Cursos: Bacharelado em Ciência da Computação e
Bacharelado em Sistemas de Informação
Disciplinas: (1493A) Teoria da Computação e Linguagens Formais,
(4623A) Teoria da Computação e Linguagens Formais e
(1601A) Teoria da Computação
Professora: Simone das Graças Domingues Prado
e-mail: simonedp@fc.unesp.br
home-page: wwwp.fc.unesp.br/~simonedp/discipl.htm

Apostila 06
Assunto: Teoria da Computação

Objetivos:
⇒ Estudar a Computabilidade
⇒ Estudar a Decidibilidade
⇒ Estudar a Redutibilidade

Conteúdo:
1. Introdução
2. Computabilidade e Decidibilidade
3. Redutibilidade
4. Problemas Indecidíveis

1. Introdução

Na Apostila 01 (TC01.pdf), desse curso, foi mostrado o panorama do que seria estudado na disciplina (releia a introdução da Apostila 01). Nas apostilas 02 a 05 foram estudadas as Linguagens Regulares, Livres de Contexto, Livres de Contexto Determinísticas, Sensíveis ao Contexto, Recursivas e as Recursivamente Enumeráveis compondo o sub-conjunto do universo das linguagens (Figura 01). A cada Linguagem estudada foram mostradas as Gramáticas que as geravam e os Autômatos/Máquinas de Turing que as reconheciam (Tabela 01).

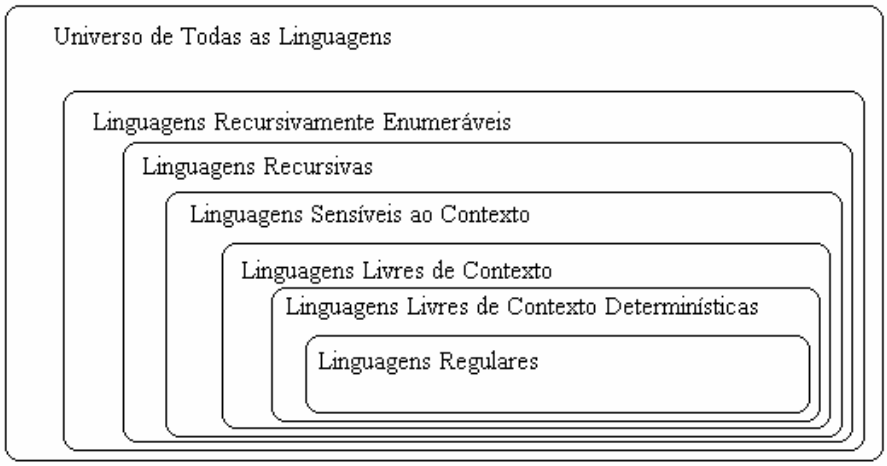

Figura 01. O Universo das Linguagens segundo Chonsky

Tabela 01. Linguagem, gramática e reconhecedor

Linguagem	**Gramática**	**Reconhecedor**
Ling Recursivamente Enumeráveis	Gramáticas Irrestritas	Máquinas de Turing
Linguagens Recursivas	Gramáticas Irrestritas	Máquinas de Turing (que sempre param)
Linguagens Sensíveis ao Contexto	Gramáticas Sensíveis ao Contexto	Autômato Limitado Linearmente
Linguagens Livres de Contexto	Gramáticas Livres de Contexto	Autômatos à Pilha
Linguagens Regulares	Gramáticas Regulares	Autômatos Finitos

Lembre-se que:

(1) Um **Autômato Finito Determinístico** é definido pela quíntupla:

$M_1 = (Q, \Sigma, \delta, q_0, F)$, onde $\delta : Q \times \Sigma \rightarrow Q$

(2) Um **Autômato Finito Não Determinístico** é definido pela quíntupla:

$M_2 = (Q, \Sigma, \delta, q_0, F)$, onde $\delta: Q \times \Sigma \rightarrow 2^Q$

(3) Um **Autômato com Pilha Não Determinístico** (APND) é definido pela sétupla:

$M_3 = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$, onde $\delta: Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \rightarrow 2^{Q \times \Gamma^*}$

(4) Um **Autômato com Pilha Determinístico**, $M_4 = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ é um Autômato com Pilha Não Determinístico sujeito as seguintes restrições, para todo $q \in Q$, $a \in \Sigma \cup \{\lambda\}$ e $b \in \Gamma$:

(a) $\delta(q,a,b)$ contém ao menos um elemento

(b) se $\delta(q,\lambda,b)$ não é vazio, então $\delta(q,c,b)$ deve ser vazio para todo $c \in \Sigma$

(5) Uma **Máquina de Turing** é definida como uma sétupla:

$M_5 = (Q, \Sigma, \Gamma, \delta, q0, F)$, onde δ: Q x $\Gamma \rightarrow$ Q x Γ x {L, R}

(6) Uma **Máquina de Turing Não Determinística**, $M_6 = (Q, \Sigma, \Gamma, \delta, q_0, F)$, é aquela em que a função de transição é definida da forma: δ: Q x $\Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$

(7) Um **Autômato Limitado Linearmente** é uma Máquina de Turing Não Determinística,

$M_6 = (Q, \Sigma, \Gamma, \delta, q_0, F)$, onde:

- Σ deve conter os símbolos [e]
- $\delta(q_i, [) = (q_j, [, R)$
- $\delta(q_i,]) = (q_j,], L)$

Perceba o grau de refinamento que existe entre essas definições. Perceba, também, que a possibilidade de haver uma máquina que reconheça uma linguagem, caracteriza a linguagem. Por exemplo, se é possível construir um Autômato Finito para uma Linguagem L, então L é regular.

Quanto às Gramáticas G = (V,T, S, P) o que muda são as formas de construção das produções:

(1) Uma gramática G é dita ser **Regular**, se A e B são elementos de V e w uma palavra de T* e as produções são da forma: A $\rightarrow$ wB ou A $\rightarrow$ w (se GLD) e A $\rightarrow$ Bw ou A $\rightarrow$ w (se GLE)

(2) Uma gramática G é dita ser **Livre de Contexto** se todas as suas produções, em P, são da forma:
$A \rightarrow \alpha$, onde $A \in V$ e $\alpha \in (V \cup T)^*$.

(3) Uma Gramática G é chamada **Irrestrita** se todas as produções são da forma: u $\rightarrow$ v, onde u $\in (V \cup T)^+$ e v $\in (V \cup T)^*$

(4) Uma Gramática Irrestrita G é **Sensível ao Contexto** se todas as produções da forma $u \rightarrow v \in P$, tem a propriedade $| u | \leq | v |$, onde u $\in (V \cup T)^+$ e v $\in (V \cup T)^*$

Após todo esse estudo das Linguagens Formais e Autômatos deve-se, neste momento, existir a preocupação com as questões pertinentes à Teoria da Computação, que são:

- Quais as capacidades e limitações fundamentais dos computadores?
- O que pode e o que não pode ser resolvido pelos computadores?
- O que faz alguns problemas serem computacionalmente mais difíceis que outros?

Ao tentar responder essas questões, exploram-se os conceitos de computabilidade, decidibilidade, redutibilidade e complexidade. Os três primeiros conceitos serão vistos aqui. A complexidade será vista em outra disciplina, de uma maneira mais formal.

2. Computabilidade e Decidibilidade

Em 1900 em um Congresso Internacional de Matemática, o matemático David Hilbert propôs 10 (dez) problemas que ainda não tinham solução, outros 13 foram publicados mais tarde. Dos 23 problemas, ainda hoje, aparecem 05 em aberto e 02 parcialmente resolvidos.

O que é interessante, para nós, é o décimo: "Encontrar um algoritmo que determine se uma equação diofantina tem solução?" Uma equação diofantina é aquela que se procuram soluções inteiras (e às vezes racionais) para equações algébricas do tipo: $x^2 + y^2 = z^2$, $x^n + y^n = z^n$, dentre outras.

Em 1931 o Teorema de Incompleteza de Godel demonstrou que a mecanização do processo de prova de teorema não tinha solução. Assim, provou a incompleteza da Lógica de Primeira Ordem, mas não fez referência à computação.

Church e Turing trouxeram ferramentas básicas para definir algoritmos de uma forma precisa (Cálculo Lambda e Máquina de Turing) e provar a indecidibilidade da validade lógica de primeira ordem. Só em 1970, Yuri Matijasevic provou a impossibilidade de solução geral para o 10º. Problema de Hilbert.

Para estudar os limites da computação é necessário escolher um modelo formal de algoritmo. Na literatura podem ser vistos os modelos: Funções Recursivas, Algoritmos de Markov, Cálculo Lambda, Máquinas de Registros, Sistemas Post, Máquina de Turing (vista na apostila 04 – TC04.pdf), etc.

A Máquina de Turing é suficientemente simples e ao mesmo tempo poderosa. A Tese de Turing e a Hipótese de Church evidenciam isso.

Tese de Turing
"Qualquer computação que pode ser executada por meios mecânicos pode ser executada por uma Máquina de Turing"

Hipótese de Church
"A capacidade de computação representada pela Máquina de Turing é o limite máximo que pode ser atingido por qualquer dispositivo de computação"

Lembre-se que:

(a) Pode existir uma linguagem que não é reconhecida pela Máquina de Turing.
(b) Se a linguagem é reconhecida pela Máquina de Turing então ela é uma Linguagem Recursivamente Enumerável.
(c) Se a Máquina de Turing sempre pára quando aceita ou rejeita uma cadeia de uma linguagem então a Linguagem é Recursiva.

Sabe-se que, pelas definições 03, 04 e 05 da Apostila 04:

(1) Uma Máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ reconhece ou aceita uma linguagem se para toda cadeia $w \in L$, a Máquina de Turing aceita a cadeia, ou seja,

$$q_0 w \vdash^* x_1 q_f x_2 \text{ para algum } q_f \in F \text{ e } x_1, x_2 \in \Gamma^*$$

Se $w \notin L$, a Máquina de Turing pode parar num estado não final ou entrar em loop.
Diz-se que essa Linguagem é aceita ou reconhecida por essa Máquina de Turing.

(2) Uma Máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ decide uma linguagem se para toda cadeia $w \in L$ a Máquina de Turing aceita a cadeia e pára, com $w \notin L$, num estado não final.
Diz-se que essa Linguagem é decidida por essa Máquina de Turing.

(3) Uma Máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ calcula uma função f(w) se para $w \in \Sigma^+$, a Máquina de Turing pára e a saída é a cadeia f(w), ou seja,

$$q_0 w \vdash^*_M q_f f(w), \quad q_f \in F$$

Diz-se que essa função é computável por essa Máquina de Turing.

Assim,

- se a Máquina de Turing M calcula uma função num certo domínio, a função é computável;
- se a Máquina de Turing M sempre pára com a resposta SIM se a cadeia pertence a Linguagem e com a resposta NÃO quando a cadeia não pertence a Linguagem, então tem a decidibilidade desse domínio, ou seja a MT é decidível
- se a Máquina de Turing pára com a resposta SIM se a cadeia pertence a linguagem e nem sempre pára quando a cadeia não pertence a Linguagem, então a linguagem é somente aceita (e não decidível).

Considerações:

- **Procedimento** é uma seqüência finita de instruções, sendo uma instrução uma operação claramente descrita, que pode ser executada mecanicamente, em tempo finito.

- **Algoritmo** são procedimentos cuja execução termina para quaisquer valores dos dados de entrada.

- Uma função é dita computável se existir um procedimento que a compute.

- Uma Linguagem de programação é a maneira mais conveniente para a representação de procedimentos no computador.

- A abordagem operacional de Turing permitiu a Von Neuman elaborar o modelo para construção de máquinas digitais.

- A abordagem lingüística de Chomsky permitiu o desenvolvimento de teorias para concepção das linguagens de programação.

Exemplo 01 – Procedimento pára e diz sim se o número inteiro i, dado como entrada, for par e não negativo; pára e diz não nos demais casos:

1. se i = 0 pare e diga sim
2. se i < 0 pare e diga não
3. diminua o valor de i de duas unidades
4. vá para o passo 1

Assim, temos um algoritmo neste exemplo.

Exemplo 02 – procedimento para determinar o menor número perfeito que é maior do que um inteiro positivo m dado (k é perfeito se for igual a soma de todos os seus divisores, exceto o próprio k). Apenas para certos valores de m é possível, já que é a existência ou não de um número infinito de números perfeitos é um problema em aberto.
Assim, temos um procedimento neste exemplo.

Uma outra visão: Estudo através da Solucionabilidade de Problemas

Um problema é solucionável, ou totalmente solucionável, se existe um algoritmo que resolva o problema para qualquer entrada informando uma resposta de aceitação ou rejeição (resposta negativa). Um problema é dito não solucionável se não existe um algoritmo que sempre resolva o problema.

Um problema parcialmente solucionável é aquele que existe um algoritmo que solucione o problema e informa a resposta se ela é afirmativa. Se for negativa, o algoritmo pode parar ou permanecer processando indefinidamente.

Figura 2. Relação entre as classes de problemas.
(Cap5. Computabilidade – Profs Divério e Menezes)

De posse desses conceitos pode-se dizer que:

- Um problema é computável se o problema é solucionável ou parcialmente solucionável
- Um problema é não computável se o problema é não solucionável.
- A união da classe dos problemas solucionáveis com classe dos problemas não solucionáveis é o universo de todos os problemas.
- A classe dos problemas solucionáveis corresponde à classe das linguagens recursivas.
- A classe dos problemas parcialmente solucionáveis corresponde à classe das linguagens enumeráveis recursivas.
- A classe dos problemas não solucionáveis é não computável.

3. Problemas Indecidíveis

Um problema é decidível se sua solução é encontrada num tempo finito, ou seja, existe uma Máquina de Turing que retorna uma resposta. Caso contrário ele é indecidível. O conceito de decidibilidade não trata a quantidade de tempo gasto e sim se ele é finito (o conceito que trabalha com a quantidade de tempo gasto é o da Complexidade).

Pode-se, ainda, ver a diferença entre decidível e não decidível com relação à linguagem. Se o problema pode ser representado por uma linguagem recursiva (onde a Máquina de Turing sempre pára) então o problema é decidível.

(A) Problema da parada

Dada uma Máquina de Turing M genérica e uma entrada genérica w, não existe uma Máquina de Turing M' capaz de decidir se M executa uma computação que aborta (pára), quando começa o processamento numa configuração inicial q_0w.

Comentário: Se o problema da parada fosse decidível então toda Linguagem Recursivamente Enumerável deveria ser uma Linguagem Recursiva. Conseqüentemente o problema da parada é indecidível.

(B) Problema de entrada num estado

Dada uma Máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ e qualquer $q \in Q$ e $w \in \Sigma^+$, decidir se o estado q é sempre visitado quando a Máquina de Turing é aplicada a w não é possível. Esse problema é indecidível.

Comentário: Poder-se-ia construir uma Máquina de Turing M' que sempre pára quando entra no estado q. Assim ter-se-ia a garantia de que a Máquina de Turing entrava no estado q. Só que o problema da parada é indecidível, então o problema da entrada num estado também é indecidível.

(C) Problema da parada numa fita em branco

Dada uma Máquina de Turing M, determinar se M pára se começar com uma fita em branco é indecidível.

Comentário: Poder-se-ia construir uma Máquina de Turing M_w que começa com a fita em branco, escreve w nela e se posiciona na configuração q_0w. Fica claro que M_w pára na fita em branco se, e somente se, M aceita a cadeia w, ou seja, pára com a entrada w. Só que o problema da parada é indecidível, então esse também é.

(D) Linguagem gerada por uma Gramática Irrestrita é nula

Seja G uma Gramática Irrestrita. Então o problema de determinar se $L(G) = \varnothing$ é indecidível.

Comentário: Supondo a existência uma Máquina de Turing M e uma cadeia w, tal que M salva sua entrada na fita. Então M pára se ao entrar no estado final, w está gravada na fita. Suponha M_w uma Máquina de Turing para cada w, tal que $L(M_w) = L(M) \cap \{w\}$.
Supondo L_w e L, duas Linguagens Recursivamente Enumeráveis, é possível construir G_w tal que $L_w = L(G_w)$ e $L = L(G)$.
$L(G_w) = \varnothing$ se, e somente se, $w \notin L(G)$. Suponha que exista um algoritmo para determinar se $L(G_w) = \varnothing$. Então existirá uma Máquina de Turing que para qualquer M e w seja possível determinar se $w \in L(M)$. Se existe essa Máquina de Turing então a Linguagem aceita é uma Linguagem Recursivamente Enumerável, o que contradiz resultados anteriores. Então $L(G) = \varnothing$, com G sendo uma Gramática Irrestrita, não é decidível.

(E) Linguagem gerada por uma Máquina de Turing é finita

Seja M uma Máquina de Turing. Então a questão, se L(M) é finita, é indecidível.

Comentário: Como o problema da parada não é decidível então saber se o conjunto das cadeias geradas pela Máquina de Turing é finito não é possível. Então esse problema é indecidível.

Teorema de Rice:
Qualquer propriedade de linguagens reconhecidas por Máquinas de Turing é indecidível.

Uma propriedade sobre uma Linguagem Recursivamente Enumerável (aceita por uma Máquina de Turing) é o conjunto de Linguagens Recursivamente Enumeráveis, ou seja, a propriedade livre de contexto é o conjunto das Linguagens Livres de Contexto

Então se pode dizer que, sendo M é uma Máquina de Turing:
Se L = L(M), decidir se L é uma Linguagem Regular é indecidível.
Se L = L(M), decidir se L é uma Linguagem Livre de Contexto é indecidível.
Se L = L(M), decidir se L é uma Linguagem Sensível ao Contexto é indecidível.
Se L = L(M), decidir se L é uma Linguagem Recursiva é indecidível.

4. Redutiblidade

Uma redução é um modo de converter um problema em outro de tal modo que a solução para o segundo pode ser utilizada para resolver o primeiro.

Reduzir um problema A para um problema B usando uma redução por mapeamento significa que existe uma função computável que converte instâncias do problema A para instâncias do problema B. Se uma tal conversão (função de) existir, chamada de uma redução, pode-se resolver A com um resolvedor (solucionador) para B.

Definição 02:
Uma linguagem A é redutível por mapeamento para uma linguagem B, $A \leq B$, se existe uma função computável $f: \Sigma^* \rightarrow \Sigma^*$ onde para todo w,

$$w \in A \Leftrightarrow f(w) \in B$$

A função f é chamada redução de A para B.

Teorema 01:
Se A redutível a B ($A \leq B$) e B é decidível então A é decidível.

Teorema 02:
Se A redutível a B ($A \leq B$) e A é indecidível então B é indecidível.