Teoria da Computação

Tese de Church-Turing e Máquina de Turing Universal

Thiago Alves

Algoritmos

- Um algoritmo é uma sequência de instruções simples para realizar uma tarefa
- Apesar de sempre terem sido bastante utilizados na matemática, a noção de algoritmo não foi definida formalmente até o século XX

Algoritmos

- Turing criou as suas máquinas para definir formalmente a noção de algoritmo
- Máquinas de Turing computam em strings
- E se quisermos outro tipo de entrada?
  - Por exemplo, verificar uma propriedade de um grafo?

Algoritmos

- E se quisermos outro tipo de entrada?
  - Por exemplo, verificar uma propriedade de um grafo?
  - Temos que representar o grafo como string
  - Vamos usar a notação “O” como a representação de O em string

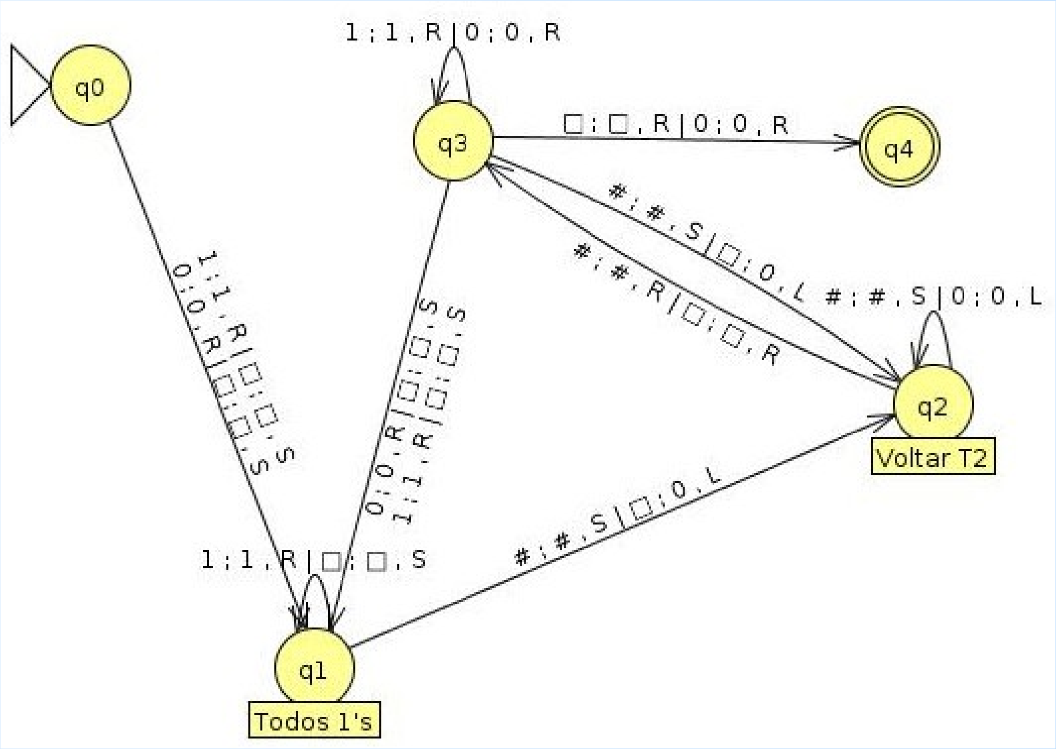
Algoritmos

- A = {“G” | G é um grafo completo}
- Como representar um grafo G como uma string?

Algoritmos

- A = {“G” | G é um grafo completo}
- Como representar um grafo G como uma string?
  - “G” = 0111#1011#0101#1110#
  - Grafo com 4 vértices
  - Cada bloco representa as arestas de um vértice
  - Matriz de adjacências

Algoritmos



- M decide A, logo A é decidível.

Algoritmos

- Programas também podem ser entradas
- Para um interpretador ou compilador
- Como representar uma máquina de Turing em string binária?

Máquinas de Turing em Strings

- Vamos focar em máquinas de Turing com alfabeto de entrada {0, 1}
- Vamos associar com cada elemento importante nas transições, uma quantidade positiva de 1's
  - Estados
  - Símbolos da fita
  - Direção

Máquinas de Turing em Strings

- Estados:
  - q_1 (inicial) $\rightarrow 1^1$
  - q_2 (aceitação) $\rightarrow 1^2$
  - $q_3 \rightarrow 1^3$
  - ...

Máquinas de Turing em Strings

- Símbolo
  - $B \rightarrow 1^1$
  - X_1 (0) $\rightarrow 1^2$
  - X_2 (1) $\rightarrow 1^3$
  - $X_4 \rightarrow 1^4$
  - ...

Máquinas de Turing em String

- Direções
  - D_1 (L) → 1^1
  - D_2 (S) → 1^2
  - D_3 (R) → 1^3

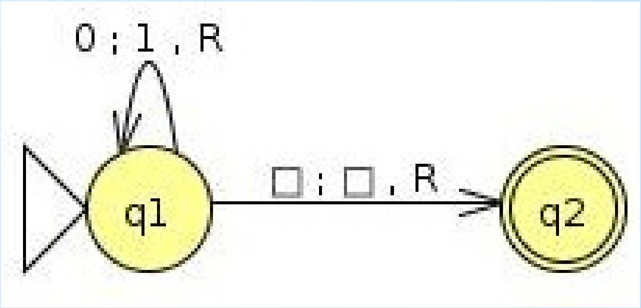
Máquinas de Turing em Binário

- Seja $\delta(q_i, X_j) = (q_k, X_l, D_m)$
- Podemos representar essa regra pela string $1^i01^j01^k01^l01^m$
- O símbolo 0 serve como separador
- Não existem 0's consecutivos nas transições

Máquinas de Turing em Binário

- Representamos uma máquina de Turing concatenando os códigos de cada transição separados por 0 para separação
  - $Código_1 0 Código_2 0 \ldots 11 Código_m 0$
  - Em que $Código_i$ é a representação binária da transição i

Máquinas de Turing em Binário



- 101101011101110101010110101110

Máquina de Turing Universal

- Podemos criar uma máquina de Turing U que recebe o código de uma máquina de Turing M e o código da entrada w
- U é chamada de máquina de Turing Universal
- U recebe uma máquina M e entrada w
- U simula a execução de M com w

Máquina de Turing Universal

- U precisa simular a fita de M
- Representamos a fita de M com a mesma codificação
- Usando 0's como separadores
- Fita: 101B
- 1110110111010

Máquina de Turing Universal

- Seja “M” a codicação de M
- Seja “w” a codificação de w
- A máquina U executa a máquina M com entrada w através de “M”#“w”

Execução da Máquina Universal

- U guarda “M” na primeira fita
- U guarda “w” na segunda fita
- U guarda o estado inicial 1 na terceira fita

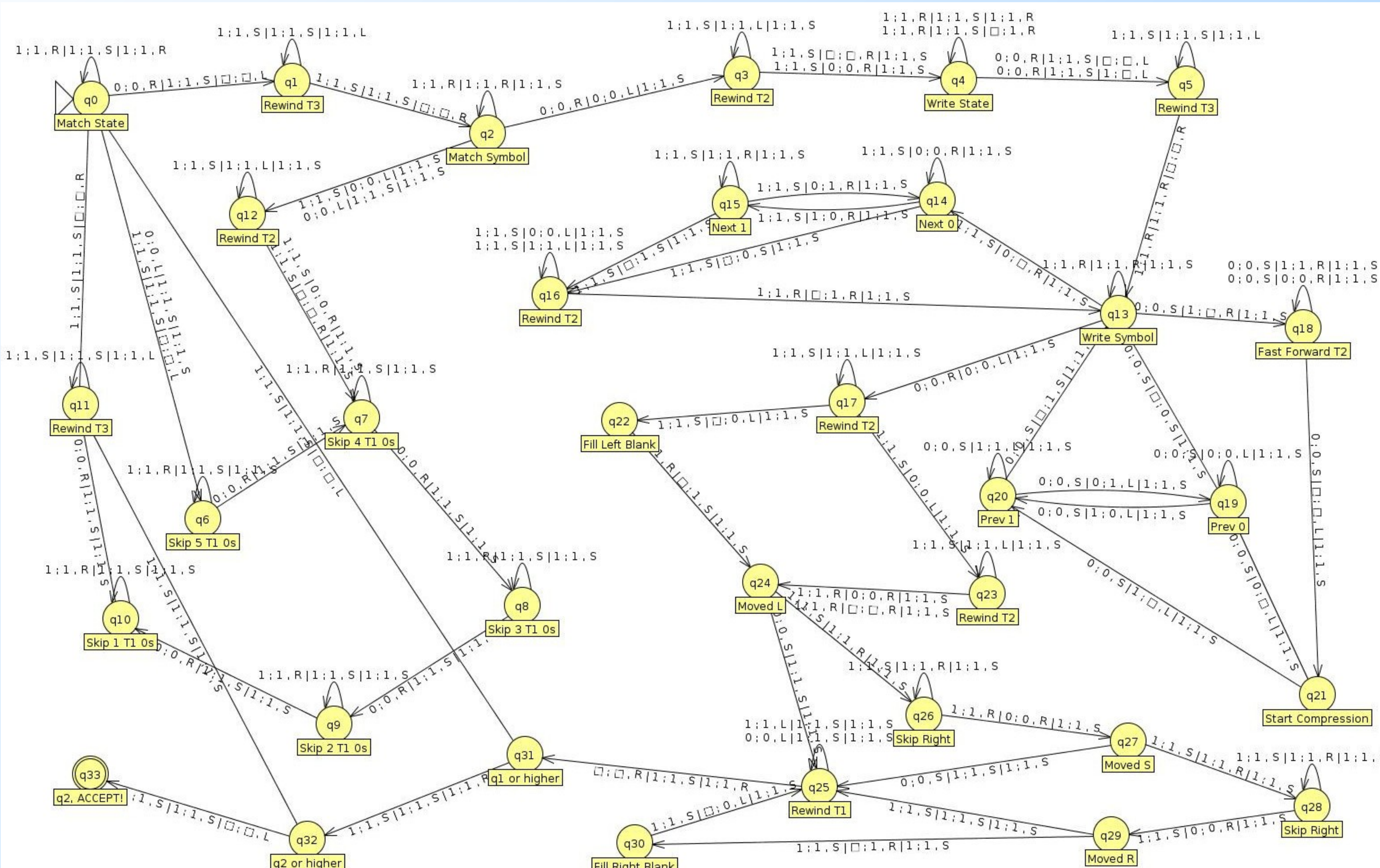
Máquina de Turing Universal

- No estado atual, na terceira fita procura a transição correspondente na primeira fita
- Verifica o símbolo lido na segunda fita
- Modifica o estado na terceira fita

Máquina de Turing Universal

- Modifica a segunda fita de acordo com o novo símbolo da transição
- Muda a posição da segunda fita de acordo com a direção da transição

Máquina de Turing Universal



Máquina de Turing Universal

- Vamos executar U com:
- “M”: 10110101110111010101101011110
- “w”: 110110

Algoritmos

- Turing criou as suas máquinas para definir formalmente a noção de algoritmo
- Outros formalismos foram criados com o mesmo objetivo
  - Cálculo-λ
  - Gramáticas irrestritas
  - Linguagem L
  - Funções μ-recursivas

Cálculo-λ

$$\lambda f.\lambda x.f\ (f\ x)$$

Gramáticas Irrestritas

$$S \to ABCS,$$
$$S \to T_c,$$
$$CA \to AC,$$
$$BA \to AB,$$
$$CB \to BC,$$
$$CT_c \to T_c c,$$
$$CT_c \to T_b c,$$
$$BT_b \to T_b b,$$
$$BT_b \to T_a b,$$
$$AT_a \to T_a a,$$
$$T_a \to e$$

Linguagem L

$$
\begin{array}{ll}
[A] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & \text{IF } X \neq 0 \text{ GOTO } A
\end{array}
$$

Funções μ-recursivas

$$f(x, 0) = u^1{}_1(x)$$

$$f(x,y+1) = s(u^3{}_2(y,f(x,y),x))$$

Algoritmos

- Todos os formalismos para a noção de algoritmo foram mostrados equivalentes
  - Cálculo-λ
  - Máquinas de Turing e extensões
  - Gramáticas irrestritas
  - Linguagem L
  - Funções μ-recursivas
  - E outros

Tese de Church-Turing

- É a conexão entre a definição informal de algoritmo e a definição formal
- Tudo que pode ser feito com um algoritmo pode ser feito com uma máquina de Turing e vice-versa

Tese de Church-Turing

- É uma tese pois não pode ser provada por conta da definição informal de algoritmo
- As equivalências entre as variações de máquinas de Turing e os outros formalismos para algoritmos podem ser vistas como evidências da tese

Algoritmos

- Agora vamos usar algoritmos em alto nível para representar máquinas de Turing

Algoritmos

- A = {“G” | G é um grafo conexo}
- Máquina de Turing M que decide A:

M = “On input $\langle G \rangle$, the encoding of a graph G:

1. Select the first node of G and mark it.

2. Repeat the following stage until no new nodes are marked:

3. For each node in G, mark it if it is attached by an edge to a node that is already marked.

4. Scan all the nodes of G to determine whether they all are marked. If they are, *accept*; otherwise, *reject*.”

Teorema

- Se A é uma linguagem livre de contexto então A é recursiva.

Prova

- Suponha A livre de contexto
- Logo, existe G tal que L(G) = A
- Podemos construir uma máquina de Turing D que sempre pára e que L(D) = A
- D representa o algoritmo CYK para saber se G gera uma string w

Prova

$D =$ "On input $w = w_1 \cdots w_n$:

1. If $w = \varepsilon$ and $S \rightarrow \varepsilon$ is a rule, *accept*. [[handle $w = \varepsilon$ case]]
2. For $i = 1$ to n: [[examine each substring of length 1]]
3. For each variable A:
4. Test whether $A \rightarrow \mathtt{b}$ is a rule, where $\mathtt{b} = w_i$.
5. If so, place A in $table(i, i)$.
6. For $l = 2$ to n: [[l is the length of the substring]]
7. For $i = 1$ to $n - l + 1$: [[i is the start position of the substring]]
8. Let $j = i + l - 1$, [[j is the end position of the substring]]
9. For $k = i$ to $j - 1$: [[k is the split position]]
10. For each rule $A \rightarrow BC$:
11. If $table(i, k)$ contains B and $table(k + 1, j)$ contains C, put A in $table(i, j)$.
12. If S is in $table(1, n)$, *accept*. Otherwise, *reject*."

- Logo, A é recursiva.

Hierarquia de Chomsky