TEORIA DA COMPUTAÇÃO
1ª Prova - Resumo

Nikolas Machado Corrêa

Introdução

Conceitos:

1. **Algoritmo:** descrição finita de passos; método que sempre produz um resultado.
2. **Computabilidade:** formalizar o conceito de algoritmo sem fazer referência a linguagens ou dispositivos físicos.

Turing e Church criaram dispositivos para modelar o processo de computação; utilizados para expressar algoritmos.

Funções parciais: processos que nem sempre levam a um resultado; funciona para alguns casos. Cada programa define uma função parcial.

Modelos de Computação: há problemas para os quais nenhum programa computacional pode fornecer uma solução; para as funções computáveis, a Teoria da complexidade estuda os recursos necessários para tais computações. Considerando memória e tempo ilimitados, todas as funções que podem ser efetivamente computadas são funções computáveis.

Máquina de Turing ↔ Cálculo Lambda ↔ Teoria de funções recursivas
(os três modelos expressam mesma classe de funções)

Turing Completa: linguagens que conseguem resolver qualquer função computável; prova-se através de um modelo universal;

Funções não-computáveis → Problema da Parada:

Dadas uma descrição de um algoritmo A e uma entrada finita I, decida se o programa termina de rodar ou rodará infinitamente, testando-o com um algoritmo H.

H = 1, se A parar em I
H = 0, se A não parar

Utilizando um programa C, para qualquer programa A, as seguintes propriedades valem:

Se H(A, A) = 1, então C(A) não para e A(A) para
Se H(A, A) = 0, então C(A) para e A(A) não para

Como A pode ser o próprio C, chega-se à contradição:
C(C) para se, e somente se, C(C) não para. Assim, H não existe.

Máquinas de Turing (MT)

Semelhante a autômatos finitos, porém com memória ilimitada; pode fazer tudo o que um computador também consegue.

Definição formal → Uma máquina de Turing é uma 7-upla: ***(Q, Σ, Γ, σ, q0, q_{aceita}, $q_{rejeita}$)***

Q = conjunto estados
Σ = alfabeto entrada (sem branco)
Γ = alfabeto fita (com o branco)
σ = funções de transição
q0, q_{aceita}, $q_{rejeita}$ = estado inicial, de aceitação e de rejeição, respectivamente.

Configuração de uma MT: possível valor para o estado atual, conteúdo da fita e posição da cabeça de leitura.

Exemplo: 1011**q7**01111

1. Fita: 101101111
2. Estado: q7
3. Cabeça: segundo 0

Linguagem Turing-reconhecível ou recursivamente enumerável

Se alguma MT a reconhece; cadeias aceitas são chamadas de linguagem reconhecida por M → L(M). Podem **aceitar**, **rejeitar** ou entrar em **loop**, nunca chegando a um estado de parada. (M aceita w: se uma sequência de configurações Ci (i = 1,2,3,...,k) levam a uma configuração Ck de aceitação.)

Linguagem Turing-decidível ou linguagem recursiva

Se alguma máquina de Turing a decide, isto é, sempre aceitam ou rejeitam.

Toda linguagem Turing-decidível é Turing-reconhecível porém
Algumas linguagens Turing-reconhecíveis não são decidíveis

Reconhecedores versus Decisores: Uma máquina M é um reconhecedor se reconhece L(M). Se, além disso, M nunca entra em loop infinito, então M é dito um decisor.

Variantes da MT: supondo que a cabeça de uma MT possa ficar parada, sem necessidade de mover-se à esquerda ou direita, essa MT pode ser convertida. Para mostrar a equivalência de dois modelos, basta simular um pelo outro.

Máquina de Turing Multifita

É como uma comum, com várias fitas, tendo cada uma sua própria cabeça de leitura e escrita. São equivalentes se reconhecem as mesmas linguagens.

Teorema: *Toda MT multifita tem uma MT de uma única fita que lhe é equivalente.*

Prova: *usa-se uma MT de única fita S para simular uma MT multifita M. Para cada fita de M, S usa um delimitador #. Usa-se um ponto • para marcar onde estaria a cabeça em cada fita de M.*

Teorema: *Uma linguagem é Turing-reconhecível se e somente se uma MT multifita a reconhece.*

Prova: uma linguagem Turing-reconhecível é reconhecida por uma MT comum, que é um caso da MT multifita. A outra direção é dada pelo teorema anterior.

Máquinas de Turing Não-Determinísticas

Aquelas em que em qualquer ponto em uma computação, a máquina pode proceder de acordo com várias possibilidades. A computação é uma árvore, cujos ramos expressam as diferentes possibilidades. Se uma ramo leva a um estado de aceitação, a MT aceita.

Teorema: *toda MT Não-determinística possui uma determinística que é equivalente.*

Prova: *simulando através de uma MT determinística, testando todos os possíveis ramos a fim de encontrar um nó de aceitação; cada configuração é um nó.*

OBS: busca em largura explora todos os ramos na mesma profundidade, garantindo que a MT determinística visite todo o nó até encontrar um que aceite.

Prova formal: usando 3 fitas;

1. 1ª contém a entrada e não é alterada
2. 2ª mantém uma cópia de um dos ramos da 1ª
3. 3ª é a fita de endereço, guardando a posição

Uma linguagem é decidível se e somente se alguma máquina de Turing não-determinística a decide.

Enumeradores

Podem ser vistos como uma MT com uma impressora anexa; a linguagem enumerada por um enumerador E é a coleção de cadeias que ela imprime.

Teorema: *uma linguagem é Turing-reconhecível se somente se um enumerador a enumera.*

Prova: *dado um enumerador E que enumere A, uma MT reconhece A. Assim, basta rodar E e comparar as cadeias de saída com w; se w aparece em E, M aceita as cadeias que foram impressas em E.*

Outra direção: *se M reconhece A, pode-se construir um E para A. Assim, roda-se M sobre cada entrada; se computações são aceitas, imprime-se a saída correspondente. Dessa forma, M aceita uma cadeia que, em algum momento, aparecerá em E.*

Tipos de Provas

1. Por construção: teoremas que enunciam que um tipo particular de objeto existe;
2. Por contradição: assume-se que o teorema é falso e mostra-se que isso leva a uma consequência falsa, isto é, uma contradição;
3. Por indução: para mostrar que todos os elementos de um conjunto têm uma propriedade específica. Toda prova por indução é composta pela *base* e o *passo da indução.*

Decidibilidade - Descrevendo MTs de alto-nível

1. $A = \{\langle G \rangle \| G \text{ eh um grafo nao-direcionado conexo}\}$

- M = "Sobre a entrada $\langle G \rangle$, a codificação de um grafo G:
  1. Selecione o primeiro nó de G e marque-o.
  2. Repita o seguinte estágio até que nenhum novo nó seja marcado:
  3. Para cada nó em G, marque-o, se ele estiver ligado por uma aresta a um nó que já está marcado.
  4. Faça uma varredura em todos os nós de G para determinar se eles estão todos marcados. Se estiverem, *aceite*, caso contrário, *rejeite*."

2. $A_{AFD} = \{\langle B, w \rangle | B \text{ eh um AFD que aceita a cadeia de entrada } w\}$.

- **Teorema**: A_{AFD} é uma linguagem decidível.
- **Prova**: Simplesmente precisamos apresentar uma MT, M, que decide A_{AFD}.
- "Sobre a entrada $\langle B, w \rangle$, onde B é um AFD, e w é uma cadeia:
  1. Simule B sobre a entrada w.
  2. Se a simulação termina em um estado de aceitação, *aceite*. Se ela termina em um estado de não-aceitação, *rejeite*."

3. $A_{AFN} = \{\langle B, w\rangle | B \text{ eh um AFN que aceita a cadeia de entrada } w\}$

- **Teorema**: Todo autômato finito não-determinístico (AFN) tem um autômato finito determinístico (AFD) equivalente.
- **Ideia da prova**: Se uma linguagem é reconhecida por um AFN, então temos de mostrar a existência de um AFD que também a reconhece.

4. $A_{EXR} = \{\langle R, w\rangle | R \text{ eh uma expressão regular que gera uma cadeia } w\}$

- **Teorema**: A_{EXR} é uma linguagem decidível.
- A prova do teorema anterior usa o seguinte teorema:
- **Teorema**: Toda expressão regular tem um autômato finito não-determinístico (AFN) equivalente.
- **Ideia da prova**: converta a expressão regular R para um AFN equivalente.

5. $V_{AFD} = \{\langle A\rangle | A \text{ eh um AFD e } L(A) = \emptyset\}$

- **Prova**: Um AFD aceita alguma cadeia sse é possível atingir um estado de aceitação a partir do estado inicial passando pelas setas do AFD. Para testar essa condição, podemos projetar uma MT T, que usa um algoritmo de marcação similar aquele usado no exemplo do grafo direcionado conexo.
- T="Sobre a entrada $\langle A\rangle$, onde A é um AFD:
  1. Marque o estado inicial de A.
  2. Repita até que nenhum estado novo venha a ser marcado:
  3. Marque qualquer estado que tenha uma transição chegando nele a partir de qualquer estado que já está marcado.
  4. Se nenhum estado de aceitação estiver marcado, aceite; caso contrário, rejeite."

6. $EQ_{AFD} = \{\langle A, B\rangle | A \text{ e } B \text{são AFDs e } L(A) = L(B)\}$

- **Prova**: Para provar esse teorema usamos o teorema anterior da vacuidade. Construimos um novo AFD C a partir de A e B, tal que C aceita aquelas cadeias que são aceitas ou por A ou por B, mas não por ambos. Consequentemente, Se A e B reconhecem a mesma linguagem, C não aceitará nada.

- A linguagem C é:

$$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

7. $A_{GLC} = \{\langle G, w\rangle | G \text{ é uma GLC que gera a cadeia } w\}$.

- S="Sobre a entrada $\langle G, w\rangle$, onde G é uma GLC e w uma cadeia:
  1. Converta G para uma gramática equivalente na forma normal de Chomsky.
  2. Liste todas as derivações com $2n - 1$ passos, onde n é o comprimento de w.
  3. Se alguma das derivações gera w, aceite; se não, rejeite."