecimento da linguagem: $L = \{a^*b^* \mid * \geq 0\}$. Para cada a lido, deve-se inseri-lo na pilha, e para cada b lido deve-se remover um a da pilha, no término uma palavra p será reconhecida se no final do processo a pilha estiver vazia.

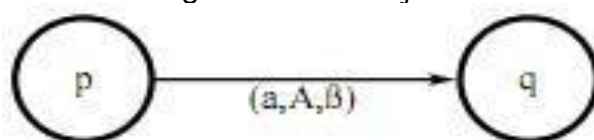
Figura 1.6 Elementos da Transição



Fonte: Internet

A figura 1.7 representa a transição, onde temos (a) é o símbolo lido da fita, (A) representa o símbolo lido da pilha, e (β) a palavra gravada na pilha. Resumindo temos:

Figura 1.7 Transição



Fonte: Internet

Vamos entender o AP com o exemplo:

$$L1 = \{a^*b^* \mid * \geq 0\}$$

$$M1 = (\{a,b\}, \{q1,q2,q3\}, \delta, q1, \{B\})$$

$$\delta_1(q1, a, ?) = \{(q1, B)\}$$

$$\delta_1(q1, b, B) = \{(q2, ?)\}$$

$$\delta_1(q1, ?, ?) = \{(q3, ?)\}$$

$$\delta_1(q2, b, B) = \{(q2, ?)\}$$

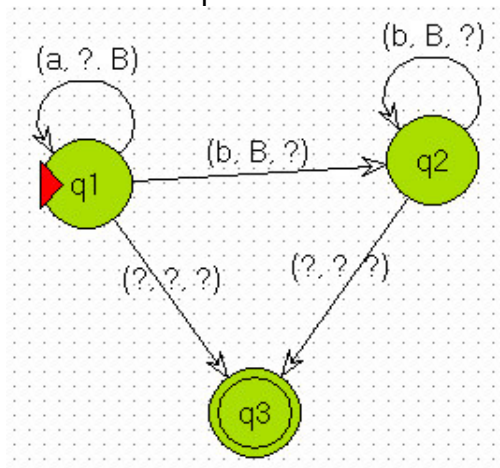
$$\delta_1(q2, ?, ?) = \{(q3, ?)\}$$

A máquina M1 para representar a linguagem L1 é iniciada no estado q1 e para cada símbolo (a) lido da fita é armazenado um símbolo (B) na pilha, no estado q2 é realizada a verificação se cada símbolo (b) da fita existe um correspondente (B) na pilha. O algoritmo somente aceitará a palavra se ao terminar de ler toda ela a pilha estiver vazia.

A representação desse autômato em grafo pode ser observado na figura 1.8 e essa máquina aceita palavras como: ab, aabb, aaabbb,...

Para explicar melhor, a máquina M1 possui o estado q1 com três símbolos, os quais o primeiro da esquerda para direita é o símbolo a ser reconhecido que será lido na fita (a), o segundo é o símbolo a ser lido da pilha (?=vazio) e o terceiro é o símbolo a ser gravado na pilha (B). Para cada (a) lido na fita um (B) foi gravado na pilha, podendo ser repetido quantas vezes for desejado. A transição do estado q1 para o q2 consiste em uma leitura de três símbolos, o qual tem o símbolo que será lido da fita (b), em seguida o símbolo que será lido da pilha (B), e o terceiro o símbolo que será gravado na pilha (?) assim sendo um vazio. No estado q2 com os mesmos três símbolos, temos o símbolo que será lido da fita (b), em seguida o símbolo que será lido da pilha (B), e o terceiro o símbolo que será gravado na pilha (?) assim sendo um vazio, podendo ser repetido quantas vezes for desejado. Quando não existir mais símbolos para serem lidos na fita e da pilha, chega-se ao final.

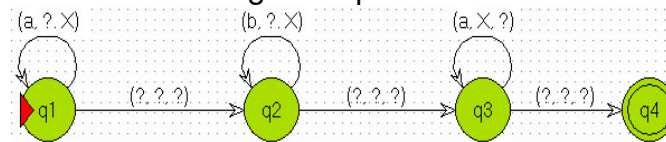
Figura 1.8 Gerado pela ferramenta Simulador



Fonte: Próprio Autor

Mais um exemplo. Fazer uma máquina para: $M2 = \{a^n b^m a^{n+m} \mid n \geq 0 \text{ e } m \geq 0\}$ Esta máquina aceita palavras como: abaa, aabbaaaa, aaabbbaaaaaa, ... como pode ser observado na figura 1.9.

Figura 1.9 Autômato gerado pela ferramenta Simulador



Fonte: Próprio Autor

1.10.3 Tipo 1

Das linguagens sensíveis ao contexto, essas são representadas por uma forma restrita de uma máquina de Turing não determinística que satisfaça as duas condições seguintes: Seu alfabeto de entrada inclui dois símbolos especiais, servindo como marcadores de direção da esquerda e direita; e Necessitam de um ponteiro que se move para a esquerda levando além do marcador.

1.10.4 Tipo 0

Das linguagens recursivas, essas são representadas por máquinas como as de Turing, Moore e Mealy. A Máquina de Turing é um recurso teórico conhecido também como máquina universal, que foi concebido pelo matemático britânico Alan Turing, antes do surgimento dos modernos computadores digitais. É um modelo abstrato de computador, que trata apenas dos aspectos lógicos de funcionamento da memória, estados e transições. Porém com essa abstração pode-se modelar qualquer computador digital. A Máquina de Moore é outro recurso que utiliza apenas ações de entrada e a saída depende somente do estado. A vantagem do modelo de Moore é a simplificação do comportamento. Por exemplo, modelar um elevador com quatro estados possíveis para sua porta "Aberta", "Fechada", "Abrindo", "Fechando". A máquina de estados reconhece dois comandos: "comando_abrir" e "comando_fechar" que disparam a alteração de estado. A ação de entrada no estado "Abrindo" liga o motor que abre a porta, a ação de entrada no estado "Fechando" liga o motor na outra direção, fechando a porta. Os estados "Aberta" e "Fechada" não desempenham nenhuma ação. E a Máquina de Mealy é mais um recurso onde a saída depende da entrada e do estado. O uso de uma máquina Mealy normalmente leva a uma redução no número de estados. O exemplo anterior nessa máquina seria: inicie o motor para fechar a porta se o "comando_fechar" chegar e inicie o motor na direção oposta para abrir a porta se o "comando_abrir" chegar.

1.10.5 Máquina de Turing

A Máquina de Turing (MT) é bastante simples, baseia-se em uma fita de comprimento infinito que pode se mover para direita ou para esquerda célula-a-célula e cada célula pode conter um caractere ou estar vazio. A cabeça chamada de C é fixa e pode ler e escrever conteúdo em uma célula, deixando-o cheio ou vazio.

A MT é composta por uma fita que como já dito é infinita e pode se mover. Por um registrador de estados que armazena o estado da máquina, isto é, um dispositivo que armazena os estados, incluindo o estado inicial com o qual o registrador é inicializado. A MT também é composta por uma tabela de

transição, que determina o caractere a ser escrito, como também o sentido da navegação da fita, se para esquerda e ou para direita, e qual o novo estado, isto é, caractere que foi lido na fita e o estado em que se encontra. E caso não exista entrada de dados a máquina para.

A séptupla que define a MT é $M = (Q, \Sigma, \Gamma, s, b, F, \delta)$ onde:

Q é um conjunto finito de estados;

Σ é um alfabeto finito de caracteres;

Γ é o conjunto finito de caracteres;

s é o estado inicial;

b é o símbolo branco; e

δ é o conjunto dos estados finais.

Para entender o funcionamento da MT vamos verificar a figura 1.10 com o exemplo utilizando o alfabeto $\{b, 0\}$, no qual o b (bê) representa o caractere branco e o 0 (zero) o caractere a ser lido e escrito. O comportamento dessa máquina será o seguinte: Ela espera uma série de 0's na fita, com a cabeça posicionada inicialmente no 0 mais à esquerda, e repete os 0's com um b no meio. Representando, temos "000" que se torna "000b000". Nesse exemplo o conjunto de estados é $\{s1, s2, s3, s4, s5\}$ e o estado inicial é $s1$. Para visualizar o exemplo, segue a tabela de ação

Figura 1.10 Máquina de Turing

Estado	Caractere	Caractere		Estado
Ação	Lido	Escrito	Sentido	
Novo				

s1	0	b	→	s2
s2	0	0	→	s2
s2	b	b	→	s3
s3	b	0	←	s4
s3	0	0	→	s3
s4	0	0	←	s4
s4	b	b	←	s5
s5	0	0	←	s5
s5	b	0	→	s1

Fonte: Próprio Autor

A leitura dessa tabela pode ser feita da seguinte forma como mostra a figura 1.11: Estando a MT no estado $s1$ e o caractere lido pela cabeça for zero, então se escreve o caractere b e move-se uma posição para a direita, além de mudar o estado para $s2$.

Figura 1.11 MT

Passo	Estado	Fita

1	s1	00
2	s2	b0
3	s2	b0b
4	s3	b0bb
5	s4	b0b0

6	s5	b0b0
7	s5	b0b0
8	s1	00b0
9	s2	0bb0
10	s3	0bb0
11	s3	0bb0b
12	s4	0bb00
13	s4	0bb00
14	s5	0bb00
15	s1	00b00

Fonte: Próprio
Autor

O comportamento dessa máquina pode ser descrito como um laço, que é iniciado em s1, substitui o primeiro zero por um b, então usa o s2 para mover para a direita, passando pelos 0's e pelo primeiro b encontrado, s3 então passa pela próxima sequência de 0's e substitui o primeiro b que encontrar por um 0, s4 move de volta para a esquerda, passando pelos 0's até encontrar um b e passando para o estado s5. O s5 então move para a esquerda, passando pelos 0's até achar o b escrito inicialmente em por s1. Substitui o b por 0, move uma posição para a direita e entra no estado s1 para realizar outra passada pelo laço, repetindo o processo até s1 encontrar um b que fique entre duas cadeias de 0's.

Finalizando, os principais objetivos de um projeto de linguagem de programação são a obtenção da:

- Abstração; Clareza;
- Confiabilidade;
- Eficiência;
- Legibilidade;
- Ortogonalidade;
- Simplicidade; e
- Suporte

Livros da série Fundamentos da Engenharia de Software



Fundamentos da Engenharia de Software: Conceitos Básicos é uma coletânea de disciplinas que integradas servem para fundamentar o

entendimento da construção de projetos de software com qualidade, isto é, baseado em processos maduros e reconhecidos pela comunidade tecnológica. O objetivo deste livro é fornecer ao leitor as bases necessárias para o desenvolvimento de aplicações sejam Desktop, Web ou Mobile. Iniciando a leitura na Teoria da Computação, passando por Processos, Linguagens, Bancos de Dados e finalizando com Sistemas de Informação e Colaboração. Este livro pode ser lido capítulo a capítulo ou somente a disciplina desejada, pois sua elaboração consiste na compilação das disciplinas fundamentais da Engenharia de Software que são independentes, mas ao mesmo tempo se integram objetivando o desenvolvimento de aplicações.



Introdução à Banco de Dados. Neste são abordados os conceitos básicos de bancos de dados e seus sistemas gerenciadores, mas com o foco

na arquitetura relacional, porque ainda hoje o mercado faz uso em larga escala desses bancos de dados, mesmo que o paradigma predominante seja o orientado a objetos e que, já existam a um bom tempo bancos orientados a objeto, até mesmo os bancos objetos-relacionais que são um híbrido entre essas duas arquiteturas, o que predomina ainda é o relacional, assim, este material é focado na linguagem de consulta estruturada para os SGBD-Rs do mercado, com foco na comparação de cinco dos mais utilizados bancos relacionais, os quais são: Oracle, SQLServer, MySQL, SQLBase e Interbase.



Este livro é sobre processos de desenvolvimento de software, evidenciando a necessidade de qualidade na construção de sistemas, conceituando a diferença entre desenvolvimento Adhoc e com processo. Para isso é realizado a introdução à engenharia de requisitos abordando as técnicas para a elicitação de requisitos que forneçam subsídios necessários para uma construção de software com maior qualidade, enfatizando a necessidade de se aplicar na construção de qualquer sistema as técnicas de análise e modelagem, evidenciando o uso da linguagem da Linguagem de Modelagem Unificada (UML) para diagramar um projeto de software, explicando a necessidade do uso de modelos na construção, entrando com detalhes na análise orientada a objetos, com o objetivo de explorar os seus conceitos de requisitos e modelagem integrados. Este material é finalizado com a introdução à medidas de esforço de desenvolvimento, técnica necessária para responder as perguntas básicas de qualquer desenvolvimento: Qual o prazo e custo? E para responder a essas questões é abordado o uso da métrica análise de ponto de função.



Este livro aborda os sistemas que são classificados como informação, a exemplo, sistemas de apoio a decisão, sistemas estratégicos, sistemas

gerenciais e sistemas transacionais. A produção deste material que compõe o volume 4 da coleção Fundamentos da Engenharia de Software é resultado da compilação das aulas produzidas nas disciplinas que compõem os capítulos deste livro.



A motivação deste livro é exemplificar os conceitos de Padrões de Projetos utilizando a linguagem de programação Java, sendo a construção uma

compilação das aulas produzidas com o intuito de facilitar o entendimento do assunto abordando os seguintes temas: Paradigma Orientado a Objetos que introduz o leitor nos conceitos do POO; Linguagem de Modelagem Unificada para apresentar a simbologia UML dos conceitos de POO; Linguagem de Programação Java apresentando essa poderosa linguagem de programação orientada a objetos para exemplificar os padrões de projeto; e Padrões de Projetos que neste livro aborda os mais referenciados nas academias, sendo eles o GRASP e GoF.



Este livro é o resultado do uso da ferramenta MS Project da Microsoft utilizada na aplicação dos conceitos de gestão de projetos do PMBOK com as premissas da engenharia de testes para aquisição de qualidade nos produtos de software.



desenvolvimento e lógica de programação.

Este livro aborda basicamente os conceitos básicos de programação como autômatos, tipos de linguagens, princípios dos compiladores, paradigmas de



dicas de codificação para a linguagem de marcação ASPX, para a linguagem de script mais utilizada pelos navegadores o JavaScript com exemplos de CSS e principalmente dicas de código para a linguagem de programação CSharp e de banco de dados SQL com foco no SQLServer.

Este livro introduz nas tecnologias Web abordando os conceitos básicos para desenvolvimento para Internet com a apresentação da plataforma Dot Net e exibindo