Teoria da Computação

Resumo

Conteúdo

Capítulo 1

Teoria dos Autómatos

1.1 Autómatos Finitos Deterministas

Um autómato finito determinista é um 5-tuplo $M = (Q, \Sigma, \delta, q_0, F)$, em que:

- Q é um conjunto finito de estados
- Σ é o alfabeto (conjunto finito de símbolos)
- $\delta : Q \times \Sigma \to Q$ é a função de transição
- $q_0 \in Q$ é o estado inicial
- $F \subseteq Q$ é o conjunto dos estados de aceitação

Chama-se *palavra* sobre Σ a uma sequência finita de símbolos de Σ e *linguagem* a um conjunto de palavras formadas com símbolos de um alfabeto Σ. O conjunto de todas as palavras formadas sobre Σ é o fecho do alfabeto, Σ^*.

1.1.1 Computação por um AFD

Seja $M = (Q, \Sigma, \delta, q_0, F)$ um AFD e $w = w_0 w_1 ... w_n$ uma string de Σ^*.

Diz-se que M aceita w se, e somente se, existe uma sequência de estados $r_0, ..., r_n, r_{n+1} \in Q$ tal que:

- $r_0 = q_0$ a computação inicia-se no estado inicial
- $\delta(r_i, w_i) = r_{i+1} \forall i \in \{0, ..., n\}$ a computação segue de estado para estado de acordo com a função de transição
- $r_{n+1} \in F$ termina num estado de aceitação

Linguagem Reconhecida por um AFD

A linguagem reconhecida por um AFD M define-se como sendo:

$$L(M) = \{w \in \Sigma^* : M \text{ aceita } w\}$$

Uma linguagem diz-se regular se existe um AFD que a reconhece.

1.2 Operações e Expressões Regulares

1.2.1 Operações Regulares

Sejam A e B duas linguagens. Definem-se as seguintes operações regulares:

- União: $A \cup B = \{x | x \in A \vee x \in B\}$
- Concatenação: $A.B = \{x.y | x \in A \vee x\ y \in B\}$
- Estrela (Fecho de Kleene): $A^* = \{x_1.x_2...x_k | k \geq 0 \wedge x_i \in A, 1 \leq i \leq k\}$

Definição de Fecho

Diz-se que um conjunto é fechado sob uma operação n-ária se ao ser aplicada a qualquer n-uplo de elementos do conjunto se obtém um elemento do conjunto.

As linguagens regulares são fechadas para todas as operações regulares.

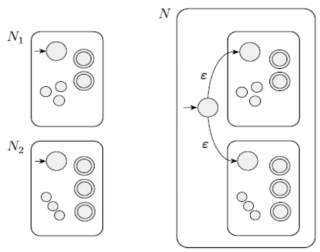

Figura 1.1: Fecho da União

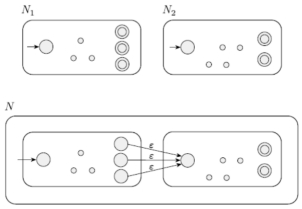

Figura 1.2: Fecho da Concatenação

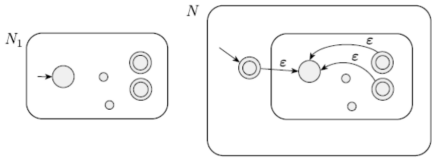

Figura 1.3: Fecho da Estrela

1.2.2 Expressões Regulares

R é uma expressão regular sobre um alfabeto Σ se é:

- a, para algum $a \in \Sigma$
- ε
- ϕ
- $(R_1.R_2)$, onde R_1 e R_2 são expressões regulares
- $(R_1.R_2)$, onde R_1 e R_2 são expressões regulares

- $(R_i)^*$, onde R_i é uma expressão regular

Cada expressão regular R denota uma linguagem regular: a linguagem $L(R)$.

As operações nas expressões regulares têm a seguinte ordem de prioridade:

- Estrela
- Concatenação
- União

Uma linguagem é regular se e só se alguma expressão regular a denota.

1.3 Autómatos Finitos Não Deterministas

Um autómato finito não determinista é um 5-tuplo $M = (Q, \Sigma, \delta, q_0, F)$, em que:

- Q é um conjunto finito de estados
- Σ é o alfabeto (conjunto finito de símbolos)
- $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ é a função de transição, em que $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$
- $q_0 \in Q$ é o estado inicial
- $F \subseteq Q$ é o conjunto dos estados de aceitação

Os AFND diferem dos AFD na função de transição. Nos AFND esta permite ε como input e o resultado da sua aplicação é um conjunto de estados de Q, i.e. o conjunto das partes de Q.

1.3.1 Computação por um AFND

Seja $M = (Q, \Sigma, \delta, q_0, F)$ um AFND e $s' = s'_1...s'_m$ uma string sobre Σ. M aceita s' se existe uma sequência de estados $a_0, a_1, ..., a_n$ e s' pode ser escrita como $s = s_1 s_2 ... s_n$ em que $s_i = s'_j$ ou $s_i = \varepsilon$ de tal modo que:

- a_0 é o estado inicial da máquina
- $a_{i+1} \in \delta(a_i, s_{i+1} \forall i \in \{0, ..., n-1\}$
- $a_n \in F$

Fecho-ε

Define-se por fecho épsilon de um estado q o conjunto de todos os estados atingíveis a partir de q por um caminho de zero ou mais transições épsilon. Representa-se por $E_\varepsilon(q)$. Se R é um conjunto de estados, define-se:

$$E_\varepsilon(R) = \bigcup_{q \in R} E_\varepsilon(q)$$

1.3.2 Equivalência entre autómatos

Dois autómatos dizem-se equivalentes se reconhecem a mesma linguagem.
Cada AFND tem um AFD equivalente.

Equivalência entre AFND e AFD

Seja $N = (Q, \Sigma, \delta, q_0, F)$ um AFND. É possível construir um AFD $M = (Q', \Sigma, \delta', q_0', F')$ tal que $L(N) = L(M)$.

- $Q' = \mathcal{P}(Q)$
- $q_0' = E_\epsilon(q_0)$, i.e. o conjunto dos estados atingíveis por caminhos vazios a partir do estado inicial.
- $\forall q' \in Q$ considere-se $R \subset Q : R = q'$. Para cada $a \in \Sigma$ definimos $\delta'(q', a) = \delta'(R, a) = \bigcup_{r \in R} E_\varepsilon(\delta(r, a)) = \{q \in Q : \exists r \in R, q \text{ atingível por transição } \varepsilon \text{ de } \delta(r, a)\}$
- $F' = \{R \in Q' : R \text{ contém algum estado de aceitação de } N\}$

1.3.3 Autómatos Finitos Não Deterministas Generalizados

Um autómato finito não determinista generalizado é um 5-tuplo $M = (Q, \Sigma, \delta, q_{start}, q_{accept})$ onde:

- Q é um conjunto de estados
- Σ é o alfabeto de entrada
- $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to R$ é a função de transição, em que R é a coleção de todas as expressões regulares definidas sobre Σ
- q_{start} é o estado inicial
- q_{accept} é o estado de aceitação

Um AFNG possui expressões regulares nas etiquetas das transições, por oposição aos símbolos isolados, podendo numa só transição consumir a palavra inteira do input. Num AFNG:

- O estado inicial tem transições para todos os outros estados, mas não recebe nenhuma transição a partir de outro estado, nem de si próprio.
- Só há um estado de aceitação, que recebe transições de todos os outros estados, mas não tem transições a sair.
- Com exceção do estado inicial e do estado de aceitação há uma transição de cada estado para todos os estados, incluindo o próprio.

Computação Por Um AFNG

Se um AFNG aceita uma palavra $w = w_1w_2...w_k$ sobre Σ então existe uma sequência de estados $q_0, q_1...q_k$, tal que:

- $q_0 = q_{start}$ é o estado inicial
- $q_k = q_{accept}$ é o estado de aceitação
- $\forall w_i, w_i \in L(R_i)$ onde $R_i = \delta(q_{i-1}, q_i)$, i.e. R_i é e expressão regular da transição de q_{i-1} para q_i.

Converter AFD/AFND em AFNG

- Adicionar um novo estado inicial q_{start} com uma transição-ε para o estado inicial original.
- Adicionar um único estado de aceitação q_{accept} e transições-ε de todos os estado de aceitação atuais para q_{accept}
- Se existem transições com múltiplas etiquetas ou múltiplas transições entre dois estados, substituir cada uma por uma única transição etiquetada com a união das etiquetas originais.
- Adicionar transições etiquetadas com ϕ entre pares de estados para os quais não havia transições, exceto para q_{start} e q_{accept}.

Converter AFNG em Expressão Regular

Seja G um AFNG e k o seu número de estados. Se $k = 2$ então a ER é a etiqueta de q_{start} para q_{accept}. Caso contrário executa-se o seguinte algoritmo até $k = 2$:

- Seleciona-se um estado q_{rip}, diferente de q_{start} e q_{accept}
- Determina-se todos os pares (q_i, q_j) para os quais há um caminho $q_i \rightarrow q_{rip} \rightarrow q_j$
- Remove-se q_{rip}
- Adicionam-se as transições $q_i \rightarrow q_j$, etiquetadas de acordo com o seguinte esquema:

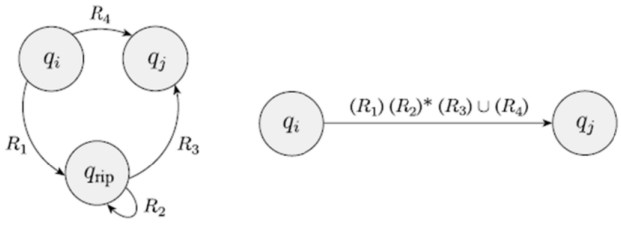

Figura 1.4: Remoção de estados em AFNG

1.4 Lema do Bombeamento Para Linguagens Regulares

O lema do bombeamento para linguagens regulares é usado para demonstrar que uma linguagem não é regular. Apesar de necessário, não é condição suficiente para mostrar que uma linguagem é regular, i.e. todas as linguagens regulares verificam o lema do bombeamento, mas algumas linguagens não regulares também.

1.4.1 Enunciado

Seja L uma linguagem regular infinita. Existe $p \geq 1$, tal que cada palavra $w \in L, |w| \geq p$ pode ser decomposta em $w = xyz$ tal que:

- $|y| \geq 1$
- $|xy| \leq p$
- $\forall i \geq 0, xy^i z \in L$

1.5 Gramáticas Livres de Contexto

Uma gramática é um 4-tuplo $G = (V, \Sigma, R, S)$ onde:

- V é o conjunto de variáveis não terminais
- Σ é o alfabeto de símbolos terminais ($\Sigma \cap V = \emptyset$)
- $R \subseteq (V \times (V \cup \Sigma)^*)$ é o conjunto de regras de produção
- $S \in V$ é o símbolo inicial

Uma gramática diz-se livre (ou independente) do contexto quando todas as suas produções são da forma:

$$A \to \beta \text{ onde } A \in V \wedge \beta \in (V \cup \Sigma)^*$$

Se todas as produções de uma GIC forem da forma:

$$A \to \alpha X$$

$$A \to X\alpha$$

$$A \to \alpha \text{ onde } A, X \in V, \alpha \in \Sigma^*$$

Então só tem produções lineares e gera uma linguagem regular.

1.5.1 Ambiguidade

Uma palavra w deriva-se ambiguamente numa GIC G se existirem duas ou mais derivações esquerdas de w em G. Também existem duas árvores de derivação diferentes e duas derivações direitas.

Uma gramática diz-se ambígua se derivar alguma string w ambiguamente.

1.5.2 Gramática Reduzida

Uma GIC diz-se reduzida se não tem símbolos inúteis, isto é, que são improdutivos ou inacessíveis. Para obter a GIC reduzida eliminam-se as variáveis improdutivas e de seguida os símbolos inacessíveis.

Marcar Variáveis Produtivas

Para marcar as variáveis produtivas usa-se o algoritmo PROD:

- Marcar a variável $A \in V$ que tenha alguma produção $A \to x$, com $x \in \Sigma^*$
- Marcar todo o símbolo $A \in V$ que tenha alguma produção $A \to X_1X_2...X_k$ onde X_i são variáveis marcadas até não ser possível marcar mais nenhuma variável.

Marcar Símbolos Acessíveis

Para marcar os símbolos acessíveis usa-se o algoritmo ACESS:

- Marcar o símbolo inicial S
- Marcar todo o símbolo X (terminal ou variável) que ocorra no lado direito de alguma produção onde o membro esquerdo esteja já marcado, até não ser possível marcar mais nenhum símbolo.

1.5.3 Gramática Livre-ε

Uma produção-ε é da forma:

$$A \to \varepsilon, A \in V$$

Um símbolo $A \in V$ diz-se *nulável* se pode derivar em ε.

Uma GIC G diz-se livre-ε se não tem produções ε com exceção de $S \to \varepsilon$ e S não ocorre no membro direito de outras produções.

Marcar Símbolos Nuláveis

Para marcar os símbolos nuláveis usa-se o algoritmo NUL:

- Marcar todo o símbolo $A \in V$ que tenha alguma produção $A \to \varepsilon$
- Marcar todo o símbolo $A \in V$ que tenha alguma produção $A \to X_1X_2...X_k$ onde X_i são variáveis já marcadas, até não ser possível marcar mais nenhum símbolo.

Converter em Gramática Livre-ε

Dada uma GIC $G = (V, \Sigma, R, S)$ existe uma GIC livre-ε equivalente a G:

- Determinar os símbolos nuláveis de G
- Para cada produção $A \to X_1X_2...X_k$:

  - Para cada X_i nulável introduzimos uma produção em que concretizamos X_i em ε
  - Para cada combinação de vários X_i nuláveis na mesma produção, introduzimos uma produção em que concretizamos esses X_i em ε.
- Eliminamos todas as produções ε
- Se S é nulável, introduzimos S_0 com as produções $S_0 \to S$ e $S_0 \to \varepsilon$

1.5.4 Produções Singulares

Uma produção diz-se singular se for da forma $A \to B$, com $A, B \in V$. Dada uma gramática livre-ε é possível encontrar e eliminar as produções singulares.

Algoritmo SING-A

O algoritmo SING-A permite marcar todos os símbolos atingíveis por produções singulares a partir de A, numa gramática livre-ε. Executa-se SING-A para cada $A \in V$:

- Marcar o símbolo A
- Marcar cada $X \in V$ que ocorra em alguma produção singular onde o membro esquerdo esteja já marcado, até que não seja possível marcar mais variáveis.

Converter em GIC sem Produções Singulares

Para obter uma GIC livre-ε sem produções singulares:

- $\forall A \in V$ determinar os símbolos deriváveis por produções singulares a partir de A, através de SING-A
- $\forall X \in$ SING-$A, X \in V$ introduzir produções a partir de A, com os membros direitos não singulares que saiam de X
- Cortar todas as produções singulares

1.5.5 Forma Normal de Chomsky

Uma GIC está na forma normal de Chomsky (FNC) sse todas as suas produções estão numa das formas:

- $A \to BC, A, B, C \in V$
- $A \to a, A \in V, a \in \Sigma$
- $S \to \varepsilon$, S é o axioma e S não ocorre no membro direito de outras produções

Conversão para FNC

Dada uma GIC reduzida, livre-ε e sem produções singulares:

- Substituem-se regras onde o membro direito tem mais que 2 símbolos:

  $$A \to u_1 u_2 ... u_k, k \geq 3, u_i \text{ é um terminal, substitui-se por:}$$

  $$A \to u_1 A_1; ...; A_{k-1} \to u_{k-1} u_k \text{ onde } A_i \text{ são novas variáveis}$$

- Eliminam-se regras onde o membro direito tem terminais e variáveis. Para cada u_i terminal no membro direito de uma produção com comprimento 2:
  - Juntar uma produção $U_i \to u_i, U_i$ nova variável
  - Substituir u_i por U_i nos membros direitos de produções que incluam terminais e variáveis

1.6 Autómatos de Pilha

Um autómato de pilha é um 6-tuplo $(Q, \Sigma, \Gamma, \delta, q_0, F)$ onde:

- Q é um conjunto finito de estados
- Σ é o alfabeto de entrada
- Γ é o alfabeto da pilha
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ é a função de transição
- $q_0 \in Q$ é o estado inicial
- $F \subset Q$ é o conjunto de estados de aceitação

Geralmente, o símbolo \$ é usado para denotar o fundo da pilha.

1.6.1 Função de transição num Autómato de Pilha

$\delta(q, a, \alpha) = \{(q', u), ...\}$ é a função de transição não determinista, interpretada da seguinte forma:

Estando no estado q, por leitura do símbolo $a \in \Sigma_\varepsilon$ e estando a ver $\alpha \in \Gamma_\varepsilon$ no topo da pilha.
O AP transita para uma configuração onde o estado é q' e o topo da pilha foi substituído pela palavra u:

- Se $u = \varepsilon$ fez *pop* do topo da pilha
- Se $u = \alpha$ a pilha ficou igual
- Se $u = \beta\alpha$ fez *push* de β
- Caso contrário, α foi substituído por u

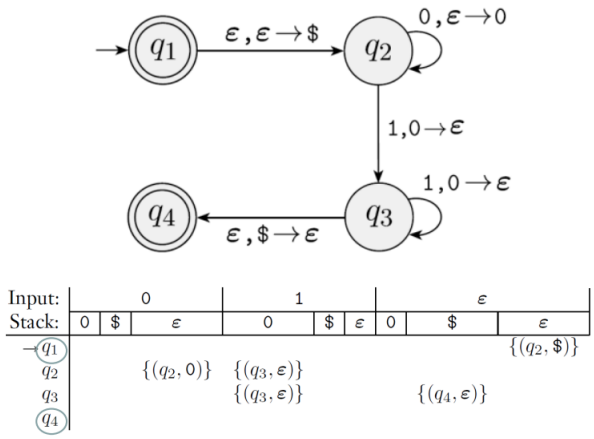

Figura 1.5: Exemplo de autómato que aceita a linguagem $\{0^n 1^n | n \geq 0\}$. A etiqueta num arco $a, b \to c$ significa ler a na fita, ver b no topo da pilha, substituir b por c

1.6.2 Configuração de um Autómato de Pilha

Um AP está numa configuração (q, w, u) se está no estado q, w é o conteúdo por consumir na fita e u é o conteúdo da pilha.

$(q, aw, \alpha v) \vdash (q', w, \beta v)$ por leitura de a a partir de q, mudou de configuração para o estado q' e substituiu α por β no topo da pilha.

$\vdash^*$ significa zero ou mais mudanças de configuração.

1.6.3 Computação por um Autómato de Pilha

Um AP $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ aceita um input w se w pode ser escrito como $w_1 w_2 ... w_m, w_i \in \Sigma_\varepsilon$ e existe uma sequência de estados $r_0, r_1, ..., r_m \in Q$ e uma sequência de palavras em Γ^* $s_0, s_1, ..., s_m$ de forma a:

- $r_0 = q_0$ e $s_0 =$ ", M inicia no estado q_0 com a pilha vazia
- $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a), i = 0, ..., m-1, s_i = at, s_{i+1} = bt : a, b \in \Gamma_\varepsilon$ e $t \in \Gamma^*$, existe uma transição de r_i para r_{i+1} por leitura de w_{i+1} quando a está no topo da pilha, que é substituído por b
- $r_m \in F$, um estado de aceitação é atingido quando o input foi consumido

Isto é, w é aceite se $(q_0, w, \varepsilon) \vdash^* (q_f, \varepsilon, X), q_f \in F$. Diz-se que os autómatos de pilha funcionam pelo critério dos estados de aceitação, o conteúdo da pilha é irrelevante.

Linguagem de um Autómato de Pilha

A linguagem de um AP $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ é:

$$L(M) = \{w \in \Sigma^* : (q_0, w, \varepsilon) \vdash^* (q_f, \varepsilon, X), q_f \in F$$

Uma linguagem é independente do contexto se e só se existe um autómato de pilha que a reconhece:

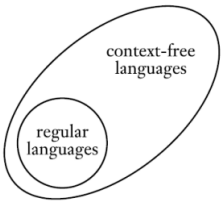

Figura 1.6: Diagrama de Venn das linguagens regulares e livres de contexto

Se uma linguagem é independente do contexto então existe algum autómato de pilha que a reconhece.

1.6.4 Obter AP a Partir de GIC

- Introduzir estados q_{start}, q_{loop} e q_{accept}
- Introduzir transições:
  - q_{start} para q_{loop} com empilhamento de $S\$$
  - q_{loop} para q_{accept} que limpa \$ depois do input estar consumido
  - q_{loop} para q_{loop}:
    - Por cada terminal a consumir a na fita a fazer *pop* a na pilha
    - Por cada variável A empilhar o lado direito das produções de A

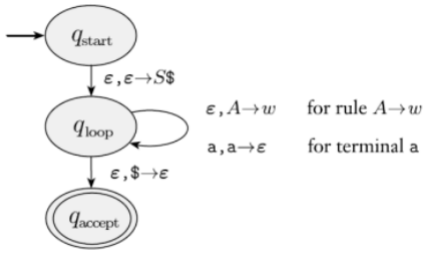

Figura 1.7: Grafo de execução do algoritmo

1.7 Máquinas de Turing

Uma máquina de Turing é um 7-tuplo $MT = (Q, \Sigma, \Gamma, \delta, q_0, q_{aceita}, q_{rejeita})$ onde:

- Q é um conjunto finito de estados
- Σ é o alfabeto de entrada (não contém o símbolo branco)

- Γ é o alfabeto de fita $\Sigma \subset \Gamma$ (inclui branco)
- $\delta : Q \times \Sigma \to Q \times \Gamma \times \{L, R\}$ é a função de transição
- q_0 é o estado inicial
- q_{aceita} é o estado de aceitação
- $q_{rejeita}$ é o estado de rejeição

1.7.1 Computação por uma Máquina de Turing

A computação inicia com a cabeça o mais à esquerda na fita, posicionada sobre o primeiro símbolo do input. As células da fita à direita do input contêm branco.

Se em algum momento a máquina tenta mover a cabeça para a esquerda do início, a cabeça permanece na mesma posição. A computação termina quando a máquina entra num dos estados de aceitação ou rejeição. Se nenhum desses casos ocorre, a computação continua para sempre.

Por contraste aos autómatos finitos, uma MT tanto pode ler a partir da fita como escrever sobre ela, a cabeça pode mover-se tanto para a esquerda quanto para a direita. A máquina para quando entra num estado de aceitação ou rejeição, independentemente do conteúdo da fita. indeterminadamente.

1.7.2 Configuração de uma Máquina de Turing

Uma configuração de uma MT mostra o conteúdo da fita, o estado em que a máquina está e a posição onde está a cabeça de leitura/escrita.

- A configuração inicial é $q_0 w$
- Uma configuração de aceitação tem o estado q_{accept} e de rejeição tem o estado q_{reject}

1.7.3 Linguagem de uma Máquina de Turing

A linguagem de uma máquina de Turing M, $L(M)$ é o conjunto das palavras para as quais M para numa configuração de aceitação.

Linguagem Turing-Reconhecível

Uma linguagem L é Turing-reconhecível se existe uma máquina de Turing M tal que para toda a palavra x pertence a L, M e aceita x.

Linguagem Turing-Decidível

Máquinas que nunca entram em loop são *decisores*.
Uma linguagem L é Turing-decidível (ou decidível), se alguma máquina de Turing M a decide: $\forall x \in L, M$ para e aceita x, $\forall x \in \overline{L}, M$ para e rejeita x.

Uma linguagem Turing-decidível é sempre reconhecível, mas o oposto nem sempre se verifica.

Duas máquinas de Turing dizem-se equivalentes se reconhecem a mesma linguagem.

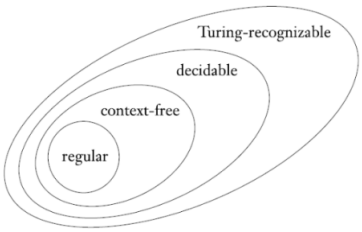

Figura 1.8: Diagrama de Venn das linguagens formais, incluindo as linguagens Turing-reconhecíveis

1.7.4 Variantes da Máquina de Turing

Existem diversas variantes da máquina de Turing, todas com o mesmo poder computacional.

Variante Stay

A função de transição admite "stay".

Variante Multi-fita

A função de transição é do tipo:

$$\delta : Q \times \Gamma^k \to Q \times Q \times \Gamma^k \times \{L, R, S\}^k$$

Onde k é o número de fitas.

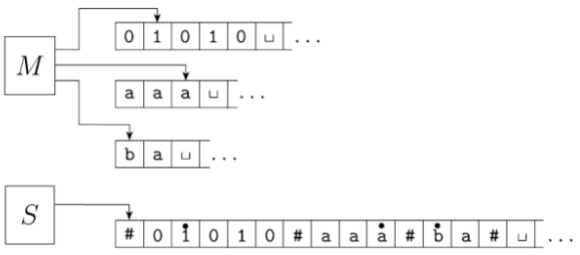

Figura 1.9: Ideia da prova de equivalência de MT multi-fita com MT tradicional

Variante Não Determinista

- A função de transição devolve subconjuntos de ternos:

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

- A computação é uma árvore de configurações onde a raiz é a configuração inicial
- Uma palavra é aceite se alguma folha é uma configuração de aceitação

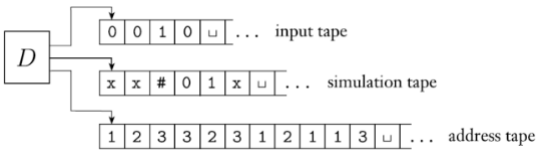

Figura 1.10: Ideia da prova de equivalência de MT não determinista com MT tradicional

Capítulo 2

Decidibilidade

2.1 Correspondência

Considerem-se dois conjuntos A e B e uma função $f : A \to B$:

- f é uma correspondência um-para-um se é injetiva
- f é *onto* (sobrejetiva) se $\forall b \in B, \exists a \in A : f(a) = b$
- A e B são do mesmo tamanho se existe $f : A \to B$ (correspondência) que é um-para-um e *onto*
- Numa correspondência todo o elemento de A mapeia num único elemento de B e cada elemento de B tem um elemento de A que mapeia nele.

2.1.1 Contabilidade

Um conjunto diz-se contável sse é finito ou do tamanho de $\mathbb{N}$.

Teorema

Algumas linguagens não são Turing-reconhecíveis.

- O conjunto de todas as strings é contável
- É possível mapear o conjunto das máquinas de Turing no conjunto das strings binárias, codificando cada máquina M em $< M >$.
- O conjunto de todas as strings binárias infinitas não é contável
- É possível mapear o conjunto de todas as linguagens no conjunto de todas as strings binárias infinitas, logo o conjunto de todas as linguagens não é contável.

2.2 Redução por Mapeamento

Uma linguagem A é reduzível por mapeamento a B, $A \leq_m B$ se existe uma função computável $f : \Sigma^* \to \Sigma^*$ tal que $\forall w \in A \Leftrightarrow f(w) \in B$. A f chama-se redução de A para B.

2.2.1 Resumo

Seja $A \leq_m B$ uma redução por mapeamento computável (para as classes de complexidade assume-se ser polinomial):

A	$\leq_m$	B
Decidível	$\Leftarrow$	Decidível
Indecidível	$\Rightarrow$	Indecidível
Turing-Reconhecível	$\Leftarrow$	Turing-Reconhecível
Não Turing-Reconhecível	$\Rightarrow$	Não Turing-Reconhecível
$\mathbb{P}$	$\Leftarrow$	$\mathbb{P}$
$\mathbb{NP}$-completo	$\Rightarrow$	$\mathbb{NP}$-completo, sse $\in \mathbb{NP}$
$\mathbb{NP}$	$\Leftarrow$	$\mathbb{NP}$

2.3 Máquina de Turing Universal

Existe uma máquina de Turing U que aceita como input uma outra máquina de Turing M codificada como $M \rightarrow < M >$ na fita de U. U pode simular o comportamento de qualquer máquina de Turing $< M >$, pelo que é apelidada de máquina universal.

2.3.1 Problemas Decidíveis

Problemas de aceitação:

- A_{DFA}
- A_{NFA}
- A_{REX}
- A_{CFG}

Problemas de vacuidade:

- E_{DFA}
- E_{CFG}

Problema de equivalência:

- EQ_{DFA}

2.3.2 Problemas Indecidíveis

Problema de aceitação:

- A_{TM}

Problema de vacuidade:

- E_{TM}

Problemas de equivalência:

- EQ_{CFG}
- EQ_{TM}

Também o problema da paragem é indecidível:

$$HALT_{TM} = \{< M, w > | M \text{ é uma } MT \text{ e } M \text{ pára sobre o input } w\}$$

E o problema da linguagem regular:

$$Regular_{TM} = \{< M > | M \text{ é uma } MT \text{ e } L(M) \text{ é uma linguagem regular}\}$$

Resumo

Problema	Decidível	Turing-Reconhecível	Co-Turing-Reconhecível
A_{TM}	Não	Sim	Não
EQ_{TM}	Não	Não	Não
$\overline{A_{TM}}$	Não	Não	Sim
$\overline{EQ_{TM}}$	Não	Não	Não

2.4 Teorema de Rice

O problema de determinar se a linguagem de uma dada máquina de Turing tem uma propriedade P, não trivial, é indecidível. Seja P a linguagem de todas as descrições de máquinas de Turing, onde P:

- É não trivial, i.e., contém algumas mas não todas as descrições de MT.
- P representa uma propriedade da linguagem dessas MTs:
  - Sempre que $L(M1) = L(M2)$, tem-se $\langle M1\rangle \in P$ sse $\langle M2\rangle \in P$, $M1$ e $M2$ quaisquer MTs.
- P é uma linguagem indecidível

Capítulo 3

Complexidade Temporal

Seja M uma máquina de Turing determinística que pára sobre todos os inputs. A complexidade temporal de M é uma função $f : \mathbb{N} \to \mathbb{N}$, onde $f(n)$ é o número máximo de passos que M usa sobre qualquer input de tamanho n.

3.1 Notação

Sejam $f, g : \mathbb{N} \to \mathbb{R}^+$ duas funções.

3.1.1 O-grande

Diz-se que $f(n) = O(g(n))$ se $\exists c, n_0 : \forall n \geq n_0, f(n) \leq cg(n)$. $g(n)$ é um limite superior assintótico para $f(n)$.

3.1.2 o-pequeno

Diz-se que $f(n) = o(g(n))$ se $\exists c, n_0 : \forall n \geq n_0, f(n) < cg(n)$. $f(n)$ é assintoticamente menor que $g(n)$.

3.2 Complexidade e Modelo de Computação

A complexidade temporal de alguns problemas depende do modelo de computação utilizado.

3.2.1 Multi-Fita e Uni-Fita

Seja $t(n)$ uma função, onde $t(n) \geq n$. Toda a MT multi-fita de tempo $t(n)$ tem uma MT uni-fita equivalente que corre em tempo $O(t^2(n))$.

A ideia da prova consiste em simular a MT multi-fita na uni-fita. Como simular cada passo da multi-fita demora, no máximo $O(t(n)$, o tempo total usado é $O(t^2(n))$.

3.2.2 Determinística e Não Determinística

Seja M um decisor não determinístico. O tempo de execução de M é $f : \mathbb{N} \to \mathbb{N}$, onde $f(n)$ é p número máximo de passos que M usa em qualquer ramo da sua computação.

Seja $t(n)$ uma função, onde $t(n) \geq n$. Toda a MT não determinística uni-fita de tempo $t(n)$ tem uma MT determinística uni-fita equivalente de tempo $2^{O(t(n))}$.

A ideia da prova consiste em criar uma MT determinística que simula a árvore de computação da MT não determinística. Num input de tamanho n, cada ramo tem, no máximo, comprimento $t(n)$ e cada nó tem b filhos, onde b é o número máximo de escolhas da função de transição. O número máximo de folhas é $b^{t(n)}$. Como a simulação visita todos os nós de uma dada profundidade antes de partir para a próxima, o tempo para atravessar da raiz até um nó é $O(t(n))$, pelo que o tempo de execução é $O(t(n)b^{t(n)}) = 2^{O(t(n))}$.

3.3 Classes de Complexidade

3.3.1 Classe TIME

Seja $t : \mathbb{N} \to \mathbb{N}$ uma função. A classe de complexidade temporal TIME é:

$TIME(t(n)) = \{L|\ L$é uma linguagem decidida por uma MT de tempo $O(t(n))\}$

3.3.2 Classe $\mathbb{P}$

$\mathbb{P}$ é a classe das linguagens que são decidíveis em tempo polinomial numa MT determinística uni-fita:

$$\mathbb{P} = \bigcup_k TIME(n^k)$$

$\mathbb{P}$ corresponde à classe dos problemas que podem ser realisticamente resolúveis num computador.

PATH

O problema PATH define-se da seguinte forma:

$PATH = \{< G, s, t > |\ G$ é um grafo direcionado que tem um caminho de s para $t\}$

Tem-se que PATH $\in \mathbb{P}$. A seguinte MT é um decisor para PATH que corre em tempo polinomial:

- Marcar nó s
- Repetir até que mais nenhum nó seja marcado:
  - Verificar todos os arcos de G. Se (a, b) é um arco onde a está marcado e b não está marcado, marcar b.
  - Se t foi marcado, aceitar. Senão, rejeitar.

Se G tem n nós existem n^n caminhos possíveis de s para t. No terceiro passo marca-se, no mínimo, um nó a cada iteração e o segundo passo repete-se, no máximo, $n-1$ vezes. A MT corre em $O(n^2)$.

3.3.3 Classe NTIME

NTIME(t(n)) é a classe de complexidade temporal não determinística:

$NTIME(t(n)) = \{L|\ L$ é uma linguagem decidível por uma MT não determinística de tempo $O(t(n))\}$

3.3.4 Classe $\mathbb{NP}$

Uma linguagem está em $\mathbb{NP}$ sse é decidível por alguma MT não determinística de tempo polinomial:

$$NP = \bigcup_k NTIME(n^k)$$

É fácil verificar que P $\subseteq \mathbb{NP}$. Esta classe pode ainda ser definida como a classe das linguagens que possuem verificadores em tempo polinomial.

Verificador

Um verificador para uma linguagem A é um algoritmo V, onde:

$$A = \{w|\ V \text{ aceita } < w, c > \text{ para alguma } string\ c\}$$

O tempo de execução do verificador é medido exclusivamente em termos do comprimento de w. Um verificador usa ainda um certificado, c, para verificar que $w \in A$. Para verificadores polinomiais, c tem comprimento polinomial no comprimento de w.

CLIQUE

O problema CLIQUE define-se da seguinte forma:

$$CLIQUE = \{< G, k > |\ G \text{ é um grafo não direcionado com uma } k - clique\}$$

Em que uma $k-clique$ é um sub-grafo onde todos os pares de nós estão ligados por uma aresta (sub-grafo completo). Tem-se que CLIQUE $\in \mathbb{NP}$. O seguinte verificador para CLIQUE corre em tempo polinomial, sobre o input $<< G, k >, c >$:

- Testar se c é um sub-grafo com k nós em G
- Testar se G contém todas as arestas que ligam os nós em c
- Se ambos os testes passam, aceitar. Senão, rejeitar.

$\mathbb{NP}$-completude

Uma linguagem B diz-se $\mathbb{NP}$-completa se:

- $B \in \mathbb{NP}$
- $\forall A \in \mathbb{NP}$, A é redutível em tempo polinomial a B

Se $B \in$ P e B é $\mathbb{NP}$-completa, então $\mathbb{P}$=$\mathbb{NP}$.

A linguagem:

$$SAT = \{\varphi |\ \varphi \text{ é uma expressão booleana satisfazível}\}$$

É $\mathbb{NP}$-completa. Algumas outras linguagens $\mathbb{NP}$-completas são:

- $3SAT$
- $CLIQUE$
- $HAMPATH$
- $UHAMPATH$
- $VERTEX - COVER$