João Guilherme Fritsche Colombo

# MÁQUINAS DE TURING NÃO DETERMINÍSTICAS COM COMBINADORA PARA A COMPUTAÇÃO DE FUNÇÕES

Florianópolis
2019

João Guilherme Fritsche Colombo

# MÁQUINAS DE TURING NÃO DETERMINÍSTICAS COM COMBINADORA PARA A COMPUTAÇÃO DE FUNÇÕES

Trabalho de Conclusão de Curso submetido como requisito parcial para a obtenção do grau de Bacharel em Ciências da Computação.

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Curso de Graduação em Ciências da Computação

Orientadora: Profa. Dra. Jerusa Marchi

Florianópolis
2019

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Colombo, João Guilherme Fritsche
Máquinas de Turing não determinísticas com combinadora para a computação de funções / João Guilherme Fritsche Colombo ; orientadora, Jerusa Marchi, 2019.
72 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2019.

Inclui referências.

1. Ciências da Computação. 2. Teoria da Computação. 3. máquinas de Turing não determinísticas. 4. funções computáveis. I. Marchi, Jerusa. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

João Guilherme Fritsche Colombo

**MÁQUINAS DE TURING NÃO DETERMINÍSTICAS COM COMBINADORA PARA A COMPUTAÇÃO DE FUNÇÕES**

"Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do grau de Bacharel em Ciências da Computação e aprovado em sua forma final pelo Curso de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina."

Florianópolis, 22 de março de 2019.

---

Prof. Dr. Rafael Luiz Cancian
Coordenador do Curso

---

Profa. Dra. Jerusa Marchi
Orientadora

**Banca Examinadora:**

---

Prof. Dr. Cezar Augusto Mortari
Universidade Federal de Santa Catarina

---

Prof. Dr. Elder Rizzon Santos
Universidade Federal de Santa Catarina

*para*
*Alan M. Turing*
*1912-1954*

## AGRADECIMENTOS

À minha mãe, Rosemary, e ao meu pai, Sandro. À minha orientadora, Jerusa Marchi. À minha namorada, Bruna. Às minhas irmãs, Maria Alice e Patricia. Ao meu irmão, Fernando, e à sua esposa, Fernanda. Ao meu meu avô, Edegar, e às minhas avós, Iolanda e Tereza. Ao meu tio, Valmor. Ao meu primo, Ricardo. A toda minha família. Aos meus amigos, Eduardo Dudel, Felipe Goedert, Guilherme Reinaldo, Marcos Dias, Marcos Schead, Matheus Alves, Matheus Picolli e Nathan Molinari. Aos membros da banca, Cezar Mortari, Elder Santos e Olinto Furtado. Aos meus professores, Luiz Santos, José Güntzel, Marcelo Carvalho, Rafael Cancian, Rosvelter da Costa e Sérgio Peters e à minha professora, Melissa Weber. À Universidade Federal de Santa Catarina (UFSC). À Secretaria de Relações Internacionais da UFSC e à Universidade Técnica de Munique.

Também aos autores citados. Aos responsáveis e doadores do projeto Sci-Hub, que lutam pelo conhecimento. Aos usuários dos sites de perguntas e respostas Mathematics Stack Exchange e TEX Stack Exchange, que fomentam questões, respectivamente, a respeito da Matemática e do TEX, software utilizado para formatação. Aos desenvolvedores e contribuidores do Graphviz, software de visualização de grafos utilizado na elaboração das ilustrações. Aos criadores e contribuidores do abnTEX2, pacote para o TEX que facilitou a adequação para as normas da ABNT.

*Mas pra quem tem pensamento forte,*
*o impossível é só questão de opinião.*
*(Charlie Brown Jr.)*

## RESUMO

Este trabalho aponta que as definições existentes de funções computadas por máquinas de Turing não determinísticas não permitem adaptar diretamente todos os problemas de decisão para funções e não justificam a obtenção, quando é, em tempo polinomial de suas saídas. Por esse motivo, propõe uma variação, baseada na associação de uma árvore de computação à estratégia de *divisão e conquista*, denominada máquina de Turing não determinística com combinadora, capaz dessa adaptação e com uma definição de tempo bem justificada. Além disso, exemplifica a variação proposta com máquinas que computam as funções das classes $\#P$ e $OptP$, as funções características dos complementos das linguagens decididas por máquinas de Turing não determinísticas e uma função que resolve o Problema de Deutsch-Jozsa.



**Palavras-chaves:** Teoria da Computação, máquinas de Turing não determinísticas, funções computáveis.

**ABSTRACT**

We bring attention to the fact that existing definitions of functions computable by nondeterministic Turing machines are not able to directly adapt all decision problems to functions and that none of them give a justification for its outputs' obtainment, when it is, in polynomial time. Motivated by that, we propose a new nondeterministic Turing machine variation, based on the association of a computation tree and the *divide and conquer* strategy, capable of this adaptation and with a well-justified time definition. Besides that, we exemplify the proposed variation with machines that compute the functions of the $\#P$ and $OptP$ classes, the characteristic functions of the complements of the languages decided by non-deterministic Turing machines and a function that solves the Deutsch-Jozsa Problem.

**Keywords:** Theory of Computation, Nondeterministic Turing Machines, Computable Functions.

## SUMÁRIO

# 1 INTRODUÇÃO

A Teoria da Computação é a área que estuda e classifica o que pode e o que não pode ser feito mecanicamente. Vinda da Matemática, ela nasceu em 1936, após Alonzo Church e Alan Turing, independentemente, resolverem de formas diferentes o famoso Problema de Decisão[1], conhecido como *Entscheidungsproblem*[2]. Este é um dos problemas propostos poucos anos antes por David Hilbert em sua busca pela fundamentação da Matemática[3]. Ele questionava a existência de algum método geral de decidir se um enunciado matemático é verdadeiro, falso ou, após os Teoremas da Incompletude de Kurt Gödel[4], impossível de ser provado. (HOPCROFT; ULLMAN, 1979; SIPSER, 2007; COPELAND, 2017; TURING, 1936)

Para a surpresa dos(as) matemáticos(as) da época, a resposta foi negativa. Church e Turing provaram isto formalizando rigorosamente a noção, muito usada mas até o momento apenas intuitiva (SIPSER, 2007), de algoritmo[5], definindo e delimitando o que é e o que não é computável e mostrando que existem problemas indecidíveis. (SIPSER, 2007; COPELAND, 2017)

Church, poucos meses antes de Turing, usando o seu $\lambda$-cálculo definiu o *efetivamente calculável*, já Turing formalizou uma máquina abstrata que captura todas as capacidades *puramente mecânicas* de um computador humano. Embora tenham usado técnicas e expressões diferentes, o *efetivamente calculável* de Church e o *puramente mecânico* de Turing, são, e assim provado por Turing[6], equivalentes. (TURING, 1936; COPELAND, 2017)

---

[1] Emil Post também resolveu o problema de maneira similar a de Turing, formalizando uma máquina, porém alguns meses depois e ciente do trabalho de Church (DAVIS, 2004; POST, 1936).

[2] "*Entscheidung*" e "*Problem*" são, respectivamente, "decisão" e "problema" em alemão, "*Entscheidungsproblem*" é a composição das duas palavras (o "s" serve apenas para compor e facilitar a pronúncia).

[3] O problema já havia sido formulado por outros matemáticos; porém ficou famoso pelo destaque que Hilbert deu a ele (PETZOLD, 2008; KLEENE, 1952).

[4] O Primeiro Teorema da Incompletude mostra que, em qualquer sistema formal que contém a aritmética, há teoremas que não são alcançáveis, ou seja, que podem ser verdadeiros mas que não podem ser provados (NAGEL; NEWMAN, 2001). Mas isso não excluía a decidibilidade, que, em vez de decidir sobre a verdade, decidiria sobre a provabilidade de um enunciado matemático (PETZOLD, 2008).

[5] Podemos descrever um algoritmo informalmente como uma receita de bolo: uma sequência de instruções para obter um resultado. Embora Turing e Church tenham definido o que hoje conhecemos como algoritmo, o termo, que é uma corrupção do nome do matemático persa Muḥammad ibn Mūsā al-Khwārizmī (ERICKSON, 2018), só começou a ser utilizado nesse sentido em 1960 (PETZOLD, 2008).

[6] Church publicou seu trabalho depois do início e durante o período de finalização do de Turing. Mas Turing, por ter usado uma técnica diferente, decidiu enviar seu trabalho para ser publicado mesmo assim, para então, depois, enviar um apêndice em que mostra esta equivalência (PETZOLD, 2008).

A solução de Turing é considerada, até mesmo por Church[7], superior e mais intuitiva (COPELAND, 2017; CHURCH, 1937), e, para muitos, Alan Turing é considerado o pai da Computação[8]. Mas mesmo que, com suas máquinas, Turing tenha resolvido o problema de decisão, elas em si não decidiam um determinado problema, mas computavam números[9]. Porém, como notado por Turing, elas podiam ser, e foram, facilmente adaptadas para outros propósitos, como o de decidir e o de computar funções (HOPCROFT; ULLMAN, 1979).

Turing fez a primeira prova de equivalência entre modelos da Teoria da Computação (TURING, 1936). Um modelo ser equivalente a outro significa que todo problema que pode ser resolvido nele pode ser resolvido no outro e vice-versa. Depois do $\lambda$-cálculo e da máquina original de Turing, novos modelos surgiram, porém nenhum deles resolvia um número maior de problemas, fato que evidencia e faz ser universalmente aceita a Tese de Church-Turing: *tudo que é computável é computável por uma máquina de Turing.* (COPELAND, 2017; HOPCROFT; ULLMAN, 1979; SIPSER, 2007)

A partir desses modelos, iniciou-se o estudo da complexidade computacional, que agrupa, em diversas classes, problemas computáveis de acordo com a sua dificuldade de computação — não o quão difícil é achar uma forma de o computar, mas o quão difícil é computá-lo. Dificuldade esta que pode estar relacionada a qualquer recurso computacional, por exemplo tempo[10] ou espaço. (SIPSER, 2007; HOPCROFT; ULLMAN, 1979; DEAN, 2016)

Duas dessas classes protagonizam o principal problema em aberto da Computação[11]: o $\mathcal{P}$ *versus* $\mathcal{NP}$[12]. Ambas dizem respeito ao tempo polinomial[13], porém sobre modelos diferentes: a primeira sobre a máquina de Turing original e a segunda sobre a variação que acrescenta o não determinismo, chamada de máquina de Turing não determinística[14].

---

[7] Que depois orientou Turing em seu doutorado (PETZOLD, 2008).

[8] É um exagero dizer que uma pessoa é o pai (ou a mãe) de uma área tão diversificada como a Computação, mas se quisermos afirmar isto, Turing com certeza é uma das pessoas a se considerar, tanto que o equivalente ao prêmio Nobel da área é chamado de prêmio Turing: <https://amturing.acm.org>.

[9] Computavam a parte fracionária deles, um número era computável se e somente se sua parte fracionária podia ser computada por uma máquina de Turing.

[10] Não estamos falando sobre a medição do tempo em si, mas do número de passos de computação necessários.

[11] Que é um dos sete problemas do milênio do Instituto Clay de Matemática, dos quais apenas um foi resolvido até hoje e possuem, cada um, uma recompensa de US$ 1.000.000 (COOK, 2000).

[12] A primeira descrição informal do problema que se tem registro é em uma carta do matemático John Nash à Agência Nacional de Segurança dos Estados Unidos de 1955 (NASH, 1955) e também em uma carta de Gödel a John V. Newman de 1956 (AARONSON, 2017).

[13] Aquele que pode ser descrito por um polinômio em relação ao tamanho da entrada.

[14] Muito conhecida, outra formulação do problema é que na classe $\mathcal{P}$ estão os problemas com respostas computáveis em tempo polinomial e na classe $\mathcal{NP}$ estão os problemas com respostas verificáveis em tempo polinomial.

O que está em aberto é se há problemas que estão em $\mathcal{NP}$ mas não em $\mathcal{P}$. (SIPSER, 2007; COOK, 2000)

Se esse for o caso, que é o mais acreditado pela comunidade (GASARCH, 2019), isso quer dizer que a máquina de Turing não determinística é mais rápida que a original. Porém, diferentemente da original, esta ainda não foi bem-adaptada para a computação de funções e só é bem definida para a decisão de problemas.

## 1.1 OBJETIVO

O objetivo geral deste trabalho é propor uma variação da máquina de Turing não determinística para a computação de funções.

### 1.1.1 OBJETIVOS ESPECÍFICOS

Para realizar de maneira satisfatória o objetivo geral, iremos, em ordem:

1. Revisar a computação de funções por máquinas de Turing determinísticas.
2. Apontar o que falta nas definições existentes de funções computadas por máquinas de Turing não determinísticas.
3. Descrever informalmente a variação proposta e a sua inspiração, a estratégia de *divisão e conquista*.
4. Definir formalmente a variação proposta, provando a sua equivalência com a máquina de Turing determinística.
5. Exemplificar a variação proposta e a sua versatilidade.
6. Comparar a variação proposta com as definições existentes de funções computadas por máquinas de Turing não determinísticas.

## 1.2 ESTRUTURAÇÃO

Além desta, este trabalho ainda possui quatro seções. Na Seção 2, definiremos o modelo de computação da máquina de Turing para a decisão de problemas, em suas variações determinísticas e em sua variação não determinística. Na Seção 3, definiremos como computar uma função, de forma determinística, com uma máquina de Turing determinística e apresentaremos definições existentes, porém insatisfatórias, de como computar uma função, de forma não determinística, com uma máquina de Turing não determinística. Na Seção 4, proporemos e definiremos nossa variação. E, na Seção 5, concluiremos o trabalho com considerações sobre o que foi feito e trabalhos futuros. Para a elaboração dos exemplos foi desenvolvido um simulador, cujo código-fonte está disponível e documentado no Apêndice A.

# 2 MÁQUINAS DE TURING

Nesta seção, definiremos o modelo de computação da máquina de Turing para a decisão de problemas. Na primeira subseção, definiremos a máquina padrão, determinística e de uma única fita, que é a mais parecida com a original proposta por Turing[1]. Na segunda, definiremos a variação que possui um número arbitrário de fitas. Esta é útil para provar a equivalência entre a máquina de Turing determinística de uma única fita e a variação não determinística, que iremos, finalmente, definir na última subseção. Nossas definições são inspiradas naquelas apresentadas por Hopcroft e Ullman (1979) e Sipser (2007), porém adaptadas para estabelecer de forma mais clara suas relações com a proposta deste trabalho.

Esta seção forma a base para as definições das demais. Ela é composta por definições e teoremas e serve apenas como referência; por isso, não possui justificativas nem discussões a respeito das diferenças entre as definições feitas e as definições dos livros utilizados como inspiração. Os únicos pré-requisitos para o seu entendimento são a Teoria dos Conjuntos, a Teoria das Linguagens Formais e a Teoria dos Grafos, diferentemente dos livros utilizados como inspiração que, além destas, requerem também a Teoria dos Autômatos.

## 2.1 MÁQUINAS DE TURING DETERMINÍSTICAS DE UMA ÚNICA FITA

Para entender as definições dadas nesta seção, precisamos deixar claro duas analogias usadas por Turing — a de uma fita e a de um estado. A fita substitui e representa o papel que era utilizado nas computações feitas por pessoas, de sua época, início do século XX. Ela é uma fita unidimensional, dividida em símbolos, infinita, portanto teórica, e que nos dias atuais é mais comparada a uma memória de um computador. O estado é o "estado da mente" de quem realizava as computações. Com estas analogias, Turing abstrai uma computação, tratando-a como uma sequência de manipulações, de símbolos, em uma fita, baseadas em um estado e em um símbolo observado da fita, o qual vamos chamar de símbolo atual.

**Definição 2.1.1.** Uma **máquina de Turing determinística de uma única fita** $M$ é uma sétupla:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$$

tal que:

- $Q$ é o conjunto, finito e não vazio, de estados,

---

[1] Como mencionado na Introdução (página 18, nota de rodapé 9), a máquina proposta por Turing não tem o propósito de decidir problemas, mas o de computar (a parte fracionária de) números. A semelhança entre a máquina determinística que vamos definir e a original está em suas formas de computar, mais especificamente em suas definições de movimento (Definição 2.1.3).

- $\Sigma \subset \Gamma$ é o conjunto, finito e não vazio, de símbolos do alfabeto de entrada,
- $\Gamma$ é o conjunto de símbolos do alfabeto da fita,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\triangleleft, \circ, \triangleright\}$ é a função de transição, onde $\triangleleft$, $\circ$ e $\triangleright$ representam, respectivamente, "esquerda", "parado" e "direita",
- $q_0 \in Q$ é o estado inicial,
- $B \in (\Gamma - \Sigma)$ é o símbolo para "em branco",
- $A \subseteq Q$ é o conjunto de estados de aceitação.

**Definição 2.1.2.** Uma **configuração** $c$ de uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ é uma tripla $c = (\alpha, q, s\beta)$ tal que $q \in Q$, $s \in \Gamma$ e $\alpha$ e $\beta \in \Gamma^*$. Dizemos que $M$ está em $c$ quando para $M$:

- $q$ é o estado atual,
- $s$ é o símbolo atual,
- $\alpha$ são os símbolos da fita entre o primeiro símbolo diferente de $B$ (incluindo) e $s$ (não incluindo),
- $\beta$ são os símbolos da fita entre $s$ (não incluindo) e último símbolo diferente de $B$ (incluindo).

**Definição 2.1.3.** Um **movimento** $c_1 \vdash_M c_2$ de uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ é a aplicação da função de transição $\delta$ a alguma configuração $c_1 = (\alpha e, q, sd\beta)$ de $M$ tal que $s \in \Gamma$, $e$ e $d \in (\Gamma \cup \{\varepsilon\})$, $\alpha$ e $\beta \in \Gamma^*$ e:

- $c_2 = (\alpha e, p, s'd\beta)$ se e somente se $\delta(q, s) = (p, s', \circ)$,
- $c_2 = (\alpha, p, es''d\beta)$ se e somente se $\delta(q, s) = (p, s', \triangleleft)$ e $e \neq \varepsilon$,
- $c_2 = (\alpha es'', p, d\beta)$ se e somente se $\delta(q, s) = (p, s', \triangleright)$ e $d \neq \varepsilon$,
- $c_2 = (\varepsilon, p, Bs''d\beta)$ e $\alpha = \varepsilon$ se e somente se $\delta(q, s) = (p, s', \triangleleft)$ e $e = \varepsilon$,
- $c_2 = (\alpha es'', p, B)$ e $\beta = \varepsilon$ se e somente se $\delta(q, s) = (p, s', \triangleright)$ e $d = \varepsilon$

tal que $s'' \in (\Gamma \cup \{\varepsilon\})$ é igual a $\varepsilon$ se e somente se $s' = B$, $\delta(q, s) = (p, s', \triangleleft)$ e $d\beta = \varepsilon$ ou $s' = B$, $\delta(q, s) = (p, s', \triangleright)$ e $\alpha e = \varepsilon$, caso contrário, $s''$ é igual a $s'$.

**Definição 2.1.4.** Uma configuração $c_1$ de uma máquina de Turing determinística de uma única fita $M$ **origina** outra configuração $c_n$ de $M$ se e somente se existe uma sequência de configurações $c_1, c_2, \ldots, c_n$, $n \in [2, \infty) \cap \mathbb{N}$, tal que $c_i \vdash_M c_{i+1}$ para todo $i \in [1, n-1] \cap \mathbb{N}$.

**Definição 2.1.5.** A **configuração inicial** de uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ para uma entrada $w \in \Sigma^*$ é:

- $(\varepsilon, q_0, w)$ se e somente se $w \neq \varepsilon$,
- $(\varepsilon, q_0, B)$ se e somente se $w = \varepsilon$.

**Definição 2.1.6.** Uma **configuração final** de uma máquina de Turing $M$ é uma configuração $c_f$ de $M$ tal que $c_f$ não origina nenhuma outra configuração de $M$.

**Definição 2.1.7.** Uma **configuração de aceitação** de uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ é uma configuração $c_a = (\alpha, q, s\beta)$ de $M$ tal que $c_a$ é uma configuração final de $M$ e $q \in A$.

**Definição 2.1.8.** Uma **configuração de rejeição** de uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ é uma configuração $c_r = (\alpha, q, s\beta)$ de $M$ tal que $c_r$ é uma configuração final de $M$ e $q \notin A$.

**Definição 2.1.9.** Uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ **aceita** uma entrada $w \in \Sigma^*$ se e somente se a configuração inicial de $M$ para $w$ é ou origina uma configuração de aceitação de $M$.

**Definição 2.1.10.** Uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ **rejeita** uma entrada $w \in \Sigma^*$ se e somente se a configuração inicial de $M$ para $w$ é ou origina uma configuração de rejeição de $M$.

**Definição 2.1.11.** Uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ **semidecide** uma linguagem $L \subseteq \Sigma^*$ se e somente se para todo $w \in L$, $M$ aceita $w$ e para todo $w \in (\Sigma^* - L)$, complemento de $L$, $M$ não aceita $w$.

**Definição 2.1.12.** Uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ **decide** uma linguagem $L \subseteq \Sigma^*$ se e somente se $M$ semidecide $L$ e para todo $w \in (\Sigma^* - L)$, complemento de $L$, $M$ rejeita $w$.

**Exemplo 2.1.1.** Uma máquina de Turing determinística de uma única fita $M$ que decide a linguagem $\{\ (01)^n$ tal que $n \in \mathbb{N}\ \}$:

$$M = (\{q_0, q_1, q_a, q_r\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_a\})$$

tal que $q_a$ representa "aceita", $q_r$ representa "rejeita" e $\delta(q, s)$ é definida pelo Quadro 1.

| $q$ \ $s$ | 0 | 1 | $B$ |
|---|---|---|---|
| $q_0$ | $(q_1, 0, \triangleright)$ | $(q_r, 1, \circ)$ | $(q_a, B, \circ)$ |
| $q_1$ | $(q_r, 0, \circ)$ | $(q_0, 1, \triangleright)$ | $(q_r, B, \circ)$ |

Quadro 1 – Imagens da função de transição da máquina do Exemplo 2.1.1. Fonte: adaptado do primeiro exemplo de Turing (1936).

**Teorema 2.1.** *Se uma máquina de Turing determinística de uma única fita $M$ = ($Q$, $\Sigma$, $\Gamma$, $\delta$, $q_0$, $B$, $A$) decide uma linguagem $L \subseteq \Sigma^*$, então podemos construir uma máquina de Turing de uma única fita $M'$ = ($Q$, $\Sigma$, $\Gamma$, $\delta$, $q_0$, $B$, $A'$) que decide o complemento de $L$.*

*Prova.* Basta fazer com que $A'$ seja igual a ($Q$ - $A$). Assim, para todo $w \in L$, a mesma configuração inicial de $M$ e $M'$ para $w$, já que ambas possuem o mesmo estado inicial $q_0$, são ou originam, já que ambas possuem a mesma função de transição $\delta$, a mesma configuração final $(\alpha, q, \beta)$ tal que se $q \in A$, então $M$ aceita $w$, $q \notin A'$ e $M'$ rejeita $w$ e se $q \notin A$, então $M$ rejeita $w$, $q \in A'$ e $M'$ aceita $w$. □

**Teorema 2.2** (Máquina de Turing Universal)**.** *Existe uma máquina de Turing determinística de uma única fita $U$, denominada máquina de Turing Universal, tal que para uma entrada $\langle M, w\rangle$ que simboliza uma máquina de Turing determinística de um única fita $M$ = ($Q$, $\Sigma$, $\Gamma$, $\delta$, $q_0$, $B$, $A$) e uma entrada de $M$, $w \in \Sigma^*$, $U$ aceita $\langle M, w\rangle$ se e somente se $M$ aceita $w$ e $U$ rejeita $\langle M, w\rangle$ se e somente se $M$ rejeita $w$, ou seja, $U$ semidecide a linguagem { $\langle M, w\rangle$ tal que $M$ aceita $w$ }.*

*Observação.* O Teorema 2.2 não será provado neste texto. A sua prova é extensa e trabalhosa. A ideia da prova é definir como simbolizar máquinas de Turing determinísticas de uma única fita e construir uma máquina de Turing determinística de uma única fita que tem como entrada qualquer uma destas simbolizações e simula os movimentos, ou seja, as funções de transição, das máquinas simbolizadas.

**Exemplo 2.1.2.** Uma máquina de Turing Universal é o simulador desenvolvido para a elaboração dos exemplos, cujo código-fonte está disponível e documentado no Apêndice A.

**Teorema 2.3** (Problema da Parada[2])**.** *Não existe uma máquina de Turing determinística de uma única fita $H$ que decide a linguagem semidecidida pela máquina de Turing Universal (Teorema 2.2).*

*Prova.* Prova por contradição. Assuma que exista uma máquina $H$ que decida { $\langle M, w\rangle$ tal que $M$ aceita $w$ }. Pelo Teorema 2.1, podemos construir uma máquina $H'$ que decide { $\langle M, w\rangle$ tal que $M$ rejeita $w$ }. Podemos construir uma máquina $H''$ com base em $H'$ que decide { $\langle M\rangle$ tal que $M$ rejeita $\langle M\rangle$ } (prova trivial). Mas seja $\langle H''\rangle$ a simbolização de $H''$, $H''$ aceita $\langle H''\rangle$ se e somente se $H''$ rejeita $\langle H''\rangle$. □

---

[2] O Problema da Parada mostra que nem toda linguagem é decidível, ou seja, nem tudo é computável. Muitos dizem que Turing formulou e provou a indecidibilidade do Problema da Parada, porém as máquinas originais propostas por Turing, não decidem uma linguagem, mas, computam números, e, na forma definida por Turing, só computam quando não param. Martin Davis acredita que foi o primeiro a usar o termo em aulas dadas em 1952 (COPELAND, 2004, p. 40, nota de rodapé 61) e Stephen Kleene, embora não tenha definido um "Problema da Parada", definiu, também em 1952, máquinas de Turing que computam quando param (KLEENE, 1952, cap. 13, p. 382). Mas apenas a formulação é mal atribuída, o resultado é de Turing, que mostrou que nem tudo é computável, definindo dois números que dependem de uma máquina paradoxal para serem computados.

**Definição 2.1.13.** O **tempo de execução** de uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ para uma entrada $w \in \Sigma^*$ é o número de movimentos necessários para a configuração inicial de $M$ para $w$ originar uma configuração final de $M$ se e somente se esta origina uma configuração final de $M$, 0 se e somente se esta é uma configuração final de $M$ e indefinido se e somente se esta não é e nem origina uma configuração final de $M$.

**Definição 2.1.14.** A **complexidade de tempo** de uma máquina de Turing determinística de uma única fita $M$ é a função $f : \mathbb{N} \to \mathbb{N}$ tal que $f(n)$ é o máximo entre os tempos de execução de $M$ para as entradas de $M$ de tamanho $n$.

**Definição 2.1.15.** Uma função $g : \mathbb{N} \to \mathbb{R}^+$ é um **limitante superior assintótico** de uma função $f : \mathbb{N} \to \mathbb{R}^+$ se e somente se existem $c$ e $n_0 \in \mathbb{N}$ tal que $f(n) \leq c \cdot g(n)$ para todo $n \geq n_0$.

**Definição 2.1.16.** O conjunto $\mathcal{O}[g(n)]$ é o conjunto de todas as funções $f : \mathbb{N} \to \mathbb{R}^+$ para as quais a função $g : \mathbb{N} \to \mathbb{R}^+$ é um limitante superior assintótico.

**Definição 2.1.17.** A **classe de complexidade de tempo** $DTIME[t(n)]$ é o conjunto das linguagens decididas por máquinas de Turing determinísticas com complexidade de tempo $f(n) \in \mathcal{O}[t(n)]$.

**Definição 2.1.18.** A classe de complexidade $\mathcal{P}$ é conjunto das **linguagens decididas em tempo polinomial** por máquinas de Turing determinísticas de uma única fita:

$$\mathcal{P} = \bigcup_{k \in \mathbb{N}} DTIME[n^k]$$

## 2.2 MÁQUINAS DE TURING DETERMINÍSTICAS MULTIFITA

Nesta subseção, definiremos o modelo de computação da máquina de Turing determinística multifita para a decisão de problemas. As definições não refeitas são as mesmas do modelo de computação da máquina de Turing determinística de uma única fita, feitas na subseção anterior.

**Definição 2.2.1.** Uma **máquina de Turing determinística de *k* fitas**, $k \in \mathbb{N}$, é uma máquina de Turing, assim como a da Definição 2.1.1, mas onde:

$$\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{\triangleleft, \circ, \triangleright\})^k$$

**Definição 2.2.2.** Uma **configuração** $c$ de uma máquina de Turing determinística de $k$ fitas $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ é uma tripla $((\alpha_1, \alpha_2, \ldots, \alpha_k), q, (s_1\beta_1, s_2\beta_2, \ldots, s_k\beta_k))$ tal que $q \in Q$ e para todo $i \in [1, k] \cap \mathbb{N}$, $s_i \in \Gamma$ e $\alpha_i$ e $\beta_i \in \Gamma^*$. Dizemos que $M$ está em $c$ quando para $M$:

- $q$ é o estado atual,
- $s_i$ é o símbolo atual da $i$-ésima fita,
- $\alpha_i$ são os símbolos da $i$-ésima fita entre o primeiro símbolo diferente de $B$ (incluindo) e $s_i$ (não incluindo),
- $\beta_i$ são os símbolos da $i$-ésima fita entre $s_i$ (não incluindo) e último símbolo diferente de $B$ (incluindo).

**Definição 2.2.3.** Um **movimento** $c_1 \vdash_M c_2$ de uma máquina de Turing determinística de $k$ fitas $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ é a aplicação da função de transição:

$$\delta(q, (s_1, s_2, \ldots, s_k)) = (p, ((s'_1, X_1), (s'_2, X_2), \ldots, (s'_k, X_k)))$$

a alguma configuração

$$c_1 = ((\alpha_1 e_1, \alpha_2 e_2, \ldots, \alpha_k e_k), q, (s_1 d_1 \beta_1, s_2 d_2 \beta_2, \ldots, s_k d_k \beta_k))$$

de $M$ tal que $c_2 = ((\alpha'_1, \alpha'_2, \ldots, \alpha'_k), p, (\beta'_1, \beta'_2, \ldots, \beta'_k))$ e para todo $i \in [1, k] \cap \mathbb{N}$, $s_i \in \Gamma$, $e_i$ e $d_i \in (\Gamma \cup \{\varepsilon\})$, $\alpha_i$ e $\beta_i \in \Gamma^*$ e:

- $\alpha'_i = \alpha_i e_i$ e $\beta'_i = s'_i d_i \beta_i$ se e somente se $X_i = \circ$,
- $\alpha'_i = \alpha_i$ e $\beta'_i = e_i s''_i d_i \beta_i$ se e somente se $X_i = \triangleleft$ e $e_i \neq \varepsilon$,
- $\alpha'_i = \alpha_i e_i s''_i$ e $\beta'_i = d_i \beta_i$ se e somente se $X_i = \triangleright$ e $d_i \neq \varepsilon$,
- $\alpha'_i = \varepsilon$, $\beta'_i = B s''_i d_i \beta_i$ e $\alpha_i = \varepsilon$ se e somente se $X_i = \triangleleft$ e $e_i = \varepsilon$,
- $\alpha'_i = \alpha_i e_i s''_i$, $\beta'_i = B$ e $\beta_i = \varepsilon$ se e somente se $X_i = \triangleright$ e $d_i = \varepsilon$

tal que $s''_i \in (\Gamma \cup \{\varepsilon\})$ é igual a $\varepsilon$ se e somente se $s'_i = B$, $X_i = \triangleleft$ e $d_i \beta_i = \varepsilon$ ou $s'_i = B$, $X_i = \triangleright$ e $\alpha_i e_i = \varepsilon$, caso contrário, $s''_i$ é igual a $s'_i$.

**Definição 2.2.4.** A **configuração inicial** de uma máquina de Turing determinística de $k$ fitas $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ para uma entrada $w \in \Sigma^*$ é a configuração:

- $((\underbrace{\varepsilon, \varepsilon, \ldots, \varepsilon}_{k}), q_0, (w, \underbrace{B, B, \ldots, B}_{k-1}))$ se e somente se $w \neq \varepsilon$,
- $((\underbrace{\varepsilon, \varepsilon, \ldots, \varepsilon}_{k}), q_0, (\underbrace{B, B, \ldots, B}_{k}))$ se e somente se $w = \varepsilon$.

**Exemplo 2.2.1.** Uma máquina de Turing determinística de duas fitas $M$ que decide a linguagem { $w$ tal que $w$ é uma cadeia do tipo 001011011101111011111 ... }:

$$M = (\{q_0, q_v, q_c, q_a, q_r\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_a\})$$

tal que $q_v$ representa "volta", $q_c$ representa "compara", $q_a$ representa "aceita", $q_r$ representa "rejeita" e $\delta(q, (s_1, s_2))$ é definida pelo Quadro 2. Na Figura 1 é ilustrada a aceitação

de uma entrada por $M$. Cada configuração é representada por um quadro tal que o seu estado está na coluna da esquerda e a coluna da direita possui uma linha para cada fita. Na $i$-ésima linha desta segunda coluna, representamos a $i$-ésima fita como a concatenação de $\alpha_i$, $s_i$ e $\beta_i$, destacando $s_i$, com a cor vermelha e uma sublinha. Cada configuração aponta para a configuração que é originada por ela por um movimento.

| $q$ \ $s_1s_2$ | 00 | 01 | 0$B$ |
|---|---|---|---|
| $q_0$ | $(q_r, ((0,\circ),(0,\circ)))$ | $(q_r, ((0,\circ),(1,\circ)))$ | $(q_c, ((0,\triangleright),(B,\circ)))$ |
| $q_v$ | $(q_v, ((0,\circ),(0,\triangleleft)))$ | $(q_v, ((0,\circ),(1,\triangleleft)))$ | $(q_c, ((0,\circ),(B,\triangleright)))$ |
| $q_c$ | $(q_r, ((0,\circ),(0,\circ)))$ | $(q_r, ((1,\circ),(B,\circ)))$ | $(q_v, ((0,\triangleright),(1,\circ)))$ |
| $q$ \ $s_1s_2$ | 10 | 11 | 1$B$ |
| $q_0$ | $(q_r, ((1,\circ),(0,\circ)))$ | $(q_r, ((1,\circ),(1,\circ)))$ | $(q_r, ((1,\circ),(B,\circ)))$ |
| $q_v$ | $(q_v, ((1,\circ),(0,\triangleleft)))$ | $(q_v, ((1,\circ),(1,\triangleleft)))$ | $(q_c, ((1,\circ),(B,\triangleright)))$ |
| $q_c$ | $(q_r, ((1,\circ),(0,\circ)))$ | $(q_c, ((1,\triangleright),(1,\triangleright)))$ | $(q_r, ((1,\circ),(B,\circ)))$ |
| $q$ \ $s_1s_2$ | $B0$ | $B1$ | $BB$ |
| $q_0$ | $(q_r, ((B,\circ),(0,\circ)))$ | $(q_r, ((B,\circ),(1,\circ)))$ | $(q_r, ((B,\circ),(B,\circ)))$ |
| $q_v$ | $(q_v, ((B,\circ),(0,\triangleleft)))$ | $(q_v, ((B,\circ),(1,\triangleleft)))$ | $(q_c, ((B,\circ),(B,\triangleright)))$ |
| $q_c$ | $(q_r, ((B,\circ),(0,\circ)))$ | $(q_r, ((B,\circ),(1,\circ)))$ | $(q_a, ((B,\circ),(B,\circ)))$ |

Quadro 2 – Imagens da função de transição da máquina do Exemplo 2.2.1. Fonte: adaptado do segundo exemplo de Turing (1936).


Figura 1 – A aceitação da entrada 0010110111 pela máquina do Exemplo 2.2.1. Fonte: elaborada pelo autor.

**Teorema 2.4.** *Se uma máquina de Turing determinística multifita $M$ semidecide uma linguagem $L$, então podemos construir uma máquina de Turing determinística de uma única fita $S$ que também semidecide $L$.*

*Observação.* A prova do Teorema 2.4 consiste em construir uma máquina de Turing a partir de outra, assim como fizemos na prova do Teorema 2.1, sobre o complemento de linguagens decidíveis, mas desta vez não vamos explicitar esta construção, o que seria extenso, trabalhoso e pouco intuitivo. Vamos apenas dar uma ideia geral de como a máquina construída funcionaria.

*Prova.* $S$ organiza os símbolos das fitas de $M$ em sua única fita dividindo-os com um símbolo separador e substitui os símbolos atuais das fitas por símbolos especiais que podem ser os símbolos atuais com algum marcador[3]. Para cada movimento de $M$, $S$ vai do primeiro símbolo separador até o último para determinar os símbolos atuais de $M$ para, então, simular o movimento de $M$ em cada símbolo atual. Se, por esse movimento, $S$ precisa substituir, ou seja, marcar, algum símbolo separador que está à esquerda (ou à direita) de algum dos símbolos atuais, então $S$ substitui o símbolo separador por um símbolo em branco marcado e desloca para a esquerda (ou para a direita) todos os símbolos à esquerda (ou à direita) deste separador, colocando um símbolo separador entre eles e o símbolo em branco marcado. □

## 2.3 MÁQUINAS DE TURING NÃO DETERMINÍSTICAS

Nesta subseção, definiremos o modelo de computação da máquina de Turing não determinística para a decisão de problemas. As definições não refeitas são as mesmas do modelo de computação da máquina de Turing determinística de uma única fita, feitas na Subseção 2.1.

**Definição 2.3.1.** Uma **máquina de Turing não determinística** $M$ é uma máquina de Turing, assim como a da Definição 2.1.1, mas onde:

$$\delta : Q \times \Gamma \to \mathscr{P}[Q \times \Gamma \times \{\triangleleft, \circ, \triangleright\}]$$

em que $\mathscr{P}[C]$ denota o conjunto potência do conjunto $C$.

**Definição 2.3.2.** Um **movimento** de uma máquina de Turing não determinística $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ é a aplicação da função de transição $\delta$ a alguma configuração $c_1 = (\alpha e, q, sd\beta)$ de $M$ tal que $s \in \Gamma$, $e$ e $d \in (\Gamma \cup \{\varepsilon\})$, $\alpha$ e $\beta \in \Gamma^*$ e para todo $(p, s', X) \in \delta(q, s)$:

---

[3] Precisamos apenas de uma função $f$ injetora entre os símbolos da fita de $M$ e estes símbolos especiais. Nesse caso, estamos usando um marcador como $f$, para cada símbolo $s$ da fita de $M$, $f(s)$ é igual a $s$ com este marcador. Por exemplo, usando uma sobrelinha para todo símbolo $s$ da fita de $M$, $f(s) = \bar{s}$.

- se $X = \circ$ então $c_1 \vdash_M (\alpha e, p, s'd\beta)$,
- se $X = \triangleleft$ e $e \neq \varepsilon$ então $c_1 \vdash_M (\alpha, p, es''d\beta)$,
- se $X = \triangleright$ e $d \neq \varepsilon$ então $c_1 \vdash_M (\alpha es'', p, d\beta)$,
- se $X = \triangleleft$ e $e = \varepsilon$ então $\alpha = \varepsilon$ e $c_1 \vdash_M (\varepsilon, p, Bs''d\beta)$,
- se $X = \triangleright$ e $d = \varepsilon$ então $\beta = \varepsilon$ e $c_1 \vdash_M (\alpha es'', p, B)$

tal que $s'' \in (\Gamma \cup \{\varepsilon\})$ é igual a $\varepsilon$ se e somente se $s' = B$, $X = \triangleleft$ e $d\beta = \varepsilon$ ou $s' = B$, $X = \triangleright$ e $\alpha e = \varepsilon$, caso contrário, $s''$ é igual a $s'$.

**Definição 2.3.3.** Uma **árvore de computação** de uma máquina de Turing não determinística $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ para alguma entrada $w \in \Sigma^*$ é uma árvore de configurações de $M$ tal que:

1. a raiz da árvore é a configuração inicial de $M$ para $w$,
2. se $c$ não é uma folha, então toda $c'$ tal que $c \vdash_M c'$ está na árvore,
3. para todo caminho $c_1, c_2, \ldots, c_n$, $n \in [2, \infty) \cap \mathbb{N}$, a partir da raiz da árvore, $c_i \vdash_M c_{i+1}$ para todo $i \in [1, n-1] \cap \mathbb{N}$,
4. pelo menos uma folha é uma configuração de aceitação de $M$ ou todas as folhas são configurações de rejeição de $M$.

**Definição 2.3.4.** Uma máquina de Turing não determinística $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ **rejeita** uma entrada $w \in \Sigma^*$ se e somente se existe uma árvore de computação de $M$ para $w$ tal que todas as folhas são configurações de rejeição de $M$.

**Exemplo 2.3.1.** Seja $S = \{i_1, i_2, \ldots, i_n\}$ uma sequência de inteiros e $\langle S \rangle$ uma simbolização de $S$ como $\langle i_1 \rangle \# \langle i_2 \rangle \# \ldots \# \langle i_n \rangle$, $n \in \mathbb{N}$, tal que para todo $i_j$, $j \in [1, n] \cap \mathbb{N}$, $\langle i_j \rangle$ simboliza $i_j$ como $(+)^{i_j}$ se e somente se $i_j \geq 0$ e $(-)^{-i_j}$ se e somente se $i_j < 0$ (p.ex. $+++$ simboliza o 3 e $---$ simboliza o $-3$)[4]. Uma máquina de Turing não determinística $M$ que decide a linguagem { $\langle S \rangle$ tal que a soma dos inteiros de alguma subsequência não vazia de $S$ é igual a 0 }:

$$M = (\{q_0, q_i, q_e, q_p, q_c, q_{m+}, q_{m-}, q_v, q_a, q_r\}, \{\#, +, -\}, \{\#, +, -, E, B\}, \delta, q_0, B, \{q_a\})$$

tal que $q_i$ representa "inclui", $q_e$ representa "exclui", $q_p$ representa "prepara", $q_c$ representa "calcula", $q_{m+}$ representa "marca +", $q_{m-}$ representa "marca −", $q_v$ representa "volta", $q_a$ representa "aceita", $q_r$ representa "rejeita" e $\delta(q, s)$ é definida pelo Quadro 3, na próxima página, no qual, para facilitar a leitura, estamos omitindo as chaves dos conjuntos das imagens, por exemplo, para $q = q_0$ e $s = \#$, leia $\delta(q_0, \#) = \{(q_i, \#, \triangleright), (q_e, E, \triangleright)\}$. Uma árvore de computação de $M$ é ilustrada na Figura 2, utilizando a mesma representação de configurações da Figura 1.

---

[4] Esta é uma simplificação da simbolização que estamos utilizando, já que também permitimos a intercalação de + e −. Por exemplo, $++{+}{-}{-}{+} = \langle 2 \rangle$.

| $q$ \ $s$ | $\#$ | $+$ | $-$ | $E$ | $B$ |
|---|---|---|---|---|---|
| $q_0$ | $(q_i, \#, \triangleright)$, $(q_e, E, \triangleright)$ | $(q_i, +, \triangleright)$, $(q_e, E, \triangleright)$ | $(q_i, -, \triangleright)$, $(q_e, E, \triangleright)$ | $(q_i, E, \triangleright)$, $(q_e, E, \triangleright)$ | $(q_r, B, \circ)$ |
| $q_i$ | $(q_0, E, \circ)$ | $(q_i, +, \triangleright)$ | $(q_i, -, \triangleright)$ | $(q_r, E, \circ)$ | $(q_p, B, \triangleleft)$ |
| $q_e$ | $(q_0, E, \circ)$ | $(q_e, E, \triangleright)$ | $(q_e, E, \triangleright)$ | $(q_r, E, \circ)$ | $(q_p, B, \triangleleft)$ |
| $q_p$ | $(q_p, \#, \triangleleft)$ | $(q_c, +, \circ)$ | $(q_c, -, \circ)$ | $(q_p, E, \triangleleft)$ | $(q_r, B, \circ)$ |
| $q_c$ | $(q_c, \#, \triangleleft)$ | $(q_{m-}, E, \triangleleft)$ | $(q_{m+}, E, \triangleleft)$ | $(q_c, E, \triangleleft)$ | $(q_a, B, \circ)$ |
| $q_{m+}$ | $(q_{m+}, \#, \triangleleft)$ | $(q_v, E, \triangleright)$ | $(q_{m+}, -, \triangleleft)$ | $(q_{m+}, E, \triangleleft)$ | $(q_r, B, \circ)$ |
| $q_{m-}$ | $(q_{m-}, \#, \triangleleft)$ | $(q_{m-}, +, \triangleleft)$ | $(q_v, E, \triangleright)$ | $(q_{m-}, E, \triangleleft)$ | $(q_r, B, \circ)$ |
| $q_v$ | $(q_v, \#, \triangleright)$ | $(q_v, +, \triangleright)$ | $(q_v, -, \triangleright)$ | $(q_v, E, \triangleright)$ | $(q_c, B, \triangleleft)$ |

Quadro 3 – Imagens da função de transição da máquina do Exemplo 2.3.1. Fonte: elaborado pelo autor.

**Teorema 2.5.** *Se uma máquina de Turing não determinística $M$ semidecide uma linguagem $L$, então podemos construir uma máquina de Turing determinística de três fitas $S$ que também semidecide $L$.*

*Observação.* A prova do Teorema 2.5 consiste em construir uma máquina de Turing a partir de outra, assim como fizemos na prova do Teorema 2.1, sobre o complemento de linguagens decidíveis, mas desta vez, assim como na prova do Teorema 2.4 sobre a equivalência entre as máquinas de Turing determinísticas de uma única fita e multifita, e pelos mesmos motivos, não vamos explicitar esta construção, apenas dar uma ideia geral de como a máquina construída funcionaria.

*Prova.* Em sua primeira fita, $S$ deixa a entrada. Em sua segunda fita $S$ organiza uma fila de configurações simbolizadas de uma maneira que permita que $S$ adicione uma configuração, remova uma configuração e verifique se não há nenhuma configuração nesta fila. Inicialmente, $S$ adiciona a configuração inicial de $M$ para a entrada a fila da segunda fita. Então, para cada configuração desta fila, $S$ a troca[5] pelas configurações originadas, em sua terceira fita, a partir dela por um movimento. Se em algum momento nestas trocas, $S$ originar alguma configuração de aceitação de $M$, então $S$ aceita a entrada e se isso não acontecer e a fila ficar vazia, então $S$ rejeita a entrada. □

**Corolário 2.5.1.** *Se uma máquina de Turing não determinística $M$ semidecide uma linguagem $L$, então podemos construir uma máquina de Turing determinística de uma única fita $S$ que também semidecide $L$.*

*Prova.* Pelo Teorema 2.5, podemos construir uma máquina de Turing de três fitas $S'$ que semidecide $L$ e portanto, pelo Teorema 2.4, podemos construir uma máquina de Turing de uma única fita $S$ que semidecide $L$. □

---

[5] Já que estamos trabalhando com uma fila, trocar uma configuração $c$ pelas configurações do conjunto $\{ c_1, c_2, \ldots, c_n \}$, $n \in \mathbb{N}$, significa que $c$ é a primeira configuração da fila e que as configurações deste conjunto, se esse não for vazio, serão inseridas no final da fila.

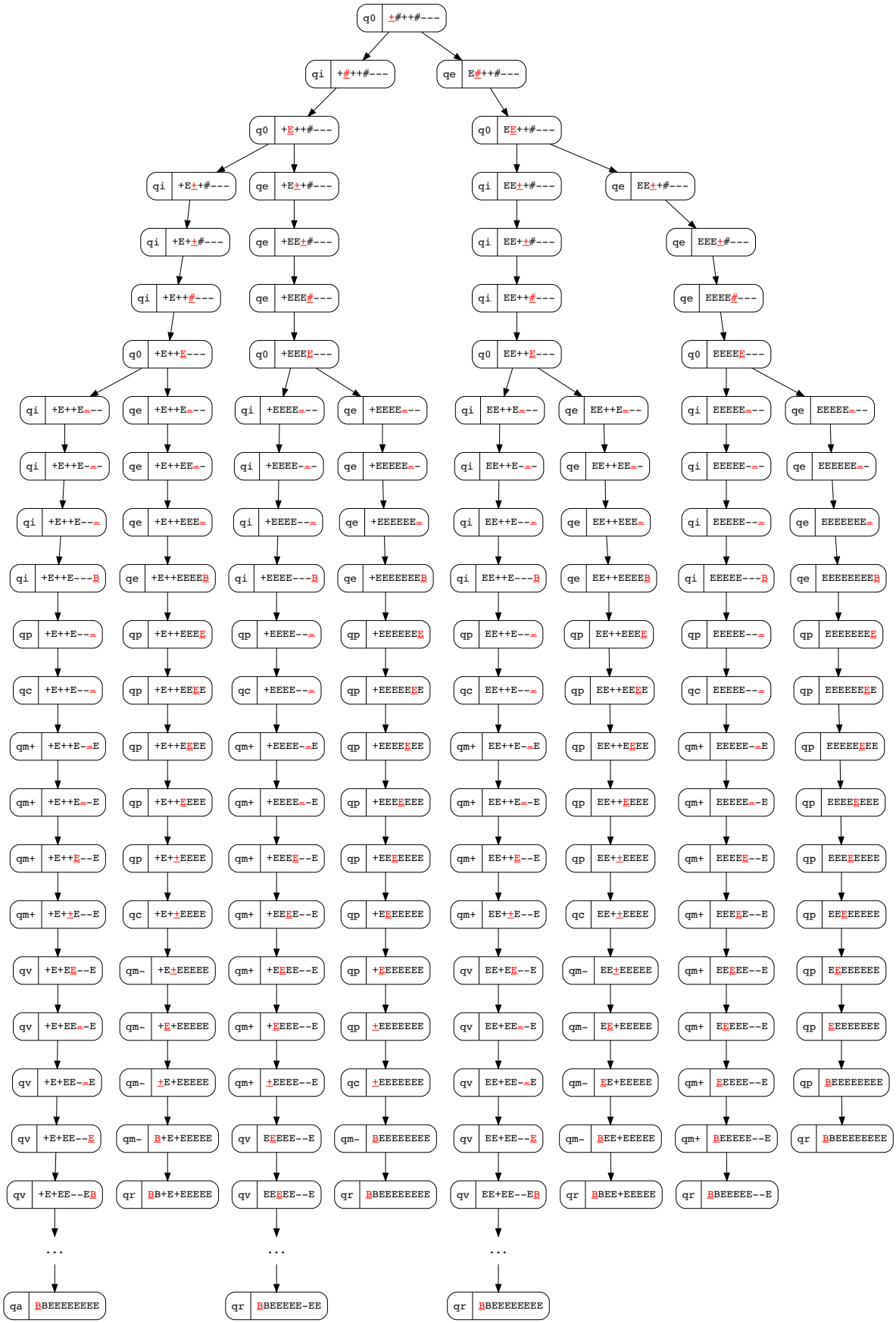


Figura 2 – Uma árvore de computação da máquina do Exemplo 2.3.1 para a entrada +#++#−−− que simboliza a sequência {1, 2, -3}. Fonte: elaborada pelo autor.

**Definição 2.3.5.** O **tempo de execução** de uma máquina de Turing não determinística $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ para uma entrada $w \in \Sigma^*$ é a altura da árvore de computação de $M$ para $w$ se e somente se existe uma árvore de computação de $M$ para $w$, caso contrário, é indefinido.

**Definição 2.3.6.** A **classe de complexidade de tempo** $NTIME[t(n)]$ é o conjunto das linguagens decididas por máquinas de Turing não determinísticas com complexidade de tempo $f(n) \in \mathcal{O}[t(n)]$.

**Definição 2.3.7.** A classe de complexidade $\mathcal{NP}$ é o conjunto das **linguagens decididas em tempo polinomial** por máquinas de Turing não determinísticas:

$$\mathcal{NP} = \bigcup_{k \in \mathbb{N}} NTIME[n^k]$$

# 3 COMPUTAÇÃO DE FUNÇÕES

*Do you have the time to listen to me whine?*
*(Green Day)*

Nesta seção, definiremos como computar uma função, de forma determinística, com uma máquina de Turing determinística de uma única fita e discutiremos as definições existentes, porém insatisfatórias, de como computar uma função, de forma não determinística, com uma máquina de Turing não determinística. Todas as definições feitas nesta seção foram adaptadas para a notação desenvolvida e utilizada na seção anterior.

## 3.1 COMPUTAÇÃO DETERMINÍSTICA DE FUNÇÕES

A solução[1] para um problema de decisão é uma resposta que pode ser "sim" ou "não", ou seja, ela pertence a um conjunto solução, { "sim", "não" }, de cardinalidade dois. Já a solução para uma função é uma saída dessa função, que pode ser qualquer elemento de seu contradomínio[2], ou seja, a cardinalidade do conjunto solução de uma função depende da cardinalidade de seu contradomínio.[3]

Conseguimos decidir sobre uma instância de um problema de decisão com uma máquina de Turing determinística se e somente se a configuração inicial para uma entrada que simboliza esta instância origina uma configuração final. Obtemos a solução desta instância a partir do estado desta configuração final: a resposta é "sim" e a máquina aceita se e somente se este estado pertence ao conjunto de estados de aceitação e a resposta é "não" e a máquina rejeita se e somente se este estado não pertence ao conjunto de estados de aceitação.

Para uma máquina de Turing determinística $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$, esta obtenção pode ser descrita pela função $f : Q \to$ { "sim", "não" } tal que para todo $q \in Q$, $f(q)$ é igual a "sim" se e somente se $q \in A$ e "não" se e somente se $q \notin A$. De modo geral, utilizando esta estratégia para obter uma solução de algum conjunto solução a partir de um estado de uma máquina de Turing determinística, precisamos de uma função sobrejetora em que o domínio seja o conjunto de estados e o contradomínio seja o conjunto solução.

Porém, para computação de funções, esta estratégia possui dois problemas. O primeiro é que ela não é intuitiva, pois precisaríamos ter um estado para cada saída da função, o que tornaria demasiada a função de transição. O segundo, e mais crítico, é que não poderíamos trabalhar com funções em que o contradomínio é um conjunto infinito, já que a cardinalidade do domínio de uma função sobrejetora é maior ou igual a cardinalidade de

---

[1] Normalmente, na matemática, quando falamos de soluções estamos falando de soluções para equações, mas aqui estamos usando a palavra em um sentido mais informal.

[2] Ou *é* qualquer elemento de sua *imagem*.

[3] Quando dizemos *a* solução de um problema de decisão (ou função), estamos nos referindo às diversas soluções para as diversas instâncias desse problema (ou função).

seu contradomínio e a cardinalidade do conjunto de estados de uma máquina de Turing é finito[4].

Para adaptar o modelo da máquina de Turing determinística de uma única fita para computar uma função, os autores dos livros utilizados como inspiração para a seção anterior utilizam outra estratégia. Eles obtém a solução, a saída da função, a partir, não do estado mas do que chamamos de conteúdo da fita de uma configuração:

**Definição 3.1.1.** O **conteúdo da fita** de uma configuração $c = (\alpha,\ \mathrm{q},\ s\beta)$ de uma máquina de Turing determinística de uma única fita $M = (Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ B,\ A)$ são os símbolos da fita de $M$ entre o primeiro símbolo diferente de $B$ (incluindo) e o último símbolo diferente de $B$ (incluindo).

Mas, mesmo que utilizando a mesma estratégia, os autores divergem. Sipser obtém a saída da função diretamente do conteúdo da fita e restringe as funções computáveis às máquinas que sempre decidem:

**Definição 3.1.2.** A função computada por uma máquina de Turing determinística de uma única fita $M = (Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ B,\ A)$ é a função $f : \Sigma^* \to \Sigma^*$ se e somente se para toda entrada $x \in \Sigma^*$, a configuração inicial de $M$ para $x$ é ou origina uma configuração final de $M$ com o conteúdo da fita igual a $f(x) \in \Sigma^*$. (SIPSER, 2007)

O principal problema desta restrição é que ela não permite adaptar problemas indecidíveis para funções. Não poderíamos, por exemplo, ter uma função análoga à máquina de Turing Universal (Teorema 2.2), por causa do Problema da Parada (Teorema 2.3). Hopcroft e Ullman incluem essas funções definindo as funções computáveis como funções parciais:

**Definição 3.1.3.** Uma **função parcial** $f : D \to C$ é uma função que não está necessariamente defininda para todo $x \in D$, ou seja, pode estar parcialmente definida. Quando $f$ não está definida para um $x \in D$ dizemos que $f(x)$ é indefinida. (HOPCROFT; ULLMAN, 1979)

**Definição 3.1.4.** Uma **função total** é uma função parcial $f : D \to C$ tal que para todo $x \in D$, $f(x)$ é definida. (HOPCROFT; ULLMAN, 1979)

**Exemplo 3.1.1.** A função $f : \mathbb{R} \to \mathbb{R}$ tal que para todo $x \in \mathbb{R}$ diferente de 0, $f(x) = \frac{1}{x}$ é uma função parcial, pois $f(0)$ é indefinida.

**Exemplo 3.1.2.** A função *passos* tal que para uma máquina de Turing determinística de uma única fita $M = (Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ B,\ A)$ e uma entrada $w \in \Sigma^*$ de $M$, $passos(M, w)$ é o tempo de execução de $M$ para $w$, é uma função parcial, pois não está definida quando a configuração inicial de $M$ para $w$ não é e nem origina uma configuração final de $M$. (RICH, 2008, p. 555)

---

[4] Por causa do primeiro problema, não vamos explorar uma solução para o segundo.

**Exemplo 3.1.3.** Todas as funções de transição das máquinas dos exemplos da seção anterior são funções parciais, pois, se não fossem, estas não teriam configurações finais. Por exemplo, $\delta(q_a, s)$ e $\delta(q_r, s)$ não estão definidas para nenhum $s \in \Gamma$ no Exemplo 2.1.1.

**Definição 3.1.5.** Uma função $k$-ária parcial $f : \mathbb{N}^k \to \mathbb{N}$, $k \in \mathbb{N}$, é computada por uma máquina de Turing determinística de uma única fita $M = (Q, \{0,1\}, \Gamma, \delta, q_0, B, A)$ se e somente se para todo $i_1, i_2, \ldots, i_k \in \mathbb{N}$ tal que $f(i_1, i_2, \ldots, i_k)$ é definida, a configuração inicial de $M$ para $0^{i_1}10^{i_2}1\cdots0^{i_k}$ é ou origina uma configuração final de $M$ com o conteúdo da fita igual a $0^{f(i_1,i_2,\ldots,i_k)}$ e para todo $i_1, i_2, \ldots, i_k \in \mathbb{N}$ tal que $f(i_1, i_2, \ldots, i_k)$ é indefinida, a configuração inicial de $M$ para $0^{i_1}10^{i_2}1\cdots0^{i_k}$ não é e nem origina uma configuração final de $M$. (HOPCROFT; ULLMAN, 1979)

Além de parciais, as funções computáveis definidas por Hopcroft e Ullman são funções $k$-árias. E, para eles, uma máquina de Turing pode computar uma função de um argumento, uma função diferente de dois argumentos etc. (trataremos sobre este aspecto no final desta subseção), em que os domínios são algum produto cartesiano dos números naturais. Mas, embora seja possível, por exemplo, com uma enumeração de Gödel (NAGEL; NEWMAN, 2001), associar um número a qualquer objeto matemático, esta restrição é desnecessária e acrescenta uma etapa na definição/programação de funções com máquinas de Turing.

A divergência no domínio e contradomínio, $\Sigma^* \to \Sigma^*$ na definição de Sipser e $\mathbb{N}^k \to \mathbb{N}$ na definição de Hopcroft e Ullman, é classificada por Sudkamp (1997), que chama de numéricas as funções computáveis definidas por Hopcroft e Ullman. Em contrapartida, vamos chamar de simbólicas as funções computáveis definidas por Sipser.

Um problema das duas definições é que ambas são do tipo $D^k \to D$, em que $k$ é igual a 1 na definição de Sipser e $D$ é igual a $\mathbb{N}$ na definição de Hopcroft e Ullman, e, por isso, não permitem funções do tipo $D \to C$ em que $D$ é diferente de $C$ e $C$ é diferente de $D^k$, ou seja, sempre precisamos usar o mesmo alfabeto para o formato de entrada e para o formato de saída, $\Sigma$ na definição de Sipser e $\{0, 1\}$ na definição de Hopcroft e Ullman.

Além disso, nenhuma delas define a saída da função quando o conteúdo da fita não está no formato de saída adequado, $\Sigma^*$ na definição de Sipser e $0^m$ na definição Hopcroft e Ullman[5]. Por isso, neste trabalho vamos definir, além de um alfabeto de entrada e da fita, um alfabeto de saída. Neste, teremos todos os símbolos que podem pertencer as saídas de nossas funções. Também vamos retirar o conjunto de estados de aceitação, pois estes são desnecessários para a computação de funções:

---

[5] Dependendo da definição/programação da máquina, a saída vai sempre estar neste formato, mas não defini-la quando isto não acontence, ou seja, mesmo quando uma configuração inicial origina uma configuração final (com o contéudo da fita fora do formato de saída), é um ponto fraco das definições.

**Definição 3.1.6.** Uma máquina de Turing determinística de uma única fita *para funções* é definida assim como a da Definição 2.1.1, mas sem a definição de um conjunto de aceitação e com uma componente adicional, o conjunto do alfabeto de saída $\Upsilon$:

$$M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B)$$

tal que $\Upsilon \subseteq \Gamma$.

**Definição 3.1.7.** O **conteúdo de saída da fita** de uma configuração $c = (\alpha, \mathrm{q}, s\beta)$ de uma máquina de Turing determinística de uma única fita $M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B)$ são todos os símbolos pertecentes a $\Upsilon$ do conteúdo da fita de $c$ se e somente se este possui símbolos pertecentes a $\Upsilon$, caso contrário, é igual a $\varepsilon$.[6]

E definiremos as funções computáveis como funções parciais, assim como Hopcroft e Ullman, simbólicas, assim como Sipser, e utilizando um alfabeto de saída:

**Definição 3.1.8.** A função computada por uma máquina de Turing determinística de uma única fita *para funções* $M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B)$ é a função parcial $f : \Sigma^* \to \Upsilon^*$ tal que para toda entrada $x \in \Sigma^*$, $f(x)$ é igual a $y$ se e somente se a configuração inicial de $M$ para $x$ é ou origina uma configuração final de $M$ com o conteúdo de saída da fita igual a $y$, caso contrário, $f(x)$ é indefinida.

**Definição 3.1.9.** A **função característica** de um conjunto $D$ é a função $f : D \to \{ 0, 1 \}$ tal que $f(x)$ é igual a 1 se e somente se $x \in D$ e 0 caso contrário. (SIPSER, 2007)

**Exemplo 3.1.4.** Já que linguagens são conjuntos, usamos funções características para adaptar diretamente problemas de decisão para funções. Para adaptar uma máquina de Turing determinística de uma única fita que (semi)decide uma linguagem para que ela compute a função característica desta linguagem, basta fazer com que todas suas configurações de aceitação e de rejeição possuam como conteúdo de saída da fita, respectivamente, o 1 e o 0. (SIPSER, 2007)

**Exemplo 3.1.5.** Seja $n \in \mathbb{N}$, $\Sigma_M = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ e $\langle n \rangle \in \Sigma_M^*$ a simbolização de $n$ na base dez. Uma máquina de Turing determinística de uma única fita $M$ que computa a função $S : \Sigma_M^* \to \Sigma_M^*$ tal que para todo $n \in \mathbb{N}$, $S(\langle n \rangle) = \langle (n + 1) \rangle$ e $S(\varepsilon) = \langle 1 \rangle$:

$$M = (\{q_0, q_d, q_s, q_p\}, \Sigma_M, \Sigma_M \cup \{B\}, \Sigma_M, \delta, q_0, B)$$

tal que $q_d$ representa "direita", $q_s$ representa "soma", $q_p$ representa "para" e $\delta(q, s)$ é definida pelo Quadro 4.

---

[6] Esta definição muda como definimos/programamos máquinas de Turing. Neste trabalho, optamos por uma abordagem semelhante aos *F-squares* e *E-squares* de Turing, porém sem a restrição de que eles se alternem, a nossa distinção é feita pela pertinência do símbolo ao alfabeto de saída.

| $s$ \ $q$ | $q_0$ | $q_d$ | $q_s$ |
|---|---|---|---|
| 0 | $(q_d, 0, \triangleright)$ | $(q_d, 0, \triangleright)$ | $(q_p, 1, \circ)$ |
| 1 | $(q_d, 1, \triangleright)$ | $(q_d, 1, \triangleright)$ | $(q_p, 2, \circ)$ |
| 2 | $(q_d, 2, \triangleright)$ | $(q_d, 2, \triangleright)$ | $(q_p, 3, \circ)$ |
| 3 | $(q_d, 3, \triangleright)$ | $(q_d, 3, \triangleright)$ | $(q_p, 4, \circ)$ |
| 4 | $(q_d, 4, \triangleright)$ | $(q_d, 4, \triangleright)$ | $(q_p, 5, \circ)$ |
| 5 | $(q_d, 5, \triangleright)$ | $(q_d, 5, \triangleright)$ | $(q_p, 6, \circ)$ |
| 6 | $(q_d, 6, \triangleright)$ | $(q_d, 6, \triangleright)$ | $(q_p, 7, \circ)$ |
| 7 | $(q_d, 7, \triangleright)$ | $(q_d, 7, \triangleright)$ | $(q_p, 8, \circ)$ |
| 8 | $(q_d, 8, \triangleright)$ | $(q_d, 8, \triangleright)$ | $(q_p, 9, \circ)$ |
| 9 | $(q_d, 9, \triangleright)$ | $(q_d, 9, \triangleright)$ | $(q_s, 0, \triangleleft)$ |
| $B$ | $(q_p, 1, \circ)$ | $(q_s, B, \triangleleft)$ | $(q_p, 1, \circ)$ |

Quadro 4 – Imagens da função de transição da máquina do Exemplo 3.1.5. Fonte: elaborado pelo autor.

Mas nossa definição ainda possui um problema: o domínio sempre é $\Sigma^*$, ou seja, precisamos definir uma saída para toda sentença do alfabeto de entrada e não apenas para as que estão no formato de entrada. Isso pode ser visto no exemplo da função sucessora para números naturais simbolizados na base dez (Exemplo 3.1.5), que contorna o problema definindo a sua única entrada "fora do formato de entrada", $\varepsilon$, como outra forma de simbolizar o 0.

Uma possível solução para este problema seria não originar uma configuração final quando a entrada não estiver no formato de entrada, ou seja, para toda entrada que não estivesse no formato de entrada, $f(x)$ seria indefinida. Mas, assim, perderíamos o significado de $f(x)$ ser indefinida, o qual vamos reservar apenas para as entradas em que a máquina não decide[7].

Para resolver este problema vamos utilizar um conceito da área das Linguagens de Programação, mais especificamente da área de Construção de Compiladores, a de um analisador. Vamos refinar a nossa definição de função computável para restringir o domínio apenas para as entradas que estão no formato de entrada, adicionando uma componente, uma máquina de Turing para a decisão de problemas, que decide se uma entrada para a função está ou não nesse formato:

**Definição 3.1.10.** Uma máquina de Turing determinística de uma única fita *para funções* é definida assim como a da Definição 3.1.6, mas com uma componente adicional, a máquina de Turing $D = (Q_D, \Sigma, \Gamma_D, \delta_D, q_{0D}, B_D, A_D)$:

$$M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B, D)$$

[7] Como Hopcroft e Ullman que utilizam esta mesma semântica. Ainda assim poderíamos dizer que a máquina não decide para os dois casos, porém em um, porque a entrada não está no formato de entrada, e no outro, porque a configuração inicial não é e nem origina uma configuração final, ou seja, temos uma distinção clara entre eles.

tal que $D$ decide uma lingugaem.

**Definição 3.1.11.** A função computada por uma máquina de Turing determinística de uma única fita *para funções* $M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B, D)$ é a função parcial $f : L_D \to \Upsilon^*$ tal que $L_D$ é a linguagem decidida por $D$ e para toda entrada $x \in L_D$, $f(x)$ é igual a $y$ se e somente se a configuração inicial de $M$ para $x$ é ou origina uma configuração final de $M$ com o conteúdo de saída da fita igual a $y$, caso contrário, $f(x)$ é indefinida.

**Exemplo 3.1.6.** Seja $n \in \mathbb{N}$, $\Sigma_M = \{0, 1\}$ e $\langle n \rangle \in \Sigma_M^*$ a simbolização de $n$ na base dois[8]. Uma máquina de Turing determinística de uma única fita $M$ que computa a função $S : \Sigma_M^+ \to \Sigma_M^*$ tal que para todo $n \in \mathbb{N}$, $S(\langle n \rangle) = \langle (n+1) \rangle$:

$$M = (\{q_0, q_d, q_s, q_p\}, \Sigma_M, \Sigma_M \cup \{B\}, \Sigma_M, \delta, q_0, B, D)$$

tal que $D = (\{q_0, q_a, q_r\}, \Sigma_M, \Sigma_M \cup \{B\}, \delta_D, q_0, B)$, $q_a$ representa "aceita", $q_r$ representa "rejeita", $\delta_D(q, s)$ é definida pelo Quadro 5, $q_d$ representa "direita", $q_s$ representa "soma", $q_p$ representa "para" e $\delta(q, s)$ é definida pelo Quadro 6.

| $q$ \ $s$ | 0 | 1 | $B$ |
|---|---|---|---|
| $q_0$ | $(q_a, 0, \circ)$ | $(q_a, 1, \circ)$ | $(q_d, B, \circ)$ |

Quadro 5 – Imagens da função de transição da máquina $D$ do Exemplo 3.1.6. Fonte: elaborado pelo autor.

| $q$ \ $s$ | 0 | 1 | $B$ |
|---|---|---|---|
| $q_0$ | $(q_d, 0, \triangleright)$ | $(q_d, 1, \triangleright)$ | $(q_p, 1, \circ)$ |
| $q_d$ | $(q_d, 0, \triangleright)$ | $(q_d, 1, \triangleright)$ | $(q_s, B, \triangleleft)$ |
| $q_s$ | $(q_p, 1, \circ)$ | $(q_s, 0, \triangleleft)$ | $(q_p, 1, \circ)$ |

Quadro 6 – Imagens da função de transição da máquina $M$ do Exemplo 3.1.6. Fonte: elaborado pelo autor.

**Definição 3.1.12.** O **tempo de execução** de uma máquina de Turing determinística de uma única fita *para funções* $M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B, D)$ para uma entrada $w \in L_D$ tal que $L_D$ é a linguagem decidida por $D$ é o máximo entre o tempo de execução, conforme a Definição 2.1.13, de $M$ para $w$ e o tempo de execução de $D$ para $w$.

Precisamos redefinir o tempo de execução para uma entrada, pois, para que a função computada esteja definida para esta, necessitamos de duas coisas: que esta esteja no formato de entrada, ou seja, que a configuração inicial do analisador para esta origine uma

[8] Neste exemplo, pelo tamanho do quadro de $\delta$ em relação ao do Exemplo 3.1.5, também da função sucessora, porém na base dez, fica claro o porque de a unidade básica da informação e dos computadores ser um digito binário e não um decimal.

configuração de aceitação, e que a configuração inicial para esta origine uma configuração final. Por isso, definimos o tempo de execução para esta como o máximo entre o tempo para que estas duas coisas, que podem iniciar ao mesmo tempo, aconteçam, ou seja, sejam verificadas. Por exemplo, na Figura 3, da computação de uma imagem da função computada pela máquina do Exemplo 3.1.6, utilizando a mesma representação de configurações da Figura 1, o tempo para o analisador aceitar a entrada é apenas um e o tempo para a máquina originar uma configuração final a partir da configuração inicial para a entrada é oito, portanto o tempo de execução para a entrada é, o máximo entre estes tempos, oito.

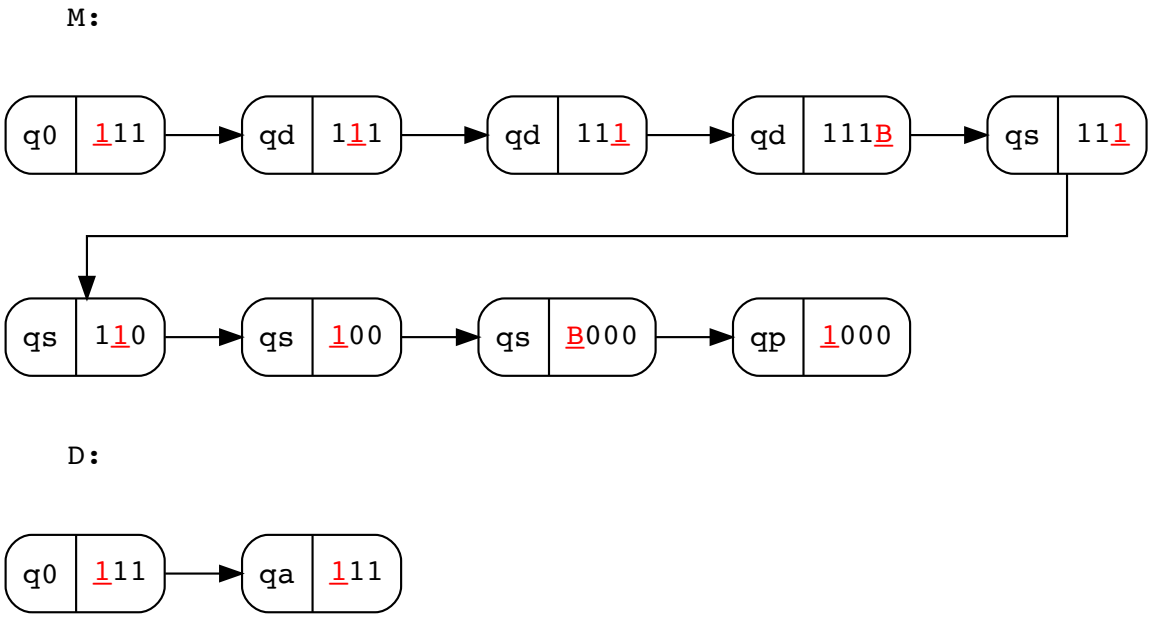


Figura 3 – A computação da imagem da função computada pela máquina do Exemplo 3.1.6 para a entrada 111. Fonte: elaborada pelo autor.

Outro conceito que também vem da área das Linguagens de Programação e que vamos utilizar em nossa proposta é o de funções variádicas, que diferentemente de funções $k$-árias, aceitam um número não arbitrário de argumentos (STROUSTRUP, 2013). É isso que Hopcroft e Ullman permitem quando computam funções diferentes (com um número de argumentos diferentes) com uma mesma máquina de Turing determinística de uma única fita e um conceito que usamos implicitamente no Exemplo 2.3.1, sobre as somas dos inteiros de uma sequência:

**Definição 3.1.13.** O **conjunto sequência**[9] $C^{\#}$ de um conjunto $C$ é a união de todos os produtos cartesianos de $C$:

$$C^{\#} = \bigcup_{k \in \mathbb{N}} C^k$$

(BURR, 2016)

[9] Não há um nome para este tipo de conjunto, escolhemos o nome “conjunto sequência” pois este é o conjunto de subsequências de uma sequência, ou seja, de sequências, e, porque este nome soa como “conjunto potência”.

**Definição 3.1.14.** Uma **função variádica** é qualquer função em que o domínio é um conjunto sequência.

**Definição 3.1.15.** Uma máquina de Turing determinística de uma única fita *para funções variádicas* é definida assim como a da Definição 3.1.10, mas com uma componente adicional, o símbolo separador, #:

$$M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B, \#, D)$$

tal que $\# \in (\Gamma - \Sigma)$.

**Definição 3.1.16.** A **configuração inicial** de uma máquina de Turing determinística de uma única fita *para funções variádicas* $M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B, \#, D)$ para as entradas $w_1, w_2, \ldots, w_n \in \Sigma^*$, $n \in \mathbb{N}$, é a configuração inicial de $M$, conforme a Definição 2.1.5, para $w_1 \# w_2 \# \ldots \# w_n$.

**Definição 3.1.17.** A função computada por uma máquina de Turing determinística de uma única fita *para funções variádicas* $M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B, \#, D)$ é a função variádica parcial $f : L_D^{\#} \to \Upsilon^*$ tal que $L_D$ é a linguagem decidida por $D$ e para todas as entradas $x_1, x_2, \ldots, x_n \in L_D$, $n \in \mathbb{N}$, $f(x_1, x_2, \ldots, x_n)$ é igual a $y$ se e somente se a configuração inicial de $M$ para $x_1, x_2, \ldots, x_n$ é ou origina uma configuração final de $M$ com o conteúdo de saída da fita igual a $y$, caso contrário, $f(x)$ é indefinida.

**Definição 3.1.18.** O **tempo de execução** de uma máquina de Turing determinística de uma única fita *para funções variádicas* $M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B, \#, D)$ para as entradas $w_1, w_2, \ldots, w_n \in L_D^{\#}$ tal que $L_D$ é a linguagem decidida por $D$ é o máximo entre o tempo de execução, conforme a Definição 2.1.13, de $M$ para $w_1 \# w_2 \# \ldots \# w_n$ e os tempos de execução de $D$ para as entradas $w_1, w_2, \ldots, w_n$.

## 3.2 COMPUTAÇÃO NÃO DETERMINÍSTICA DE FUNÇÕES

Uma configuração inicial de uma máquina de Turing não determinística pode originar mais do que uma configuração final, diferentemente de uma configuração inicial de uma máquina de Turing determinística, que pode originar no máximo uma configuração final.

Além disso, há dois casos em que conseguimos decidir sobre uma instância de um problema de decisão com uma máquina de Turing não determinística, quando a resposta é "sim" e a árvore de computação para uma entrada que simboliza esta instância possui pelo menos uma folha que é uma configuração de aceitação e quando a resposta é "não" e a árvore de comptuação para uma entrada que simboliza esta instância só possui folhas que são configurações de rejeição, diferentemente do único caso da máquina de Turing determinística quando a configuração inicial para uma entrada que simboliza esta instância origina uma configuração final.

Por isso, para adaptarmos o modelo da máquina de Turing não determinística para computar uma função, não basta apenas adaptarmos o método de obter uma solução de uma configuração final, como foi feito, na subseção anterior, para computar uma função com uma máquina de Turing determinística de uma única fita. Precisamos de uma estratégia que obtém uma única solução das folhas de uma árvore de computação, ou seja, de diversas configurações, das quais pelo menos uma é uma configuração final.

Mas este tópico não é muito explorado e as definições que se tem para uma função computável por uma máquina de Turing não determinística são raras. Há até mesmo autores que sugerem que o modelo da máquina de Turing não determinística não é adequado para a computação de funções (DAVIS; SIGAL; WEYUKER, 1994, p. 161).

Dos dois livros que foram utilizados como inspiração para a seção anterior, um, de Sipser, não dá uma definição, e o outro, de Hopcroft e Ulmman, dá uma definição informal e apenas ilustrativa para mostrar que esta não é uma tarefa trivial. Nesta subseção, vamos apresentar esta e outras definições juntamente com seus problemas. Note que estas não serão dadas em ordem cronológica, mas em uma ordem que proporciona uma análise melhor de suas semelhanças e diferenças.

### 3.2.1 HOPCROFT E ULLMAN (1979)

Hopcroft e Ullman mencionam a computação de funções com máquinas de Turing não determinísticas quando apresentam os Axiomas de Blum (HOPCROFT; ULLMAN, 1979; BLUM, 1967). Estes são axiomas que têm como objetivo a formalização das medidas de complexidade, de forma independente de um modelo de computação, desde que este compute uma função. A motivação de Hopcroft e Ullman é, portanto, satisfazer com as medidas de complexidade da máquina de Turing não determinística os Axiomas de Blum, que, por não compreenderem muitas medidas (FORTNOW, 2004), não serão detalhados aqui.

Eles apenas dão uma definição informal, mas, para facilitar a comparação com as outras definições dadas nesta subseção, vamos extrapolá-la para uma definição formal. As diferenças e lacunas preenchidas nesta formalização são descritas em suas notas de rodapé.

**Definição 3.2.1.** A função computada por uma máquina de Turing não determinística $M = (Q, \{0,1\}, \Gamma, \delta, q_0, B, A)$ é a função parcial $f : \mathbb{N} \to \mathbb{N}$ tal que para todo $x \in \mathbb{N}$, $f(x)$ é igual a $y \in \mathbb{N}$ se e somente se a configuração inicial de $M$ para $0^x$ é ou origina alguma configuração final de $M$ com o conteúdo da fita igual a $0^y$ e não origina nenhuma outra configuração final de $M$ com o conteúdo da fita diferente[10] de $0^y$, caso contrário,

[10] Eles não definem o que acontece quando se origina alguma configuração final contendo algum 1 em seu conteúdo da fita, apenas mencionam que não se pode originar alguma configuração final com o conteúdo da fita igual a $0^k$, $k \in \mathbb{N}$ e diferente de $y$.

$f(n)$ é indefinida[11]. (HOPCROFT; ULLMAN, 1979)

Esta definição possui dois problemas. O primeiro é que ela não permite adaptar diretamente todos os problemas de decisão para funções, assim como fizemos no Exemplo 3.1.4, já que conseguimos decidir sobre uma entrada de, e com uma máquina de Turing não determinística mesmo quando sua configuração inicial origina configurações finais de aceitação e de rejeição, que, pela adaptação direta, possuem conteúdos da fita diferentes.

Para esta definição, essa adaptação direta de problemas de decisão para funções só funciona nos casos de rejeição, já que a configuração inicial é ou apenas origina configurações que são ou originam configurações de rejeição, ou nos casos de aceitação em que a configuração inicial quando origina alguma configuração final, origina uma configuração de aceitação, o que não engloba todas as possíveis árvores de computação. Além disso, teríamos uma saída (0) errada para a função quando a configuração inicial originasse alguma configuração de rejeição e originasse também alguma configuração que não originasse nenhuma configuração final.

O segundo problema, apontado pelos próprios autores e mais crítico, é que, com ela, definimos saídas para a função que não conseguimos obter (computar), já que, devido ao Problema da Parada (Teorema 2.3), nos casos em que a configuração inicial não é e nem origina nenhuma configuração final ou origina alguma outra configuração que não origina nenhuma configuração final, a restrição "não origina nenhuma outra configuração final de $M$ com o conteúdo da fita diferente de $0^y$" não é verificável (computável). Isto é provado quando os autores mostram que com esta definição não é possível satisfazer o segundo axioma de Blum e ressaltam que, para isso, precisamos de uma definição inteligente de como computar uma função com uma máquina de Turing não-determinística.

### 3.2.2 LEWIS E PAPADIMITRIOU (1998) E RICH (2008)

Lewis e Papadimitriou (1998) e Rich (2008) dão, de forma independente, definições muito parecidas com a definição de Hopcroft e Ullman. Mas estes, restrigem as funções computáveis às máquinas de Turing não determinísticas em que todas as entradas possuem uma árvore de computação em que todas as folhas são configurações finais:

**Definição 3.2.2.** A função computada por uma máquina de Turing não determinística $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ é a função $f : \Sigma^* \to \Gamma^*$ se e somente se para toda entrada $x \in \Sigma^*$ de $M$, existe uma árvore de computação de $M$ para $x$ tal que todas as folhas são configurações finais de $M$ com o mesmo conteúdo da fita $f(x)$. (LEWIS; PAPADIMITRIOU, 1998; RICH, 2008)

---

[11] Eles não definem o que acontece quando não se origina nenhuma configuração final, mas estamos assumindo que, assim como fazem na definição de funções computadas por uma máquinas de Turing determinísticas de uma única fita (Definição 3.1.5), deixam a saída indefinida.

Por causa desta restrição, esta definição não possui o problema mais crítico da definição de Hopcroft e Ullman. Já que se todas as folhas de todas as árvores de computação de uma máquina de Turing não determinística são configurações finais, então todas as suas configurações iniciais são ou originam apenas configurações que são ou originam configurações finais.

Mas, por causa desta restrição, ela não permite adaptar problemas indecidíveis para funções. E restringe ainda mais a adaptação direta de problemas decidíveis para funções, já que conseguimos decidir sobre uma entrada se sua árvore de computação possui apenas uma folha que é uma configuração final, caso esta seja uma configuração de aceitação.

Lewis e Papadimitriou destacam as dificuldades associadas com a computação de funções com máquinas de Turing não determinísticas e dizem que precisamos "que todas as folhas sejam configurações finais com o mesmo conteúdo da fita $y$", pois, senão, não teríamos como escolher a saída correta para a função. O que vai contra a obtenção de uma solução na decisão de problemas com máquinas de Turing não determinísticas, que decide também quando nem todas as folhas são configurações finais e/ou nem todas as folhas são só de aceitação ou só de rejeição, que, pela adaptação direta, possuem conteúdos da fita diferentes.

### 3.2.3 GOLDREICH (2008)

Goldreich resolve o problema, levantado por Lewis e Papadimitriou, de escolher a saída correta para a função, permitindo que a máquina origine configurações com o conteúdo da fita igual a saída da função ou a um símbolo especial $\perp$:

**Definição 3.2.3.** A função computada por uma máquina de Turing não determinística $M = (Q, \{0,1\}, \Gamma, \delta, q_0, B, A)$ é a função $f : \{0,1\}^* \to \{0,1\}^*$ se e somente se para toda entrada $x \in \{0,1\}^*$, existe uma árvore de computação de $M$ para $x$ tal que todas folhas são configurações finais de $M$ com o conteúdo da fita igual a $\perp$ ou $f(x) \in \{0,1\}^*$. (GOLDREICH, 2008)

Esta definição possui a mesma restrição de Lewis e Papadimitriou e Rich, de só definir funções para máquinas em que todas as entradas possuem uma árvore de computação em que todas as folhas são configurações finais, e, portanto, assim como a definição de Lewis e Papadimitriou e Rich, embora não possua o problema mais crítico da definição de Hopcroft e Ullman, impede a adaptação de problemas indecidíveis para funções.

O símbolo especial $\perp$ é utilizado por Goldreich na adaptação direta de problemas de decisão para funções, como o conteúdo da fita das configurações de rejeição, o que permite, diferentemente das definições de Hopcroft e Ullman e Lewis e Papadimitriou e Rich, a adaptação dos casos em que a configuração inicial origina apenas configurações finais em que pelo menos uma é de aceitação e pelo menos uma é de rejeição.

Porém, diferentemente das definições de Hopcroft e Ullman e Lewis e Papadimitriou e Rich, esta definição não permite adaptar os casos de rejeição, já que, nestes, a configuração inicial é ou origina apenas configurações que são ou originam configurações de rejeição, ou seja, com o conteúdo da fita $\perp$, não atendendo a restrição "$f(x) \in \{0,1\}^*$".

### 3.2.4 KRENTEL (1986) E ROYER (2015)

Krentel, motivado em estudar a complexidade de problemas de otimização, em que se procura um valor máximo ou mínimo, define uma variação da máquina de Turing não determinística, denominada máquina de Turing métrica, que computa o máximo (ou minímo) entre os conteúdos da fita de suas folhas e uma classe de complexidade, denominada *OptP*, em que estão todos os problemas computados por sua variação em tempo polinomial.

Porém, restringe, assim como Lewis e Papadimitriou e Rich e Goldreich, suas funções computáveis às máquinas em que todas as entradas possuem uma árvore de computação em que todas as folhas são configurações finais e, assim como Lewis e Papadimitriou e Rich e Goldreich, impede a adaptação de problemas indecidíveis para funções. Por isso, neste trabalho, vamos dar apenas a definição de Royer, que, motivado em satisfazer, assim como Hopcroft e Ullman, os Axiomas de Blum com as medidas de complexidade da máquina de Turing não determinística, dá uma definição muito similar a de Krentel:

**Definição 3.2.4.** A função computada por uma máquina não determinística $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ é a função parcial $f : \Sigma^* \to \Gamma^*$ tal que para toda entrada $x \in \Sigma^*$, $f(x)$ é igual a $y$ se e somente se existe uma árvore de computação de $M$ para $x$ tal que todas as folhas são configurações finais de $M$ e $y$ é máximo numérico, ou léxico quando não simbolizam números, entre seus conteúdos da fita, caso contrário, $f(x)$ é indefinida. (ROYER, 2015)

Esta definição possui uma restrição, de só definir a saída da função para as entradas que possuem uma árvore de computação em que todas as folhas são configurações finais, similar a restrição de Lewis e Papadimitriou e Rich, Goldreich e Krentel, de só definir funções para máquinas em que *todas* as entradas possuem uma árvore de computação em que todas as folhas são configurações finais. Porém, diferentemente da segunda, a primeira restrição, de Royer, não impede a adaptação de problemas indecidíveis.

Há apenas outras duas diferenças entre a definição de Royer e a de Krentel. A primeira, é que a de Royer permite tratar funções simbólicas definindo também o máximo como o lexicamente maior. A segunda, é que a de Krentel permite tratar problemas em que se procura um valor mínimo, os quais não são tratados diretamente na de Royer[12].

---

[12] Que requer que se inverta os conteúdos de saída da fita das folhas. Por exemplo, nas funções numéricas, a máquina, em vez de originar configurações finais com o conteúdo de saída igual a $y$, teria que originar configurações finais com o conteúdo de saída igual $\frac{1}{y}$.

Diferentemente das definições de Hopcroft e Ullman, Lewis e Papadimitriou e Rich e Goldreich, a definição de Krentel e a de Royer permitem a adaptação direta de problemas decidíveis nos casos em que a configuração inicial é ou apenas origina configurações que são ou originam configurações finais, já que, nestes casos, a saída da função característica corresponde ao máximo entre os conteúdos da fita.

Porém, nenhuma delas permite a adaptação direta dos casos em que a configuração inicial origina alguma configuração de aceitação e alguma configuração que não origina nenhuma configuração final, já que, nestes casos, nem a restrição de Lewis e Papadimitriou e Rich, Goldreich e Krentel nem a restrição similar de Royer são atendidas.

Outro problema, este mais crítico, das duas definições é que ambas não especificam como a obtenção da saída da função pode ser feita, quando é, em tempo polinomial, como obter o valor máximo (ou mínimo) entre um número, possivelmente, exponencial de valores em tempo polinomial. Royer diz que talvez tenha "exagerado na dose".

No final de seu trabalho, Krentel sugere como trabalho futuro o estudo de outros operadores associativos aplicados às folhas de alguma árvore de computação, além do máximo, minímo, e que vamos apresentar a seguir, soma.

### 3.2.5 VALIANT (1979)

Assim como Krentel, Valiant também define uma classe de complexidade, a classe de problemas de contagem $\#P$ e chama de máquinas de Turing contadoras as máquinas de Turing não determinísticas que computam o número de configurações de aceitação de suas árvores de computação. Para facilitar a sua comparação com as outras definições desta seção, vamos apresentá-la de maneira similar a de Royer.

**Definição 3.2.5.** A função computada por uma máquina não determinística $M = (Q, \Sigma, \Gamma, \delta, q_0, B, A)$ é a função parcial $f : \Sigma^* \to \mathbb{N}$ tal que para toda a entrada $x \in \Sigma^*$, $f(x)$ é igual a $y$ se e somente se existe uma árvore de computação de $M$ para $x$ tal que todas as folhas são de configurações finais de $M$ e $y$ é o número de configurações de aceitação, caso contrário, $f(x)$ é indefinida. (VALIANT, 1979)

Esta definição, diferentemente das definições de Hopcroft e Ullman, Lewis e Papadimitriou e Rich e Goldreich, e assim como as de Krentel e Royer, permite a adaptação direta de problemas decíveis para funções dos casos em que a configuração inicial é ou apenas origina configurações que são ou originam configurações finais. Porém, diferentemente das definições de Krentel e Royer, que definem funções computáveis que podem corresponder à função característica, as funções computáveis definidas por Valiant necessitam, para esta correspondência e adaptação, da composição com a função $f$ tal que para todo $n \in \mathbb{N}$, $f(n)$ é igual a 1 se e somente se $n \geq 1$ e 0 caso contrário.

E, assim como as definições de Krentel e Royer, esta definição não permite a adaptação direta dos casos em que a configuração inicial origina alguma configuração de aceitação

e alguma configuração que não origina nenhuma configuração final, por causa da mesma restrição de Royer, e similar à de Lewis e Papadimitriou e Rich, Goldreich e Krentel, de só definir a saída da função para as entradas que possuem uma árvore de computação em que todas as folhas são configurações finais.

Além disso, esta definição possui o mesmo problema crítico das definições de Krentel e Royer, de não especificar como a obtenção da saída da função pode ser feita, quando é, em tempo polinomial, como obter a soma de um número, possivelmente, exponencial de valores em tempo polinomial.

# 4 PROPOSTA

Nesta seção, proporemos e definiremos a máquina de Turing não determinística com combinadora para a computação de funções. Na primeira subseção, descreveremos informalmente nossa estratégia. Na segunda, definiremos formalmente, utilizando os conceitos e notações desenvolvidas nas seções anteriores, nossa variação e provaremos a sua equivalência com a máquina de Turing determinística de uma única fita. Na última, mostraremos, com alguns exemplos, sua versatilidade.

## 4.1 DESCRIÇÃO INFORMAL

Nossa proposta é baseada na estratégia de *divisão e conquista* (ilustrada, na próxima página, na Figura 4), que, embora em seu nome só possua duas ações, é divida em três etapas (CORMEN *et al.*, 2009)[1]:

1. Divisão: divide o problema em subproblemas.
2. Conquista: obtém a solução de cada subproblema de forma direta ou indireta (utiliza a estratégia de *divisão e conquista* para obter a solução do subproblema).
3. Combinação: combina as soluções dos subproblemas em uma única solução para o problema original.

E da associação desta a uma árvore de computação de uma máquina de Turing não determinística para a decisão de problemas.[2] Primeiro, associamos a etapa de divisão às suas ramificações (vértices com mais de um filho). Então, associamos a etapa de conquista direta aos seus caminhos em que o primeiro vértice é a raiz ou o filho de uma ramificação e o último vértice é uma folha e associamos a etapa de conquista indireta aos seus caminhos em que o primeiro vértice é a raiz ou o filho de uma ramificação e o último vértice é uma ramificação.

Embora estejamos associando conquistas a caminhos, também vamos dizer que os vértices desses caminhos estão associados a essas conquistas. Por essas associações, cada vértice da árvore está associado à exatamente uma conquista. Estas associações são ilustradas na Figura 5, na qual os vértices cercados por um “D” (“I”) são os associados às conquistas diretas (indiretas).

Agora, associamos cada conquista a um problema. Primeiro, associamos o problema original à conquista associada à raiz. Então, associamos cada conquista associada ao filho

---

[1] Cormen *et al.* classifica a conquista indireta como “recursiva”, mas estamos usando nomes alternativos para que não haja confusão com “linguagens recursivas” e “linguagens recursivamente enumeráveis”, que também são nomes dados, respectivamente, a linguagens decidíveis e semidecidíveis.

[2] O que estamos descrevendo pode ser caracterizado como um processo de *tree acumulation*, mais especificamente uma *upwards accumulation* (GIBBONS, 1991). Mas, por razões de simplicidade, descreveremos nossa proposta baseados na estratégia de *divisão e conquista*.

de uma ramificação ao subproblema do problema associado à conquista associada a esta ramificação.

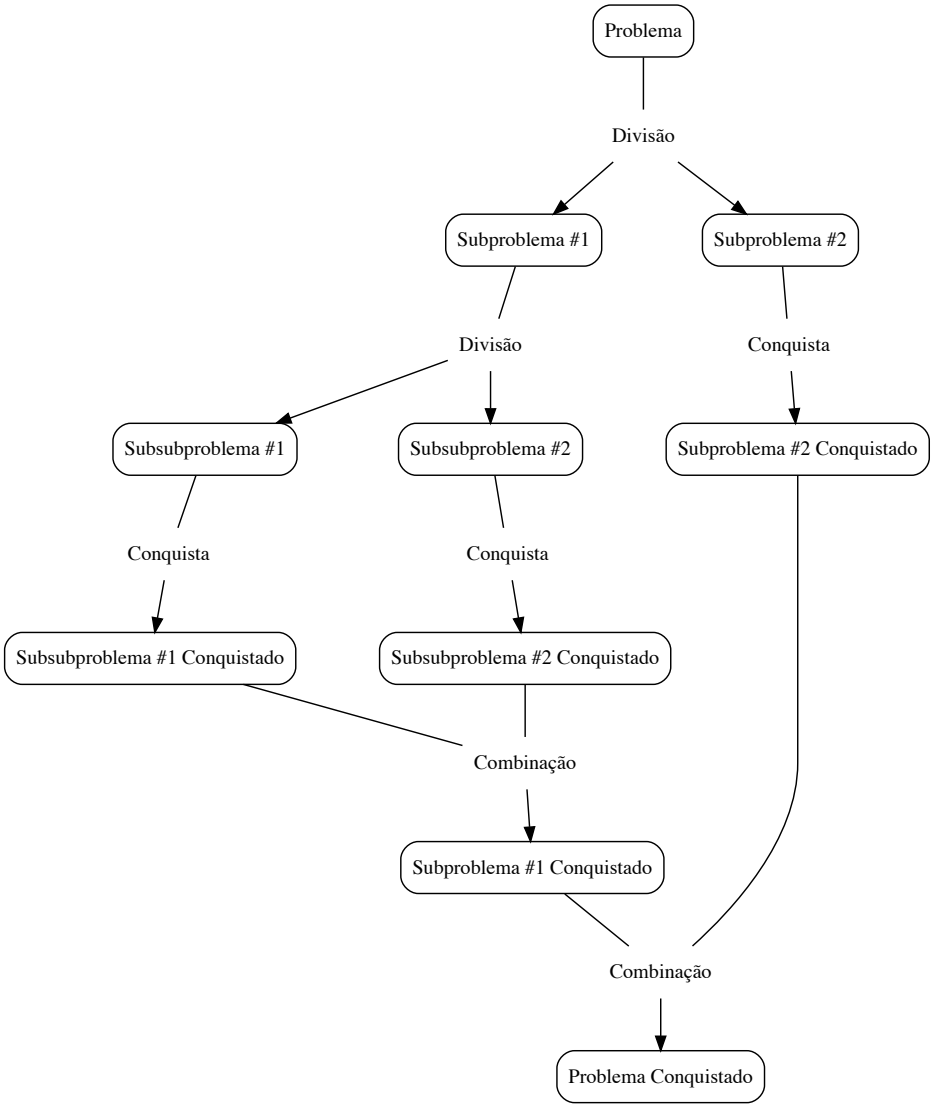


Figura 4 – A estratégia de *divisão e conquista*. Fonte: elaborada pelo autor.

Finalmente, para completar nossa associação, descrevemos como obter a solução destes problemas a partir destas conquistas, mais precisamente, como obter uma solução a partir de uma conquista direta e como combinar as soluções dos subproblemas de um problema para obter uma solução a partir de uma conquista indireta.

Para obter uma solução a partir de uma conquista direta, tratamos seu caminho como uma computação de uma máquina de Turing determinística e obtemos a sua solução com base no estado, como descrevemos na Subseção 3.1, da configuração de seu último vértice, originado a partir da configuração de seu primeiro vértice. A solução é "sim" se e somente se o último vértice deste caminho possui uma configuração de aceitação e "não" se e somente se possui uma configuração de rejeição. Nos casos em que a configuração deste último vértice não é uma configuração final, vamos dizer que a solução desta conquista direta é "pendente".

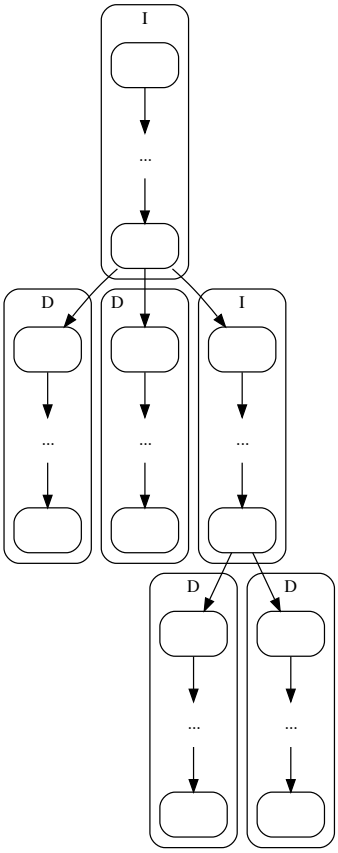


Figura 5 – A associação das conquistas da estratégia de *divisão e conquista* aos vértices de uma árvore de computação. Fonte: elaborada pelo autor.

Para combinar as soluções dos subproblemas de um problema para obter uma solução a partir de uma conquista indireta, seguimos as definições de aceitação (Definição 2.1.9) e de rejeição (Definição 2.3.4) da máquina de Turing não determinística. A solução é "sim" se e somente se pelo menos uma das soluções dos seus subproblemas é "sim", "não" se e somente se todas as soluções dos seus subproblemas são "não" e "pendente" caso contrário. A combinação das soluções de dois subproblemas é descrita no Quadro 7.

| *Solução 1* | *Solução 2* | *Combinação* |
|---|---|---|
| sim | sim | sim |
| sim | pendente | sim |
| sim | não | sim |
| pendente | sim | sim |
| pendente | pendente | pendente |
| pendente | não | pendente |
| não | sim | sim |
| não | pendente | pendente |
| não | não | não |

Quadro 7 – Combinação das soluções de dois subproblemas. Fonte: elaborado pelo autor.

Nossa proposta é adaptar a obtenção de uma solução a partir de uma conquista direta como adaptamos o modelo da máquina de Turing determinística para a computação de funções, obtendo a sua solução, não a partir do estado, mas a partir do conteúdo de saída da fita da configuração de seu último vértice. O que falta para completar nossa adaptação, mantendo nossa associação, é adaptar como combinar as soluções dos subproblemas de um problema para obter uma solução a partir de uma conquista indireta.

Poderíamos fazer como Valiant, Krentel e Royer, que definiram operadores associativos (soma, mínimo ou máximo) que realizam estas combinações. Mas, para darmos uma definição mais geral, seguindo a sugestão de Krentel, de estudar outros operadores associativos, vamos propor uma variação da máquina de Turing não determinística com uma componente adicional, uma máquina de Turing para funções, a qual vamos chamar de combinadora, que será responsável por realizar estas combinações. Devido ao nome que vamos dar a esta componente adicional, vamos chamar nossa variação de máquina de Turing não determinística com combinadora.

## 4.2 DEFINIÇÃO FORMAL

Antes de definirmos a nossa variação, vamos modificar a definição da máquina de Turing não determinística (Definição 2.3.1) como, e pelos mesmo motivos, modificamos, na Subseção 3.1, a definição da máquina de Turing determinística de uma única fita:

**Definição 4.2.1.** Uma máquina de Turing não determinística *para funções* é definida assim como a da Definição 2.3.1, mas sem um conjunto de aceitação e com duas componentes adicionais, o conjunto do alfabeto de saída $\Upsilon$ e uma máquina de Turing $D = (Q_D, \Sigma, \Gamma_D, \delta_D, q_{0D}, B_D, A_D)$:

$$M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B, D)$$

tal que $\Upsilon \subseteq \Gamma$ e $D$ decide uma linguagem.

Feitas essas modificações, podemos definir nossa variação. Não vamos alterar componentes da definição da máquina de Turing não determinística, assim como fizemos com a definição da máquina de Turing determinística de uma única fita na definição da máquina de Turing determinística multifita (Definição 2.2.1) e na da máquina de Turing não determinística (Definição 2.3.1), mas vamos, como acabamos de fazer e como fizemos na Seção 3, sobre a computação de funções com máquinas de Turing, adicionar componentes à definição; porém, para encurtar nossas definições, vamos, diferentemente destes casos, definir nossa variação como um tripla[3]:

*Convenção.* A partir de agora, para facilitar a leitura e não poluir o texto, quando estivermos falando de uma máquina de Turing vamos nos referenciar às suas componentes com os símbolos utilizados para as tais nas definições para as máquinas de Turing subscritos

[3] Senão, nossa variação seria uma 10-upla

do símbolo utilizado para a máquina de Turing em questão. Por exemplo: seja $M$ uma máquina de Turing, $\Sigma_M$ é o alfabeto de entrada de $M$.

**Definição 4.2.2.** Uma **máquina de Turing não determinística com combinadora** $M$ é uma tripla:

$$M = (N, P, C)$$

tal que:

1. $N$ é uma máquina de Turing não determinística *para funções* (Definição 4.2.1),
2. $C$ é uma máquina de Turing determinística de uma única fita *para funções variádicas* (Definição 3.1.15) tal que $\Sigma_C$ é igual a $(\Upsilon_N \cup \Upsilon_C)$ e $D_C$ decide $\Sigma_C^*$,
3. $P \in (\Upsilon_C - \Upsilon_N)$ é o símbolo para "pendente",
4. $C$ computa a função parcial e simétrica *comb* tal que para todo $s_1$, $s_2$, ..., $s_n \in \Sigma_C^*$, $n > 1$ e igual a $|\delta_N(q, s)|$ para algum $q \in Q_N$ e algum $s \in \Gamma_N$, $comb(s_1, s_2, \ldots, s_n)$ é definida e igual a:

   a) $P$ se para todo $i \in [1, n] \cap \mathbb{N}$, $s_i = P$,

   b) $w \in (\Upsilon_C^* - \{P\})$ se para todo $i \in [1, n] \cap \mathbb{N}$, $s_i \neq P$,

   c) $comb(s_1, s_2, \ldots, s_{n-1}, P)$ se $comb(s_1, s_2, \ldots, s_{n-1}, P) \neq P$.

A etapa de divisão é definida pela máquina $N$ e a etapa de combinação pela máquina $C$, que tem como alfabeto de entrada a união dos símbolos do alfabeto de saída de $N$ com os símbolos de seu alfabeto de saída, pois pode ter como uma de suas entradas o conteúdo de saída da fita de uma configuração final de $N$ ou a saída de uma combinação, ou seja, uma imagem de *comb*, e, já que não garantimos que nenhuma destas estão em um formato definido, $D_C$ decide $\Sigma_C^*$. Denotamos uma solução pendente pelo símbolo $P$ e definimos que $P \in (\Upsilon_C - \Upsilon_N)$, pois, senão, uma configuração inicial de $N$ poderia originar apenas configurações que são ou originam configurações finais com o conteúdo de saída da fita $P$, ou seja, que não podem ser combinadas — como será explicado abaixo — e nem originar novas configurações.

A função *comb* computada por $C$ é parcial e só precisa estar definida para $n$ argumentos tal que $n$ é maior que um e igual ao número de configurações ($|\delta_N(q, s)|$) originadas por um movimento a partir de alguma configuração de $N$. Ela é simétrica, pois uma configuração de $N$ origina por um movimento um conjunto, e não uma sequência de configurações, ou seja, a ordem não importa. Se todos os seus argumentos são $P$, então ela deve estar definida como $P$, pois, senão, teríamos uma combinação mesmo quando não há soluções para combinar. Se todos os seus argumentos são diferentes de $P$, então ela deve estar definida como um valor diferente de $P$, pois, senão, teríamos todas as soluções e nenhuma combinação. Se ela está definida como $y$ para os argumentos $a_1$, $a_2$, ..., $a_n$, $P$, então ela também deve estar definida como $y$ para os argumentos $a_1$, $a_2$, ..., $a_n$, $a_i$, $a_i \neq P$, pois,

senão, necessitaríamos de todas as soluções para termos certeza de uma combinação, ou seja, que todas as configurações de $N$ fossem ou originassem configurações finais.

Definido como devem ser as combinações feitas por $C$, vamos definir como aplicá-las a uma árvore de computação de $N$ e definir o que é uma árvore de computação para nossa variação. Porém, ao modificar a definição da máquina de Turing não determinística, com a Definição 4.2.1, retiramos o conjunto de estados de aceitação e, com isso, perdemos a definição de uma árvore de computação[4] (Definição 2.3.3). Por isso, vamos utilizar uma definição intermediária, a de uma subárvore de computação:

**Definição 4.2.3.** Uma **subárvore de computação** de uma máquina de Turing não determinística *para funções* $M = (Q, \Sigma, \Gamma, \Upsilon, \delta, q_0, B, D)$ para alguma entrada $w \in \Sigma^*$ é uma árvore de configurações de $M$ tal que:

- a raiz da árvore é a configuração inicial de $M$ para $w$,
- se $c$ não é uma folha, então todo $c'$ tal que $c \vdash_M c'$ está na árvore,
- para todo caminho $c_1, c_2, \ldots, c_n$, $n \in [2, \infty) \cap \mathbb{N}$, a partir da raiz da árvore, $c_i \vdash_M c_{i+1}$ para todo $i \in [1, n-1] \cap \mathbb{N}$.

É em uma subárvore de $N$ que vamos aplicar as combinações feitas por $C$. Esta aplicação será feita por meio de anotações em seus vértices. Por isso, nesta seção, diferentemente das seções anteriores, começamos a falar que um vértice "possui" uma configuração, ao invés de, um vértice "é" uma configuração.

**Definição 4.2.4.** A **saída de um vértice** $v$, denotada por $s[v]$, que possui a configuração $c$ de uma subárvore de computação de uma máquina de Turing não determinística $N$ de uma máquina de Turing não determinística com combinadora $M = (N, P, C)$ tal que $C$ computa a função *comb* é igual a:

1. o conteúdo de saída da fita de $c$ se $v$ é uma folha e $c$ é uma configuração final de $N$,
2. $P$ se $v$ é uma folha e $c$ não é uma configuração final de $N$,
3. $comb(s[v_1], s[v_2], \ldots, s[v_n])$ se $v$ possui os filhos $v_1$, $v_2$, ..., $v_n$, $n \in [2, \infty) \cap \mathbb{N}$,
4. $s[v_1]$ se $v$ só possui um filho $v_1$.

A saída de um vértice é a solução da conquista associada a ele. Os Itens 1 e 2 definem como obter a solução de uma conquista direta baseada no último vértice de seu caminho. O Item 3 aplica a combinação feita por $C$ e define como obter a solução de uma conquista indireta, combinando as soluções dos subproblemas do problema associado a ela. E, o Item 4 define como levar essas soluções pela árvore (das folhas até a raiz).

Com estas duas definições, a de uma subárvore de computação e a de uma saída de um vértice, podemos definir o que é uma árvore de computação para nossa variação:

---

[4] E, por isso, perdemos também a definição do tempo de execução, que, assim como a definição de árvore de computação, só será feita para nossa variação.

**Definição 4.2.5.** Uma **árvore de computação** de uma máquina de Turing não determinística com combinadora $M = (N, P, C)$ para alguma entrada $w \in \Sigma_N^*$ é uma subárvore de computação de $N$ para $w$ tal que a saída da raiz é diferente de $P$.

E, com esta definição, podemos definir as funções computadas por nossa variação:

**Definição 4.2.6.** A **função computada** por uma máquina de Turing não determinística com combinadora $M = (N, P, C)$ é a função parcial $f : L_{D_N} \to \Upsilon_C^*$ tal que $L_{D_N}$ é a linguagem decidida pelo analisador de $N$, $D_N$, e para todo $x \in L_{D_N}$, $f(x)$ é igual a saída da raiz da árvore de computação de $M$ para $x$ se e somente se existe uma árvore de computação de $M$ para $x$, caso contrário, $f(x)$ é indefinida.

E também provar que estas podem ser computadas por máquinas de Turing determinísticas de uma única fita, ou seja, que nossa variação é equivalente a máquina de Turing determinística de uma única fita:

**Teorema 4.1.** *Se uma máquina de Turing não determinística com combinadora $M = (N, P, C)$ computa a função $f$, então podemos construir uma máquina de Turing determinística de uma única fita $S$ que também computa $f$.*

*Observação.* Ao invés de descrevermos como $S$ funciona, assim como fizemos nos teoremas de equivalência para máquinas de Turing multifita (Teorema 2.4) e para máquinas de Turing não determinísticas (Teorema 2.5), vamos dar um passo a passo de como computar uma função computada por uma máquina de Turing não determinística com combinadora, mostrando que este pode ser realizado por uma máquina de Turing determinística de uma única fita.

*Ideia da prova.* O passo a passo consiste em uma construção gradual da árvore de computação de $M$ para uma entrada. Esta construção ocorre como uma busca-em-largura. A cada novo vértice *originado*[5] anotamos sua saída e se anotamos um valor diferente de $P$, então adicionamos seu pai à fila de vértices a serem reanotados. Para cada vértice nesta fila reanotamos sua saída e se reanotamos um valor diferente de $P$, então também adicionamos seu pai a ela. Quando reanotamos a saída da raiz e esta é diferente de $P$, então a damos como a saída da função, senão continuamos a construção da árvore. Note que precisamos manter um histórico dos vértices originados para podermos reanotar suas saídas[6].

---

[5] Formalmente, a configuração do vértice é que é originada, mas para facilitar o entendimento da ideia da prova, que é um texto informal, estamos abstraindo esse detalhe.

[6] Na prova do Teorema 2.5, sobre a equivalência entre a máquina de Turing não determinística e a máquina de Turing determinística de uma única fita, também descrevemos uma busca-em-largura, porém nesta não precisávamos manter um histórico dos vértices originados, já que, nesta, estes deixam de ser úteis após darem origem a novos vértices.

*Prova.* Para obter $f(w)$ para uma entrada $w \in L_{D_N}$ tal que $L_{D_N}$ é a linguagem decidida por $D_N$:

1 Obtenha a configuração inicial de $N$ para $w$.

2 Se a configuração obtida no Passo 1 é uma configuração final, então dê o seu conteúdo de saída da fita como a saída da função e pare.

3 Comece a construção da árvore de computação para $w$ como um único vértice contendo a configuração obtida no Passo 1.

4 Obtenha todas as configurações originadas por um movimento a partir das configurações das folhas da árvore construída até o momento.

5 Para toda configuração obtida no Passo 4: adicione um vértice contendo-a como um filho do vértice que possui a configuração que a originou.

6 Para toda folha adicionada no Passo 5 que possua uma configuração que não seja final: anote a sua saída como $P$ (Item 2 da Definição 4.2.4).

7 Para toda folha adicionada no Passo 5 que possua uma configuração final: anote a sua saída como o conteúdo de saída da fita de sua configuração (Item 1 da Definição 4.2.4).

8 Para toda folha anotada no Passo 7: adicione seu pai à fila de vértices a serem reanotados.

9 Se não há vértices na fila de vértices a serem reanotados, então vá para o Passo 4.

10 Obtenha o primeiro vértice na fila de vértices a serem reanotados e remova-o da fila.

11 Se o vértice obtido no Passo 10 possui apenas um filho, então reanote a sua saída como a saída do filho (Item 4 da Definição 4.2.4) e vá para o Passo 13.

12 Reanote a saída do vértice obtido no Passo 10 como $comb(s_{v_1}, s_{v_2}, \ldots, s_{v_n})$ tal que $s_{v_1}$, $s_{v_2}$, ..., $s_{v_n}$ são as saídas dos seus filhos $v_1$, $v_2$, ..., $v_n$, $n \in \mathbb{N}$, e $comb$ é a função computada por $C$ (Item 3 da Definição 4.2.4).

13 Se o vértice obtido no Passo 10 possui um pai, então adicione seu pai à fila de vértices a serem reanotados e vá para o Passo 10.

14 Se a saída da raiz é igual a $P$, então vá para o Passo 4.

15 Dê a saída da função como a saída da raiz. □

Por fim, vamos definir o tempo de execução para nossa variação. Para isso, vamos utilizar uma definição intermediária, assim como utilizamos a definição de uma subárvore de computação para definir uma árvore de computação. Vamos, antes de definir o tempo de execução para uma entrada $w$, definir o tempo para obter a saída de um vértice de uma subárvore de computação. Neste, assumimos que já temos esta subárvore, o tempo para

computá-la só será considerado quando definirmos o tempo de execução, ou seja, este é o tempo para obter a saída de um vértice *a partir de* uma subárvore de computação:

**Definição 4.2.7.** O **tempo para obter a saída de um vértice** $s[v]$, denotado por $t[s[v]]$, de uma subárvore de computação de uma máquina de Turing não determinística com combinadora $M = (N, P, C)$ é igual a:

1. 0 se $v$ é uma folha,
2. o tempo de execução de $C$ para as entradas $s[v_1]$, $s[v_2]$, ..., $s[v_n]$ somado ao máximo entre $t[s[v_1]]$, $t[s[v_2]]$, ..., $t[s[v_n]]$ se $v$ possui os filhos $v_1$, $v_2$, ..., $v_n$, $n \in [2, \infty) \cap \mathbb{N}$,
3. $t[s[v_1]]$ se $v$ só possui um filho $v_1$.

O Item 1 define o tempo para obter a saída das folhas, que, uma vez que não são feitas combinações neste nível, é 0. O Item 2 define o tempo necessário para as combinações, que podem ser realizadas por diversas instâncias de $C$. O Item 3, assim como o Item 4 da definição da saída de um vértice (Definição 4.2.4), define como levar estes tempos pelos vértices da subárvore (das folhas até a raiz).

Esta definição não leva em consideração o tempo para originar[7] estes vértices. O tempo para originar um vértice só será considerado na definição seguinte, do tempo de execução, na qual vamos somar a ele o tempo para obter sua saída. Por exemplo, se uma subárvore não possui ramificações, ou seja, é como uma computação determinística, então, pela definição da saída de um vértice (Definição 4.2.4) a saída de sua raiz é a saída de sua única folha e, pela definição que acabamos de estabelecer, o tempo para obter sua saída é 0, ou seja, não iremos somar nada, assim como não somamos nada quando definimos como computar uma função com uma máquina de Turing determinística de uma única fita (Definição 3.1.6).

**Definição 4.2.8.** O **tempo de execução** de uma máquina de Turing não determinística $M = (N, P, C)$ para uma entrada $w \in L_{D_N}$ tal que $L_{D_N}$ é a linguagem decidida por $D_N$ é o máximo entre o tempo de execução de $N_D$ para $w$ e o mínimo do conjunto $\{ h[T]+t[s[r]]$ tal que $T$ é uma árvore de computação de $M$, $h[T]$ é a altura de $T$ e $r$ é a raiz de $T \}$.

O tempo para originar uma árvore de computação e obter a saída de sua raiz é o mínimo de um conjunto, pois, para cada novo conjunto de folhas da árvore de computação originado, novas combinações podem ser feitas, e, para originar novas folhas, não precisamos esperar por estas combinações, ou seja, originamos e combinamos ao mesmo tempo. Além deste tempo, de originar e obter a saída, no tempo de execução também levamos em consideração o tempo necessário para o analisador, da mesma maneira que fizemos na Definição 3.1.12.

---

[7] Formalmente, as configurações dos vértices que são originadas, mas, assim como na descrição da ideia da prova do Teorema 4.1, e pelos mesmos motivos, estamos abstraindo esse detalhe.

Retomando a analogia feita por Turing, descrita na Subseção 2.1, sobre máquinas de Turing determinísticas de uma única fita, podemos descrever nossa variação como infinitas pessoas manipulando infinitas fitas. Inicialmente, apenas uma pessoa manipula, porém, quando há mais de uma possível manipulação, ou seja, o não determinismo, esta chama novas pessoas para cada uma delas e assim por diante. Porém, diferentemente da decisão de problemas com máquina de Turing não determinísticas, em que a pessoa que chama somente observa se pelo menos uma das pessoas chamadas aceita ou se todas rejeitam, na computação de funções com a nossa variação, a pessoa que chama também combina as soluções encontradas.

## 4.3 EXEMPLOS

Para deixar mais clara a nossa proposta, vamos terminar esta seção com alguns exemplos de nossa variação. No primeiro, vamos mostrar como, com a nossa variação, computamos as funções características das linguagens (semi)decididas por máquinas de Turing não determinísticas:

**Exemplo 4.3.1.** Seja $N$ uma máquina de Turing não determinística para decisão de problemas e $N_F$ a adaptação, como a do Exemplo 3.1.4, de $N$ para uma máquina de Turing não determinística *para funções*. Uma máquina de Turing não determinística com combinadora $M$ que computa a função característica da linguagem (semi)decidida por $N$: $M = (N_F, P, C)$ tal que $C$ computa a função *comb* tal que para todo $s_1$, $s_2$, ..., $s_n \in \Sigma_C^*$, $n \in [2, \infty) \cap \mathbb{N}$, $comb(s_1, s_2, \ldots, s_n)$ é igual a 1 se e somente se $s_i = 1$ para algum $i \in [1, n] \cap \mathbb{N}$, 0 se e somente se $s_i = 0$ para todo $i \in [1, n] \cap \mathbb{N}$ e $P$ caso contrário.

No segundo, vamos mostrar como, com a nossa variação, computamos as funções definidas por Valiant:

**Exemplo 4.3.2.** Seja $n \in \mathbb{N}$, $\langle n \rangle$ a simbolização de $n$ na base dois, $N$ uma máquina de Turing não determinística para decisão de problemas e $N_F$ a adaptação, como a do Exemplo 3.1.4, de $N$ para uma máquina de Turing não determinística *para funções*. Uma máquina de Turing não determinística com combinadora $M$ que computa a função computável de $N$ definida por Valiant (Definição 3.2.5): $M = (N_F, P, C)$ tal que $C$ computa a função *comb* tal que para todo $\langle n_1 \rangle$, $\langle n_2 \rangle$, ..., $\langle n_m \rangle \in \Sigma_C^*$, $m \in [2, \infty) \cap \mathbb{N}$ e $n_i \in \mathbb{N}$ para todo $i \in [1, m] \cap \mathbb{N}$, $comb(\langle n_1 \rangle, \langle n_2 \rangle, \ldots, \langle n_m \rangle) = \langle (\sum_i^m n_i) \rangle$ e para todo $w_1$, $w_2$, ..., $w_m \in \Sigma_C^*$, $m \in [2, \infty] \cap \mathbb{N}$ e $w_i = P$ para algum $i \in [1, m] \cap \mathbb{N}$, $comb(w_1, w_2, \ldots, w_m) = P$.

Na Figura 6 é ilustrada uma árvore de computação para a máquina do exemplo acima, usando a máquina de Turing não determinística do Exemplo 2.3.1 como $N$. Cada vértice é representado por um quadro tal que a primeira linha é dividida em duas colunas, a primeira para o estado de sua configuração e a segunda para a sua saída na base dez, e a

segunda para o conteúdo da fita de sua configuração, destacando seu símbolo atual com a cor vermelha e uma sublinha.

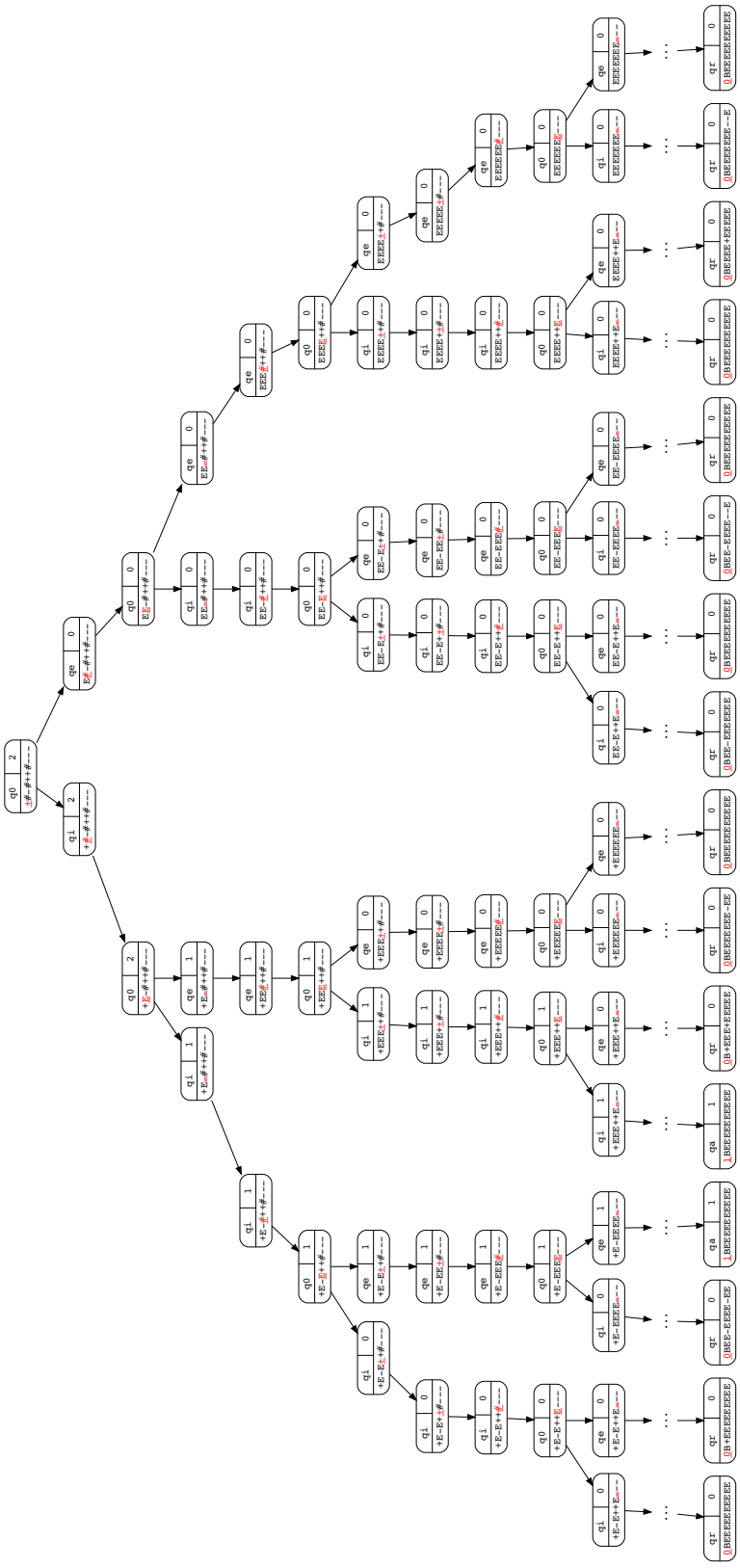

Figura 6 – Uma árvore de computação da máquina de Turing não determinística com combinadora do Exemplo 4.3.2 para a entrada +#−#++#−−− que simboliza a sequência {1, -1, 2, -3}. Fonte: elaborada pelo autor.

No terceiro, vamos mostrar como, com a nossa variação, computamos as funções definidas por Krentel e Royer:

**Exemplo 4.3.3.** Seja $n \in \mathbb{N}$, $\langle n \rangle$ a simbolização $n$ na base dois, $N$ uma máquina de Turing não determinística para decisão de problemas e $N_F$ a adaptação, como a do Exemplo 3.1.4, de $N$ para uma máquina de Turing não determinística *para funções*. Uma máquina de Turing não determinística com combinadora $M$ que computa a função computável de $N$ definida por Krentel e Royer (Definição 3.2.4): $M = (N_F, P, C)$ tal que $C$ computa a função *comb* tal que para todo $\langle n_1 \rangle$, $\langle n_2 \rangle$, ..., $\langle n_m \rangle \in \Sigma_C^*$, $m \in [2, \infty) \cap \mathbb{N}$, $n_i \in \mathbb{N}$ para todo $i \in [1, m] \cap \mathbb{N}$ e $n_j \geq n_i$, $j \in [1, m] \cap \mathbb{N}$, para todo $i \in [1, m] \cap \mathbb{N}$, $comb(\langle n_1 \rangle, \langle n_2 \rangle, \ldots, \langle n_m \rangle) = \langle n_j \rangle$ e para todo $w_1, w_2, \ldots, w_m \in \Sigma_C^*$, $m \in [2, \infty] \cap \mathbb{N}$ e $w_i = P$ para algum $i \in [1, m] \cap \mathbb{N}$, $comb(w_1, w_2, \ldots, w_m) = P$.

No quarto, vamos mostrar como, com a nossa variação, computamos as funções características dos complementos das linguagens decididas por máquinas de Turing não determinísticas. Note que as linguagens precisam ser decidíveis por causa do Problema da Parada (Teorema 2.3).

**Exemplo 4.3.4.** Seja $N$ uma máquina de Turing não determinística para decisão de problemas que decide uma linguagem e $N_F$ a adaptação de $N$ para uma máquina de Turing não determinística *para funções*, como a do Exemplo 3.1.4, porém, utilizando o 0 para as configurações de aceitação e o 1 para as configurações de rejeição. Uma máquina de Turing não determinística com combinadora $M$ que computa a função característica do complemento da linguagem decidida por $N$: $M = (N_F, P, C)$ tal que $C$ computa a função *comb* tal que para todo $s_1, s_2, \ldots, s_n \in \Sigma_C^*$, $n \in [2, \infty) \cap \mathbb{N}$, $comb(s_1, s_2, \ldots, s_n)$ é igual a 0 se e somente se $s_i = 0$ para algum $i \in [1, n] \cap \mathbb{N}$, 1 se e somente se $s_i = 1$ para todo $i \in [1, n] \cap \mathbb{N}$ e $P$ caso contrário.

No último, vamos mostrar como, com a nossa variação, computamos uma função que resolve o Problema de Deutsch-Jozsa. Este é um problema da Computação Quântica que pode ser enunciado como: dada uma função total $f : \{0,1\}^n \to \{0,1\}$, $f$ é constante, ou seja, igual a 0 para todas as entradas ou igual a 1 para todas as entradas, ou $f$ é balanceada, ou seja, igual a 0 para uma metade das entradas e igual a 1 para a outra? (POLLACHINI, 2018)

**Exemplo 4.3.5.** Seja $n \in \mathbb{N}$, $f$ uma função total $f : \{0,1\}^n \to \{0,1\}$ e $g : \{0,1\} \to \mathbb{N}$ tal que para todo $m \in \{0,1\}$, $g(m) = p$ tal que $p$ é igual ao número de imagens de $f$ que são iguais a $m$. Uma máquina de Turing não determinística com combinadora que computa $g$: $M = (N, P, C)$ tal que $N$ origina uma configuração final para cada imagem de $f$ com o conteúdo de saída da fita igual a 1 se e somente se esta é igual a $m$ e igual a 0 se e somente se esta é diferente de $m$ e $C$ é definida como a do Exemplo 4.3.2, de máquinas

de Turing não determinísticas com combinadora que computam as funções definidas por Valiant.

A função $f$ é constante se e somente se $g(0) = \langle n \rangle$ e é balanceada se e somente se $f(0) = \langle \frac{n}{2} \rangle$.

# 5 CONCLUSÃO

*Um trabalho nunca é terminado,*
*apenas abandonado.*

O objetivo geral deste trabalho, propor uma variação da máquina de Turing não determinística para a computação de funções, foi realizado na última seção. Na Introdução, Seção 1, definimos alguns objetivos específicos para que possamos classificar esta realização como satisfatória; foram eles: a revisão da computação de funções por máquinas de Turing determinísticas, realizada na Subseção 3.1, o apontamento do que falta nas definições existentes de funções computadas por máquinas de Turing não determinísticas, realizado na Subseção 3.2, a descrição informal da variação proposta e de sua inspiração, a estratégia de *divisão e conquista*, realizada na Subseção 4.1, a definição formal da variação proposta, provando a sua equivalência com a máquina de Turing determinística, realizada na Subseção 4.2 e provada no Teorema 4.1, a exemplificação da variação proposta e de sua versatilidade, realizada na Subseção 4.3, e a comparação da variação proposta com as definições existentes de funções computadas por máquinas de Turing não determinísticas, que iremos realizar na próxima subseção. Após essa comparação, iremos também sugerir alguns trabalhos futuros.

## 5.1 COMPARAÇÃO COM OUTROS TRABALHOS

Nossa definição, assim como as definições de Lewis e Papadimitriou e Rich, Goldreich, Krentel e Royer e Valiant, não possui, como mostra o Teorema 4.1, o problema crítico da definição de Hopcroft e Ullman. E, diferentemente das definições de Lewis e Papadimitriou e Rich, Goldreich e Krentel, e, assim como as definições de Hopcroft e Ullman, Royer e Valiant, nossa definição permite, como mostra o Exemplo 4.3.1, adaptar problemas indecidíveis para funções.

Também como mostra o Exemplo 4.3.1, nossa definição permite, diferentemente das definições de Hopcroft e Ullman, Lewis e Papadimitriou e Rich e Goldreich e assim como as definições de Krentel e Royer e Valiant, a adaptação direta dos casos em que a configuração inicial é ou apenas origina configurações que são ou originam configurações finais. Mas, diferentemente das definições de Krentel e Royer e Valiant, nossa definição permite também a adaptação direta dos casos em que a configuração inicial origina alguma configuração de aceitação e alguma configuração que não origina nenhuma configuração final. Ou seja, diferentemente de todas as definições apresentadas na Subseção 3.2, nossa definição permite a adaptação direta de todos os problemas de decisão para funções.

Um problema crítico das definições de Krentel e Royer e Valiant é que nenhuma delas especifica como a obtenção da saída pode ser feita, quando é, em tempo polinomial, o que fizemos, na Definição 4.2.8, para nossa variação. Além disso, por ser mais geral e

não estar engessada em um único operador associativo, diferentemente dessas definições, nossa definição mostra, nos Exemplo 4.3.3 e 4.3.2, como a obtenção da saída pode ser feita, quando é, em tempo polinomial para os operadores associativos utilizados por elas.

### 5.1.1 CONTRIBUIÇÕES SECUNDÁRIAS

Por não estarem diretamente associadas ao objetivo geral e a nenhum dos objetivos específicos deste trabalho, contribuições feitas na Seção 2, sobre máquinas de Turing, acabaram sendo ofuscadas. Gostaríamos, aqui, de destacar a definição formal de um movimento de uma máquina de Turing determinística multifita (Definição 2.2.3), a definição formal de uma árvore de computação de uma máquina de Turing não determinística (Definição 2.3.3) e a definição formal da função de transição de uma máquina de Turing não determinística que decide um problema não trivial (Quadro 3 do Exemplo 2.3.1), já que nada disso foi feito nos livros que foram utilizados como inspiração.

Estes também não possuem uma justificativa para o tempo, quando é, em tempo polinomial na decisão de problemas com máquinas de Turing não determinísticas, já que temos um número, possivelmente, exponencial de folhas que precisam ser analisadas. Eles apenas possuem uma justificativa para a origem polinomial dessas folhas, porém não para a obtenção da resposta ("sim" ou "não") a partir delas. Algo que justificamos com a associação da máquina de Turing não determinística à estratégia de *divisão e conquista* (Subseção 4.1) e com a definição e justificativa do tempo de execução para a nossa variação (Definição 4.2.8).

## 5.2 TRABALHOS FUTUROS

Neste trabalho, apenas nos preocupamos em definir, para a nossa variação, a medida de complexidade para um recurso computacional, o tempo (Definição 4.2.8), mas ainda há diversos outros que podem ser medidos, por exemplo, espaço. Sugerimos como trabalhos futuros a definição de outras medidas e a definição de classes para estas medidas.

E, por termos apenas nos preocupados com o tempo, acabamos abusando de outros recursos. Não nos preocupamos, por exemplo, com o espaço necessário para o analisador (Definição 3.1.10) e para as combinações feitas pela combinadora (Definição 4.2.2) e como a sua limitação influencia no tempo. Sugerimos como trabalhos futuros a limitação deste e de outros recursos e o estudo de otimizações para amenizar o impacto destas limitações.

Para mostrar a versatilidade de nossa variação, dos cinco exemplos que demos, apenas um era de um problema específico, o Exemplo 4.3.5, sobre o Problema de Deutsch-Jozsa, e os outros quatro eram de máquinas gerais que não correspondem a um problema em específico. Sugerimos como trabalho futuro a definição/programação de máquinas de Turing não determinística com combinadora para problemas específicos.

Além do exemplo, de um problema específico, para o Problema de Deutsch-Jozsa, também pretendíamos dar o exemplo de uma máquina que resolvesse fórmulas da lógica trivalente de Kleene, $K_3$. Esta é uma lógica não clássica feita para trabalhar com funções parciais que, para representar o indefinido (indecidível), adiciona um terceiro valor de verdade $u$ a lógica clássica (MORTARI, 2001; KLEENE, 1952). Por exemplo, seja $f$ e $g$ duas funções parciais computadas por máquinas de Turing, se $f(x) \neq y$ e $g(x)$ é indefinida, então, "$f(x) = y$ ou $g(x) = y$" é $u$.

Nossa ideia era, já que todas as funções de verdade de $K_3$ atendem as propriedades da função computada pela combinadora (Definição 4.2.2), definir uma máquina que originasse uma configuração final com o conteúdo de saída da fita igual ao valor de verdade de, e para toda subfórmula atômica de uma fórmula, sua entrada, de $K_3$, e combinasse estes valores para computar o valor de verdade, sua saída, desta fórmula. Assim, sua árvore de computação se assemelharia à árvore de suas subfórmulas.

Mas, para definirmos esta máquina de maneira simples, já que há mais de uma função de verdade, ou seja, formas de combinar, precisaríamos de uma definição para a composição de máquinas e/ou a definição de mais de uma combinadora por máquina. Sugerimos como trabalhos futuros a definição de como compor máquinas de Turing não determinísticas com combinadora e uma definição alternativa, em que a combinadora é definida para cada transição não determinística.

Para a elaboração dos exemplos, foi desenvolvido um simulador, cujo o código-fonte está disponível e documentado no Apêndice A. Este foi feito na linguagem de programação JavaScript e, por isso, pode ser facilmente incorporado em páginas interativas da *web*. Sugerimos como trabalho futuro o design de uma interface gráfica para o simulador desenvolvido, que facilite a edição e a visualização de máquinas de Turing.

# REFERÊNCIAS

AARONSON, S. $P \stackrel{?}{=} NP$. 2017. Disponível em: <https://www.scottaaronson.com/blog/?p=3095>. Acesso em: 1 mar. 2019.

BLUM, M. A Machine-Independent Theory of the Complexity of Recursive Functions. *Journal of the Association for Computing Machinery*, New York, NY, USA, v. 14, n. 2, p. 322–336, abr. 1967.

BURR, M. *Is there a mathematical equivalent to a variadic function?* 2016. Mathematics Stack Exchange. Disponível em: <https://math.stackexchange.com/q/1705136>. Acesso em: 1 mar. 2019.

CHURCH, A. Review: A. M. Turing, On Computable Numbers, with an Application to the Entscheidungsproblem. *Journal of Symbolic Logic*, v. 2, n. 1, p. 42–43, mar. 1937.

COOK, S. The P versus NP problem. 2000. Disponível em: <http://www.claymath.org/sites/default/files/pvsnp.pdf>. Acesso em: 1 mar. 2019.

COPELAND, B. J. *The Essential Turing*: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life Plus The Secrets of Enigma. Oxford: Oxford University Press, 2004.

COPELAND, B. J. The Church-Turing Thesis. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy*. Winter 2017 Edition. Stanford: Metaphysics Research Lab, Stanford University, 2017. Disponível em: <https://plato.stanford.edu/archives/win2017/entries/church-turing/>. Acesso em: 1 mar. 2019.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Introduction to Algorithms*. 3. ed. Cambridge, Massachusetts: MIT Press, 2009.

DAVIS, M. *The Undecidable*: Basic Papers on Undecidable Propositions, Unsolvable Problems, and Computable Functions. Dover ed. Mineola, New York: Academic Press, 2004.

DAVIS, M. D.; SIGAL, R.; WEYUKER, E. J. *Computability, Complexity, and Languages*: Fundamentals of Theoretical Computer Science. 2. ed. Boston: Raven Press Books, 1994.

DEAN, W. Computational Complexity Theory. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy*. Winter 2016 Edition. Stanford: Metaphysics Research Lab, Stanford University, 2016. Disponível em: <https://plato.stanford.edu/archives/win2016/entries/computational-complexity/>. Acesso em: 1 mar. 2019.

ERICKSON, J. Algorithms. 2018. Disponível em: <http://jeffe.cs.illinois.edu/teaching/algorithms/>. Acesso em: 1 mar. 2019.

FORTNOW, L. *Blum Complexity Measures*. 2004. Disponível em: <https://blog.computationalcomplexity.org/2004/04/blum-complexity-measures.html>. Acesso em: 1 mar. 2019.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Padrões de Projeto*: Soluções Reutilizáveis de Software Orientado a Objetos. Tradução: Luiz A. Meirelles Salgado. Consultoria, supervisão e revisão técnica: Fabiano Borges Paulo. Porto Alegre: Bookman, 2000.

GASARCH, W. I. Guest Column: The Third P=?NP Poll. *SIGACT News*, New York, NY, USA, v. 50, n. 1, p. 38–59, mar. 2019.

GIBBONS, J. *Algebras for Tree Algorithms*. 1991. Tese (Doutorado em Ciências da Computação) — Programming Research Group, Oxford University, Oxford, 1991.

GOLDREICH, O. *Computational Complexity*: A Conceptual Perspective. New York: Cambridge University Press, 2008.

HOPCROFT, J. E.; ULLMAN, J. D. *Introduction to Automata Theory, Languages and Computation*. Reading, Massachusetts: Addison-Wesley, 1979.

KLEENE, S. C. *Introduction to Metamathematics*. Amsterdam: North Holland, 1952.

KRENTEL, M. W. The Complexity of Optimization Problems. *Journal of Computer and System Sciences*, v. 36, n. 3, p. 490–509, jun. 1988.

LEWIS, H. R.; PAPADIMITRIOU, C. H. *Elements of the Theory of Computation*. 2. ed. Upper Saddle River, New Jersey: Prentice Hall, 1998.

MORTARI, C. A. *Introdução à Lógica*. São Paulo: Editora UNESP, 2001.

NAGEL, E.; NEWMAN, J. R. *Godel's Proof*. Edição e prefácio: Douglas R. Hofstadter. New York: NYU Press, 2001.

NASH, J. Letter to the United States National Security Agency. jan. 1955. Disponível em: <https://www.nsa.gov/Portals/70/documents/news-features/declassified-documents/nash-letters/nash_letters1.pdf>. Acesso em: 1 mar. 2019.

PETZOLD, C. *The Annotated Turing*: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine. Indianapolis, Indiana: Wiley, 2008.

POLLACHINI, G. G. *Computação Quântica*: uma abordagem para estudantes de graduação em Ciências Exatas. 2018. Monografia (Bacharelado em Engenharia Eletrônica) — Universidade Federal de Santa Catarina, Florianópolis, 2018.

POST, E. L. Finite Combinatory Processes-Formulation 1. *Journal of Symbolic Logic*, v. 1, n. 3, p. 103–105, set. 1936.

RICH, E. *Automata, Computability and Complexity*: Theory and Applications. Upper Saddle River, NJ: Pearson Prentice Hall, 2008.

ROYER, T. *Máquinas de Turing não-determinística como computadores de funções*. 2015. Monografia (Bacharelado em Ciências da Computação) — Universidade Federal de Santa Catarina, Florianópolis, 2015.

SIPSER, M. *Introdução à Teoria da Computação*. Tradução técnica: Ruy José Guerra Barretto de Queiroz. Revisão técnica: Newton José Vieira. 2. ed. São Paulo: Cengage Learning, 2007.

STROUSTRUP, B. *The C++ Programming Language*. 4. ed. Upper Saddle River, NJ: Addison-Wesley, 2013.

SUDKAMP, T. A. *Languages and Machines*: An Introduction to the Theory of Computer Science. 2. ed. Reading, Massachusetts: Addison-Wesley, 1997.

TURING, A. M. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, v. 42, n. 2, p. 230–265, nov. 1936.

VALIANT, L. G. The Complexity of Computing the Permanent. *Theoretical Computer Science*, v. 8, n. 2, p. 189–201, dez. 1979.

# APÊNDICE A – SIMULADOR

*Any application that can be written in JavaScript,*
*will eventually be written in JavaScript*
*(Jeff Atwood)*

Neste apêndice, mais especificamente nas próximas três páginas, no Código A.2, disponibilizamos o código-fonte do simulador desenvolvido para a elaboração dos exemplos. Este é uma mescla de todas as definições feitas neste trabalho, ou seja, podemos, com ele simular uma máquina de Turing multifita não determinística com analisador e com combinadora para funções variádicas (e qualquer combinação destes adjetivos).

Utilizamos a linguagem de programação JavaScript, pois assim o simulador pode ser facilmente incorporado em páginas interativas da *web*. Para que os nomes dos identificadores estejam em harmonia com as palavras reservadas da linguagem, escolhemos nomes em inglês, porém, documentamos o código com comentários em português.

Cada trecho do código corresponde a uma ou mais definições e, para manter o estilo destas, utilizamos o padrão de projeto *Factory Method* (GAMMA *et al.*, 2000), pois assim podemos escrever, na sintaxe do JavaScript, expressões do tipo `M.Config = (a, q, s, b) => { ... }` (linha 30 do Código A.2). Para facilitar o entendimento e leitura do código, estas definições estão indicadas e sublinhadas nos comentários. Note que priorizamos mais esta correspondência e a legibilidade do código do que a performance.

Abaixo, no Código A.1, exemplificamos como é utilizado o simulador, mais especificamente, como este foi utilizado para a definição/programação da máquina de Turing não determinística do Exemplo 2.3.1, das somas das subsequências:

```
import Turing from "turing";
const delta = (q, s) => {
  if (q == "q0") {
    if (s == "B") {
      return [{ p: "qr", s_: ["B"], X: ["o"] }];
    }
    return [{ p: "qi", s_: [s], X: [">"] }, { p: "qe", s_: ["E"], X: [">"] }];
  }
  // ...
}
const M = Turing.Machine(delta, { A = ["qa"] });
const w = "+#+++#---";
const tree = M.tree(w);
```

Código A.1 – Exemplo de uso do simulador. Fonte: elaborado pelo autor.

O código do simulador também está disponível em <https://github.com/joaogui/tcc> junto com o código dos exemplos, o *script* para gerar os grafos das ilustrações a partir de uma `Turing.Machine.tree` e manual de uso. O código é, assim como este trabalho, de domínio público. Apenas sugerimos a quem o utilize que cite, se possível, o trabalho, a Universidade Federal de Santa Catarina e o autor.

```
// Nome do arquivo: turing.js

// Construtor de máquinas de Turing

// Dependendo do que é passado, a máquina construída pode ser:
//   – de uma única fita ou multifita (Definição 2.1.1 ou Definição 2.2.1)
//   – determinística ou não determinística (Definição 2.1.1 ou Definição 2.3.1)
//   – com analisador (Definição 3.1.10)
//   – com combinadora (Definição 4.2.2)
//   – para funções variádicas (Definição 3.1.15)

const Machine = (
  delta, // função de transição (Definição 2.1.3 + Definição 2.3.1)
         // recebe (q, s1, ..., sn)
         // e retorna [{p, s_: [s_1, ..., s_n], X: [X1, ..., Xn]}, ...]
  {
    Y = ["0", "1"], // alfabeto de saída (Definição 3.1.7)
    q0 = "q0", // estado inicial
    B = "B", // símbolo que representa ''em branco''
    sep = "#", // símbolo separador (Definição 3.1.15)
    A = ["qa"], // estados de aceitação
    D, // analisador (Definição 3.1.10)
    C // combinadora (Definição 4.2.2)
  } = {}) => { // ''= {}'' torna o que está entre chaves (Y, q0, B, sep, A, D, C) opcional

  const M = {delta, q0, Y, B, sep, A, D, C};

  M.k = M.delta.length - 1;

  M.Config = (a, q, s, b) => { // configuração de M (Definição 2.2.2)
                               // a, s, e b são listas
    const c = {a, q, s, b};
    c.next = () => { // configurações que são originas por um movimento a partir de c
                     // (Definição 2.2.3 + Definição 2.3.2)
      const next = [];
      const transitions = M.delta(c.q, ...c.s);
      for (const { p, s_, X } of transitions) {
        const [a, s, b] = [[], [], []];
        for (let i = 0; i < M.k; i += 1) {
          if (X[i] == "o") {
            a[i] = c.a[i];
            s[i] = s_[i];
            b[i] = c.b[i];
          } else if (X[i] == "<") {
            if (c.a[i].length == 0) {
              a[i] = "";
              s[i] = M.B;
              b[i] = c.b[i] == "" && s_[i] == M.B ? "" : s_[i] + c.b[i];
            } else {
              a[i] = c.a[i].slice(0, c.a[i].length - 1);
              s[i] = c.a[i][c.a[i].length - 1];
              b[i] = c.b[i] == "" && s_[i] == M.B ? "" : s_[i] + c.b[i];
            }
          } else if (X[i] == ">") {
            if (c.b[i].length == 0) {
              a[i] = c.a[i] == "" && s_[i] == M.B ? "" : c.a[i] + s_[i];
              s[i] = M.B;
              b[i] = "";
            } else {
```

```
60                a[i] = c.a[i] == "" && s_[i] == M.B ? "" : c.a[i] + s_[i];
61                s[i] = c.b[i][0];
62                b[i] = c.b[i].slice(1);
63              }
64            }
65          }
66          next.push(M.Config(a, p, s, b));
67        }
68        return next;
69      };
70      c.output = () => { // conteúdo de saída da fita de c (Definição 3.1.7)
71        return (c.a[0] + c.s[0] + c.b[0]).split("").filter(s => M.Y.includes(s)).join("");
72      };
73      return c;
74    };
75
76    M.Config.c0 = (...ws) => { // configuração inicial de M (Definição 2.2.4 + Definição 3.1.16)
77                               // ''...ws'' é uma lista
78      const w = ws.join(sep);
79      if (w.length == 0) {
80        return M.Config(
81          Array(M.k).fill(""),  // Array(n).fill(X) retorna
82          M.q0,                 // uma lista de tamanho n
83          Array(M.k).fill(M.B), // preenchida com Xs
84          Array(M.k).fill("")
85        );
86      }
87      return M.Config(
88        Array(M.k).fill(""),
89        M.q0,
90        [w[0], ...Array(M.k - 1).fill(M.B)],
91        [w.slice(1), ...Array(M.k - 1).fill("")]
92      );
93    };
94
95    M.tree = (...ws) => { // árvore de computação de M (Definição 2.3.3 + Definição 4.2.5)
96
97      // Teorema 2.5 + Teorema 4.1
98      const root = M.tree.Node(M.Config.c0(ws));
99      const queue = [root];
100     while (queue.length != 0) {
101       const node = queue.shift();
102       for (const c of node.config.next()) {
103         node.children.push(M.tree.Node(c));
104         if (M.C != undefined) {
105           if (root.output() != undefined)  {
106             break;
107           }
108         } else {
109           if (delta(c.q, ...c.s).length == 0 && A.includes(c.q)) {
110             break;
111           }
112         }
113       }
114       queue.push(...node.children);
115     }
116     return root;
117   };
118
```

```
119   M.tree.Node = (config, children = []) => { // vértice de uma árvore de computação de M
120     const node = {config, children};
121     node.leafs = () => {
122       const leafs = [];
123       if (node.children.length == 0) {
124         leafs.push(node);
125       } else {
126         for (const child of node.children) {
127           leafs.push(...child.leafs());
128         }
129       }
130       return leafs;
131     };
132     node.output = () => { // saída do vértice (Definição 4.2.4)
133       if (M.C == undefined) {
134         return undefined;
135       }
136       if (node.children.length == 0) {
137         const c = node.config;
138         if (M.delta(c.q, ...c.s).length == 0) {
139           return c.output();
140         }
141         return undefined;
142       }
143       if (node.children.length == 1) {
144         return node.children[0].output();
145       }
146       return M.C.f(...node.children.map(node => node.output()));
147     };
148     return node;
149   };
150
151   M.accepts = w => { // aceitação (Definição 3.1.11) ou rejeição (Definição 2.3.4)
152     return M.tree(w).leafs().filter(node => {
153       const c = node.config;
154       return delta(c.q, ...c.s).length == 0 && A.includes(c.q);
155     }).length >= 1;
156   };
157
158   M.f = (...xs) => { // função computada por M (Definição 3.1.11 + Definição 4.2.6)
159     if (D == undefined || D.accepts(...xs)) {
160       if (M.C == undefined) {
161         const leafs = M.tree(xs).leafs();
162         if (leafs.length == 1) {
163           return leafs[0].config.output();
164         }
165         return undefined;
166       }
167       return M.tree(xs).output();
168     }
169     return undefined;
170   };
171
172   return M;
173 };
174
175 export default {Machine};
```

Código A.2 – Código do simulador. Fonte: elaborado pelo autor.