

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA**

Gian Carlo Figueiredo Ferrari

**HISTÓRICOS DE COMPUTAÇÃO COMO DINÂMICA
DISCRETA**

Florianópolis
2026

Gian Carlo Figueiredo Ferrari

HISTÓRICOS DE COMPUTAÇÃO COMO DINÂMICA DISCRETA

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciência da Computação para a obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Álvaro Junio Pereira Franco

Florianópolis
2026

Gian Carlo Figueiredo Ferrari

HISTÓRICOS DE COMPUTAÇÃO COMO DINÂMICA DISCRETA

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de “Bacharel em Ciência da Computação”, e aprovado em sua forma final pelo Programa de Graduação em Ciência da Computação.

Florianópolis, 15/06/2026.

Prof. Dra. Lúcia Helena Martins Pacheco
Coordenadora do Curso

Prof. Dr. Álvaro Junio Pereira Franco
Orientador

Profa. Dra. Jerusa Marchi
Co-Orientadora

Banca Examinadora:

Prof. Dr. John H. Castillo
Universidade de Nariño

Prof. Dr. Rafael de Santiago
Universidade Federal de Santa Catarina

Prof. Dr. Olinto José Varela Furtado
Universidade Federal de Santa Catarina

Se as pessoas não acreditam que a matemática é simples, é apenas porque não percebem o quão complicada é a vida.

(John von Neumann, tradução livre)

SUMÁRIO

1	Introdução	1
1.1	Objetivos	2
1.1.1	Objetivos Específicos	2
2	Fundamentação Teórica	3
2.1	Teoria de Conjuntos	3
2.1.1	Relações	4
2.1.2	Enumerabilidade	6
2.2	Dinâmica Discreta	7
2.3	Grafos	9
2.3.1	Grafos Funcionais	10
3	O Computador	11
3.1	Definições	11
3.2	Linguagens Recursivamente Enumeráveis	15
3.3	Não-determinismo	18
4	Dinâmica Computacional	20
4.1	Órbitas e Componentes	20
5	Simulação	27
5.1	Simulação Computacional	27
6	Complexidade Orbital dos Núcleos	34
6.1	Complexidade Orbital	34
7	Conclusões e Trabalhos Futuros	36
	Apêndice A — Artigo em Formato SBC	40

Resumo

Este trabalho propõe uma abordagem para o estudo da computação sob uma perspectiva dinâmica. Para isso, introduz-se uma estrutura matemática simples baseada em um conjunto de funções sobre os números naturais, a partir da qual são definidos computadores, passos de computação, históricos e simulações. Essas definições permitem interpretar a evolução de um processo computacional como uma dinâmica discreta, tornando possível analisar propriedades de sua evolução por meio de conceitos inspirados na teoria de sistemas dinâmicos.

Ao longo do texto, são estabelecidas propriedades gerais dos históricos de computação e de suas órbitas, mostrando que comportamentos característicos da computação determinística, como parada, periodicidade e evolução infinita, surgem naturalmente como consequências da estrutura proposta. Também é introduzida uma noção de simulação entre computadores, baseada na preservação dos passos de computação, permitindo comparar diferentes sistemas sob uma perspectiva estrutural.

Por fim, é apresentada uma noção de complexidade orbital, que associa aos históricos de computação uma medida de crescimento inspirada em conceitos clássicos de complexidade. Embora exploratória, a abordagem desenvolvida fornece uma linguagem unificada para descrever aspectos dinâmicos da computação e sugere novas direções para o estudo da relação entre teoria da computação e sistemas dinâmicos.

Keywords: computação, sistemas dinâmicos, máquinas de Turing, simulação, complexidade computacional.

1 INTRODUÇÃO

A teoria da computação busca compreender os limites, capacidades e propriedades dos processos computacionais por meio de modelos formais. Embora diversos desses modelos sejam equivalentes em poder computacional, conforme a tese de Church–Turing, cada um evidencia aspectos distintos da computação. Por exemplo, a complexidade descritiva (Immerman 1999) caracteriza classes de complexidade por meio de linguagens lógicas, enquanto a teoria axiomática de Blum (Blum 1967) fornece uma abordagem abstrata para a complexidade de funções recursivas, independente de um modelo específico de máquina.

Grande parte desses modelos concentra-se na descrição dos mecanismos responsáveis pela computação ou na caracterização das linguagens e funções computáveis. Neste trabalho, adota-se uma perspectiva complementar: considera-se a computação como um processo já estabelecido e investiga-se a estrutura matemática produzida por sua execução. Em vez de enfatizar o procedimento efetivo em si, busca-se compreender as propriedades do histórico de computação gerado por esse procedimento.

Para isso, propõe-se um modelo no qual um computador é definido como um conjunto de funções sobre os números naturais. A partir desse modelo são introduzidos os conceitos de passo de computação, histórico e iteração, permitindo interpretar a evolução de um processo computacional como uma dinâmica discreta. Essa interpretação torna possível aplicar conceitos inspirados na teoria de sistemas dinâmicos ao estudo da computação, como órbitas, componentes, periodicidade e comportamento assintótico.

Mostra-se ainda que, a menos de uma bijeção, toda linguagem recursivamente enumerável pode ser representada como um subconjunto específico dos naturais associado a um computador do modelo, estabelecendo sua compatibilidade com a tese de Church–Turing. Em seguida, introduz-se uma noção de simulação entre computadores, formulada como uma semiconjugação entre as dinâmicas induzidas pelos respectivos históricos de computação, permitindo comparar estruturalmente diferentes computadores.

Por fim, propõe-se uma noção exploratória de complexidade orbital, inspirada em conceitos clássicos de complexidade computacional, cujo objetivo é associar medidas de crescimento aos históricos de computação produzidos pelo modelo.

1.1 Objetivos

Propor e estudar um modelo simplificado de computação cuja formulação estabeleça uma conexão direta entre teoria da computação e dinâmica em espaços discretos, permitindo analisar históricos de computação por meio de ferramentas inspiradas em sistemas dinâmicos.

1.1.1 Objetivos Específicos

1. Definir formalmente os elementos fundamentais do modelo proposto;
2. Estudar a estrutura dinâmica induzida pelos históricos de computação, estabelecendo propriedades e uma classificação inicial de suas componentes;
3. Definir e investigar relações de simulação entre computadores, analisando a preservação de propriedades dinâmicas;
4. Introduzir uma noção preliminar de complexidade orbital para os históricos de computação.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo veremos terminologia, definições, e propriedades que serão úteis ao longo do trabalho. Em particular, elementos da teoria de conjuntos, de grafos e noções de dinâmica irão ser apresentadas.

2.1 Teoria de Conjuntos

Apresentaremos apenas o que for suficiente para sustentar a notação e os resultados utilizados no restante do texto. Como usual na literatura, $\in$ denota o pertencimento como relação, ou seja, $x \in A$ é equivalente a ‘ x pertence a A ’. Usaremos $A \subseteq B$ para indicar que A é um subconjunto de B , e neste caso também usaremos a definição usual

$$A \subseteq B \iff \forall x(x \in A \implies x \in B). \quad (2.1)$$

Também usaremos a definição extensional de igualdade, dada por

$$A = B \iff \forall x(x \in A \iff x \in B) \quad (2.2)$$

que pode também ser expressa por $A = B \iff (A \subseteq B \wedge B \subseteq A)$. Outras relações usuais são união $\cup$ e interseção $\cap$, exibidas a seguir respectivamente:

$$\forall x(x \in A \cup B \iff x \in A \vee x \in B), \quad (2.3)$$

$$\forall x(x \in A \cap B \iff x \in A \wedge x \in B). \quad (2.4)$$

Eventualmente será usada a notação

$$\bigcup_{i \in I} A_i \quad (2.5)$$

para denotar a união dos conjuntos A_i indexados por $i \in I$. Também poderá ser usada apenas a notação $\bigcup A$ para indicar o conjunto resultante da união de todos

os conjuntos $X \in A$ (onde A é uma coleção de conjuntos). Estas notações têm seu caso análogo para a interseção.

Definimos $A \times B$ como o conjunto

$$A \times B = \{(a, b) : a \in A \wedge b \in B\}, \quad (2.6)$$

chamado de **produto cartesiano** de A com B . Note que todos os pares possíveis estão considerados. Denotaremos o conjunto dos números naturais $\{0, 1, 2, 3, \dots\}$ por $\mathbb{N}$.

2.1.1 Relações

Relações serão rigorosamente subconjuntos do produto cartesiano de dois conjuntos. Neste caso mais geral, não se estipulam regras acerca da relação. No entanto, veremos duas relações específicas onde esperaremos determinado comportamento. Em relação à notação, se $R \subseteq A \times B$ está sendo lida como relação, podemos denotar como $R(a, b)$, $R(a) = b$ ou ainda aRb . A notação infixada fará mais sentido adiante.

Funções

Um tipo de relação específica que será muito pertinente neste trabalho é a relação de função. Diremos que $f : A \rightarrow B$ é uma função se

1. $f \subseteq A \times B$, ou seja, se for uma relação;
2. Toda vez que $(a_1, b_1) \in f$ e $(a_1, b_2) \in f$ temos que $b_1 = b_2$; e
3. Para todo $a \in A$ existe $b \in B$ tal que $(a, b) \in f$.

Imediatamente iremos adotar a notação usual de função dada por $f(a_1) = b_1$, ou simplesmente $f(a) = b$ para o par $(a, b) \in f$. Podemos reescrever nossa condição de função então como: $f(a_1) = b_1$ e $f(a_1) = b_2$ implica $b_1 = b_2$. O conjunto A será chamado de *domínio* de f , e o conjunto B de *contradomínio*. A *imagem* de f é também a usual:

$$Im(f) = \{f(a) : a \in A\}. \quad (2.7)$$

Por vezes, não se é exigida a terceira condição, de que todo elemento de A tenha imagem em B . Neste caso, costuma-se chamar a relação de **função parcial**. Não usaremos esta nomenclatura diretamente, mas uma forma análoga irá aparecer.

Outras definições úteis são: a imagem de um conjunto $X \subseteq A$ por uma função $f : A \rightarrow B$, dada por

$$f(X) = \{f(x) : \forall x \in X\}, \quad (2.8)$$

e a pré-imagem de um conjunto $Y \subseteq B$ por f , dada por

$$f^{-1}(Y) = \{x \in A : f(x) \in Y\}. \quad (2.9)$$

Note que temos então $f(X) \subseteq B$, $f^{-1}(Y) \subseteq A$, e que $Im(f) = f(A)$.

A composição de funções é a usual, denotada por $g \circ f$ e definida se $f : A \rightarrow B$ e $g : B \rightarrow C$, ou seja, se o domínio de g é o contradomínio de f . Escreve-se também $g \circ f(b) = g(f(b))$ para todo $b \in f(A)$.

Usaremos as definições usuais para funções injetoras, sobrejetoras e bijetoras, que ressaltamos abaixo. Se $f : X \rightarrow Y$, definimos:

1. f é injetora $\iff$ se $f(x_1) = f(x_2)$ então $x_1 = x_2$;
2. f é sobrejetora $\iff$ para todo $y \in Y$ existe $x \in X$ com $y = f(x)$; e
3. f é bijetora $\iff$ f é injetora e sobrejetora.

No caso, as funções bijetoras nos permitem considerar uma *função inversa*, dada por $f^{-1} : Y \rightarrow X$ obedecendo $f \circ f^{-1}(y) = y$ para todo $y \in Y$ e $f^{-1} \circ f(x) = x$ para todo $x \in X$. A inversa f^{-1} é também uma bijeção. Deixaremos claro quando f^{-1} diz respeito à inversa de uma bijeção ou à pré-imagem de um conjunto ou elemento.

Equivalência

Outro tipo de relação específica que aparecerá no trabalho é a relação de equivalência. Esta usará da notação infixada, e focaremos apenas no caso $R \subseteq A \times A$, ou seja, de uma relação nos elementos de A apenas. Se R for

1. Reflexiva: aRa para todo $a \in A$;
2. Simétrica: aRb se, e somente se, bRa ; e
3. Transitiva: aRb e bRc implicam aRc para todo $a, b, c \in A$,

então diremos que R é uma relação de equivalência.

Um resultado importante é que se $R \subseteq A \times A$ for uma relação de equivalência, obtemos de imediato uma *partição* do espaço, isto é, um conjunto de subconjuntos

de A , disjuntos entre si, que unidos formam A . Os conjuntos da partição *induzida* por R serão os conjuntos

$$\bar{x} = \{a \in A : xRa\}. \quad (2.10)$$

A notação deste conjunto por $\bar{x}$ tem um detalhe: qualquer elemento de $\bar{x}$ pode ser substituído pelo x na notação. Na literatura, costuma-se dizer que qualquer elemento de $\bar{x}$ funciona como *representante* de seu conjunto. Este conjunto também tem o nome de **classe de equivalência** de x . O conjunto de classes de equivalência, obtidas por uma relação de equivalência R , é denotado por A/R .

2.1.2 Enumerabilidade

Iremos definir que um conjunto A é enumerável se, e somente se, A é finito ou existe uma função bijetora de A em $\mathbb{N}$. Essa é uma definição usual (ou pelo menos uma equivalência) na literatura. Iremos utilizar do seguinte resultado da teoria de conjuntos:

A união enumerável de conjuntos enumeráveis é também enumerável.

Ou seja, se I é enumerável e todo A_i com $i \in I$ também o for, temos que

$$\bigcup_{i \in I} A_i \quad (2.11)$$

é enumerável. Podemos observar este resultado de forma intuitiva e informal. Se I é enumerável, podemos listar os A_i em ordem. Da mesma forma, como cada A_i é enumerável, podemos listar seus elementos em ordem. Obteríamos algo da forma (para $i, j, k, \dots \in I$)

$$\begin{array}{cccc} A_i & A_j & A_k & \cdots \\ i_1 & j_1 & k_1 & \cdots \\ i_2 & j_2 & k_2 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{array} \quad (2.12)$$

onde $i_1, i_2, \dots \in A_i$, $j_1, j_2, \dots \in A_j$, e $k_1, k_2, \dots \in A_k$. Conseguimos uma forma de listar todos os elementos da união, a saber, percorrendo a diagonal da direita para a esquerda começando por i_1 :

$$\bigcup_{i \in I} A_i = \{i_1, j_1, i_2, k_1, j_2, i_3, \dots\}. \quad (2.13)$$

Por exemplo, se um conjunto K qualquer for demonstrado ser a união de conjuntos enumeráveis K_i para $i \in \mathbb{N}$, então K é enumerável e existe uma bijeção

$\phi : K \rightarrow \mathbb{N}$. Aplicaremos exatamente desta forma, neste trabalho, a definição e a propriedade exibidas aqui.

Aprofundamento acerca teoria de conjuntos e enumerabilidade pode ser observada em (Enderton 1977), (Halmos 1960) ou (Jech 2003).

2.2 Dinâmica Discreta

Considere uma função $f : C \rightarrow C$. Para fins de exemplo, suponhamos que C seja um conjunto numérico como $\mathbb{R}$ ou $\mathbb{Q}$. Tome $x \in C$ e considere a sequência dada por

$$(x, f(x), f(f(x)), f(f(f(x))), \dots) \quad (2.14)$$

de aplicações sucessivas de f . Denotaremos a n -ésima aplicação de f em x por $f^n(x)$, e convencionaremos $f^0(x) = x$. Ao conjunto

$$orb(x) = \{f^n(x) : n \in \mathbb{N}\} \quad (2.15)$$

daremos o nome de **órbita** de x . O estudo da órbita de elementos de C por f , tal como propriedades de f sobre aplicações sucessivas, é parte do estudo da **dinâmica** de f (note que C está implícito, e geralmente é fixado como $\mathbb{R}$ ou $\mathbb{C}$).

Como partimos da iteração de f (i.e. de aplicações sucessivas), costuma-se classificar ainda esta forma de dinâmica como **dinâmica discreta**. O nome discreta aqui vem do fato de que o “tempo”, neste caso, é implicitamente qual passo da iteração estamos olhando. Outras formas do estudo da dinâmica de funções consideram o tempo como uma variável livre, e a dinâmica é estudada a partir de equações diferenciais nesta variável, de forma contínua.

Vejam os dois exemplos. Considere $f : \mathbb{Q} \rightarrow \mathbb{Q}$ dada por $f(x) = x^2$. A dinâmica desta função tem essencialmente três comportamentos:

1. $f(1) = 1$ e $f(0) = 0$. Dizemos neste caso que 0 e 1 são **pontos fixos**. No geral, se temos $f(x) = x$ dizemos que x é ponto fixo.
2. $f(x) < x$ se $|x| < 1$. Neste caso, a dinâmica se concentra ao redor do ponto $x = 0$.
3. $f(x) > x$ se $|x| > 1$. Neste caso, a dinâmica faz com que a aplicação sucessiva de f resulte em valores cada vez maiores.

No geral, as dinâmicas mais interessantes podem ser observadas no estudo de funções em $\mathbb{C}$. Terminaremos com um exemplo histórico deste caso. Considere

então a função $g : \mathbb{C} \rightarrow \mathbb{C}$ dada por $g(z) = z^2 - (0,123 + 0,745i)$. A dinâmica desta função tem uma característica não-trivial, que iremos explicar por analogia. No caso anterior, da função f , pudemos achar uma “fronteira” específica das duas dinâmicas no valor $x = 1$. Para a função g esta fronteira se trata de um fractal. Mesmo se classificamos a dinâmica apenas sobre a órbita de um x ser ou não limitada, obtemos uma fronteira¹ fractal.

A imagem abaixo ilustra essa fronteira (apesar da resolução limitada). Foi gerada em Python com assistência do ChatGPT da OpenAI.

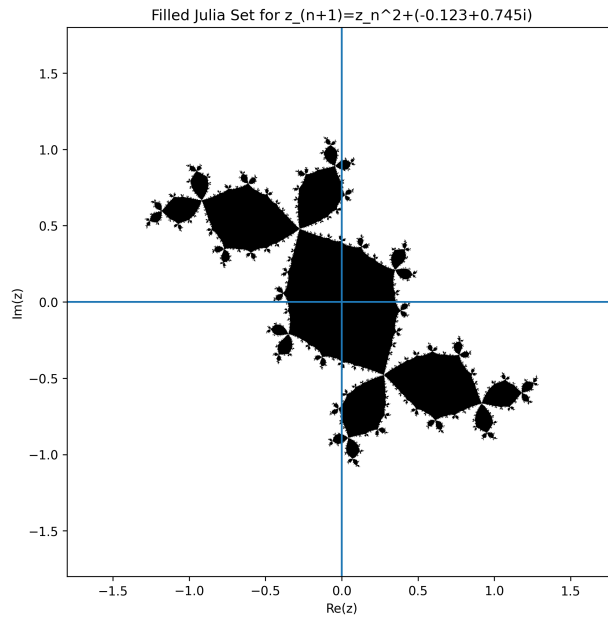


Figura 2.1: Conjunto de Julia para $g(z) = z^2 - (0,123 + 0,745i)$. Pontos em preto correspondem à valores iniciais cuja órbita se mantém limitada; os demais têm órbita ilimitada.

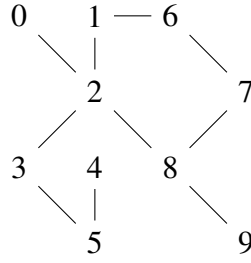
Vemos que de fato a “fronteira” entre os diferentes comportamentos orbitais é extremamente detalhada e não-trivial. Para espaços contínuos como $\mathbb{R}$ e $\mathbb{C}$, ferramentas do cálculo e da análise são capazes de ajudar no estudo da dinâmica, porém para este trabalho estaremos limitados a $\mathbb{N}$ e portanto os resultados não se aplicam.

¹O uso do termo ‘fronteira’ aqui é um artifício intuitivo, pois de fato esta fronteira não existe na forma usual como pensamos uma fronteira.

Dinâmica discreta, partindo da iteração de funções, pode ser estudada em (Sandefur 1990). Acerca da dinâmica em funções de uma variável complexa, estas podem ser estudadas em (Milnor 2006).

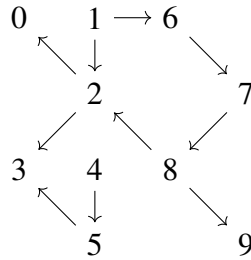
2.3 Grafos

Na forma geral, um grafo é uma dupla $G = (V, E)$ onde V é um conjunto não-vazio, chamado de vértices, e $E \subseteq V \times V$ é o conjunto de arestas. Casos simples de grafos podem ser representados por diagramas. Por exemplo, tome $V = \{0, 1, \dots, 9\}$. Podemos inferir E pelo diagrama:



Desta forma, observamos que alguns elementos de E são $(1, 2)$, $(4, 5)$ e $(8, 9)$. Neste caso, estamos também considerando o grafo como **não-dirigido**, ou seja, consideramos que $(1, 2)$ e $(2, 1)$ descrevem a mesma aresta.

Para exemplificar um caso de grafo dirigido, considere



Neste caso, temos a aresta $(1, 2)$ mas não a aresta $(2, 1)$. Podemos obter a versão não-dirigida de um grafo por meio de uma relação de equivalência. Para isto, considere um grafo G , e defina a seguinte relação de equivalência:

$$(a, b) \equiv (c, d) \iff (a, b) = (c, d) \vee (a = d \wedge b = c). \quad (2.16)$$

O grafo G' montado nas classes de equivalência $E/\equiv$ ² tem o mesmo diagrama do grafo G , exceto pelo fato das arestas serem não-dirigidas.

Se existem arestas $u_1, \dots, u_n \in E$ tais que $a, b \in V$ forem tais que $a \in u_1, b \in u_n$ e $x \in u_i \cap u_{i+1}$ para $x \neq a, b$ ³, diremos que a e b estão **conectados**. O conjunto de todos os vértices conectados a um $a \in V$ diremos que é a **componente conexa** a a . Geralmente, consideraremos apenas as componentes conexas sem mencionar um vértice específico.

Para grafos não-dirigidos, definimos o **grau** de um vértice como o número de arestas das quais o vértice é parte. Para grafos dirigidos, distinguimos entre grau de **entrada**, dado pelo número de arestas da qual a vértice é a segunda entrada, e grau de **saída**, dado pelo número de arestas da qual a vértice é a primeira entrada. No exemplo anterior, temos que o vértice 3 tem grau de entrada igual a 2 e grau de saída igual a 0. O vértice 4 tem grau de saída 1.

2.3.1 Grafos Funcionais

Um grafo dirigido específico que trabalharemos será chamado de grafo funcional. Este é obtido a partir de uma função $f : C \rightarrow C$. Consideraremos C o conjunto de vértices e as arestas serão dadas por

$$E = \{(x, f(x)) : x \in C\}. \quad (2.17)$$

Uma propriedade destes grafos é que todo vértice tem grau de saída no máximo igual a 1, pois caso contrário f não seria uma função.

Grafos podem ser estudados em (West 2001) e (Diestel 2017). O termo grafo funcional aparece explicitamente em (Harary e Palmer 1973).

²Esticando um pouco o rigor.

³Abusando da notação.

3 O COMPUTADOR

Neste capítulo, veremos as definições iniciais que propomos para obter os históricos de computação. Começaremos pelo ingrediente mais básico, aqui chamado de “computador”, e detalharemos alguns elementos. Finalizamos com uma forma de visualizar estes computadores que deverá servir de base para o resto do trabalho.

3.1 Definições

Usaremos $\mathbb{N}$ para indicar o conjunto usual dos números naturais: $\mathbb{N} = \{0, 1, 2, \dots\}$. Denotaremos por $\mathbb{N}^*$ o conjunto dos naturais sem o zero.

Definição 3.1.1. Sejam $D_1, D_2, \dots, D_k$ subconjuntos não-vazios de $\mathbb{N}$. Um conjunto de funções

$$C = \{f_i : D_i \rightarrow \mathbb{N}, 1 \leq i \leq k\} \quad (3.1)$$

define um **computador** C . À união de todos os D_i damos o nome de **domínio** de C , e denotaremos por $D(C)$. Analogamente, à união de todas as imagens de cada f_i damos o nome de **imagem** de C , e denotaremos por $I(C)$. O **espaço** de um computador será o conjunto $E(C) = D(C) \cup I(C)$.

Nota. Sempre usaremos $D_1, \dots, D_k$ para denotar os domínios das funções $f_1, \dots, f_k$ de um computador qualquer. Se estamos trabalhando com mais de um computador, deixaremos claro de quais domínios e funções se trata.

Exemplo 3.1.2. Qualquer função nos naturais, por si só, pode ser usada para definir um computador.

Exemplo 3.1.3. Seja $P = \{2n : n \in \mathbb{N}\}$ e $I = \{2n+1 : n \in \mathbb{N}\}$. Note que P se trata do conjunto de números pares e I do conjunto de números ímpares. Definimos as funções $d : P \rightarrow \mathbb{N}$ e $t : I \rightarrow \mathbb{N}$ por

$$d(n) = \frac{n}{2}, t(n) = 3n+1 \quad (3.2)$$

e obtemos um computador $C = \{d, t\}$. Note que este computador é basicamente a função conhecida pela conjectura de Collatz, que pode ser estudada de forma aprofundada em (Lagarias 2010). Por vezes, chamaremos $\{d, t\}$ de computador de Collatz, e o denotaremos por CL . Temos que $D(CL) = P \cup I = \mathbb{N}$.

Definição 3.1.4. Seja C um computador e $f_i, f_j \in C$ distintas tais que $D_i \cap D_j \neq \emptyset$. Se existe $n \in D_i \cap D_j$ tal que

$$f_i(n) \neq f_j(n) \quad (3.3)$$

então diremos que C é um computador **não-determinístico**. Se $n \in D_i \cap D_j \implies f_i(n) = f_j(n)$ para todo $n \in E(C)$, diremos que C é **determinístico**.

Nota. Iremos nos restringir ao caso determinístico, exceto quando explicitado o contrário.

Definição 3.1.5. Seja C um computador e $n \in D(C)$. A **computação** de n por C é a escolha do f_i tal que $n \in D_i$ e o cálculo de $f_i(n)$. Denotaremos por $C[n]$. Em outras palavras,

$$C[n] = f_i(n) \text{ tal que } n \in D_i. \quad (3.4)$$

Nota. Note que, no caso não-determinístico, $C[n]$ não está bem definido como notação, pois se $x_i = f_i(n) \neq f_j(n) = x_j$ então $C[n]$ não corresponde a um número só. Poderíamos fazer $C[n] = \{x_i, x_j\}$, porém isso induz uma dinâmica no espaço $\mathcal{P}(\mathbb{N})$ do conjunto das partes de $\mathbb{N}$, que não será nosso foco.

Exemplo 3.1.6. Se C é um computador determinístico, então existe um computador K com uma única função $k : D(C) \rightarrow \mathbb{N}$ tal que $K[n] = C[n]$ para todo $n \in D(C)$.

Podemos definir k como

$$k = \begin{cases} f_1(n), & n \in D_1 \\ f_2(n), & n \in D_2 \\ \vdots \\ f_{|C|}(n), & n \in D_{|C|} \end{cases} \quad (3.5)$$

e como C é determinístico, se $n \in D(C)$ então existe um único $a \in I(C)$ com $g(n) = a$ para todo $g \in C$ tal que $n \in D_g$. Assim, k está bem definida. O computador $K = \{k\}$ de imediato é tal que $D(K) = D(C)$ e $K[n] = C[n], \forall n \in D(C)$.

Exemplo 3.1.7. Retomando o computador de Collatz, podemos montar k fazendo

$$k = \begin{cases} \frac{n}{2} & \text{se } n \text{ é par} \\ 3n + 1 & \text{se } n \text{ é ímpar} \end{cases} \quad (3.6)$$

e obter um computador de função única K com $K[n] = CL[n]$.

Um comportamento conhecido do computador CL é de que, se computamos iterativamente $CL[2^n]$ para qualquer $n \in \mathbb{N}$, eventualmente caímos em um ciclo. Vejamos com um exemplo: Seja $n = 4$. Temos que $2^n = 16$. Fazendo $CL[16]$, nossa única opção é $d(16) = 8$.

Continuando a iteração,

$$\begin{aligned} CL[8] &= d(8) = 4 \\ CL[4] &= d(4) = 2 \\ CL[2] &= d(2) = 1 \\ CL[1] &= t(1) = 4 \end{aligned} \tag{3.7}$$

observamos que 4, 2, 1 formam um ciclo. O fato de termos uma certa ‘região fixa’, e que podemos observá-la iterando sobre $CL[2^n]$, motiva as próximas definições.

Definição 3.1.8. Seja C um computador. A **iteração** de n em C será a sequência de computações sucessivas dadas por

$$(n, C[n], C[C[n]], C[C[C[n]]], \dots). \tag{3.8}$$

Denotaremos o k -ésimo elemento desta iteração por $C^k[n]$, com $C^0[n] = n$.

Exemplo 3.1.9. Retomando o computador CL , podemos observar a iteração de 16:

$$(16, 8, 4, 2, 1, 4, 2, 1, 4, 2, \dots). \tag{3.9}$$

A iteração de 27 começa com:

$$(27, 82, 41, 124, 62, 31, 94, 47, 142, 71, \dots) \tag{3.10}$$

e atinge um ponto de ciclo (neste caso, 4) após 109 computações¹. De fato, a conjectura de Collatz afirma que todo $n \in \mathbb{N}$ atinge o ciclo (4, 2, 1) em algum momento da iteração.

Definição 3.1.10. Seja C um computador. Se existem $x_1, \dots, x_n \in D(C)$ tais que

$$C[x_1] = x_2, C[x_2] = x_3, \dots, C[x_{n-1}] = x_n, C[x_n] = x_1 \tag{3.11}$$

diremos que $(x_1, x_2, \dots, x_n)$ é um **ciclo** em C , e chamaremos os $x_1, \dots, x_n$ de **pontos de ciclo**. Para o caso $n = 1$, diremos que x_1 é um **ponto fixo** de C . O **menor** número natural $p > 0$ tal que $C^p[x_i] = x_i$ será o **período** de x_i . No geral, abreviaremos um ciclo de tamanho n por $(x_i)_n$.

¹Não é difícil verificar isto, um pequeno script em Python resolve.

Exemplo 3.1.11. De imediato, temos que $(4, 2, 1)$ é um ciclo do computador de Collatz, com pontos de ciclo 4, 2 e 1. Pelas funções de CL , também observamos que 0 é um ponto fixo.

Exemplo 3.1.12. Considere as funções $s_1(n) = n + 1$ e $s_2(n) = n - 1$, onde s_1 está definida nos pares e s_2 nos ímpares, e defina o computador $X = \{s_1, s_2\}$. Note que todo número natural é um ponto de ciclo de X . Se fossemos diagramar esta função, teríamos vários ciclos disjuntos, como por exemplo, $(0, 1), (2, 3), (4, 5)$, e nenhum ponto fixo.

Nota. Não há unicidade na escrita para o ciclo. Se $(x_1, x_2, \dots, x_n)$ é um ciclo em um computador qualquer, $(x_2, x_3, \dots, x_n, x_1)$ é o mesmo ciclo. Além disso, qualquer um dos elementos do ciclo pode ser usado para representá-lo, pois podemos obter o ciclo todo iterando qualquer x_i , $1 \leq i \leq n$ em C .

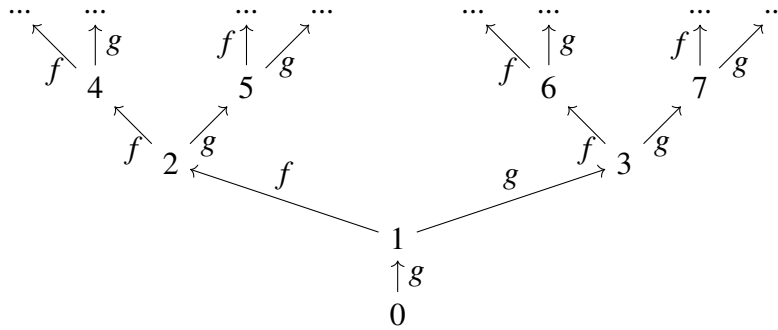
Definição 3.1.13. Dado um computador C , ao conjunto $D(C) - I(C)$ daremos o nome de conjunto de **raízes**. Denotaremos por $R(C)$.

Nota. O nome de raiz para o conjunto acima vem da seguinte visualização: podemos diagramar um dado computador começando pelos elementos da raiz, e montando árvores utilizando as funções de C para as flechas, de tal modo que uma seta de x para y nos diz que $C[x] = y$.

Exemplo 3.1.14. Considere a função sucessor nos naturais $s(n) = n + 1$, e faça C o computador com apenas esta função. Observamos que $R(C) = \{0\}$, e que podemos visualizar a função pela árvore a seguir.

$$0 \xrightarrow{s} 1 \xrightarrow{s} 2 \xrightarrow{s} 3 \xrightarrow{s} \dots$$

Exemplo 3.1.15. Considere as funções $f : \mathbb{N} - \{0\} \rightarrow \mathbb{N}$ e $g : \mathbb{N} \rightarrow \mathbb{N}$ dadas por $f(n) = 2n$ e $g(n) = 2n + 1$. O computador $C = \{f, g\}$, de raiz $R(C) = \{0\}$, pode ser visualizado pela seguinte árvore:



Note que C é não determinístico. Esse comportamento pode ser observado na visualização como árvore pelo fato de que existem números com mais de uma flecha de saída. No caso de computadores determinísticos, as flechas de saída são únicas em cada número.

Definição 3.1.16. Seja C um computador. Ao conjunto $I(C) - D(C)$ daremos o nome de conjunto de **folhas**. Denotaremos por $F(C)$.

Nota. O uso do termo folhas segue o padrão de visualização como árvore que ilustramos aqui. São os números com pelo menos uma flecha entrante mas nenhuma flecha de saída.

3.2 Linguagens Recursivamente Enumeráveis

Esta seção busca mostrar que, para cada máquina de Turing determinística, existe um computador cujo comportamento ‘imita’ o da máquina e na qual podemos observar um conjunto análogo ao da linguagem reconhecida pela máquina de Turing (neste caso, um conjunto em bijeção com a linguagem). Desta maneira concluímos que há uma ‘inclusão’, e que portanto os computadores aqui (rigorosamente, um subconjunto deles) estão em conformidade com a tese de Church-Turing.

Nos restringimos ao caso de máquinas de Turing determinísticas pois a classe de linguagens reconhecidas não muda ao introduzirmos não-determinismo, como pode ser observado em (Sipser 2006).

Definição 3.2.1. Seja C um computador. O **núcleo** de C , denotado por $N(C)$, será o subconjunto de $R(C)$ dado por

$$N(C) = \{n \in R(C) : \exists k \in \mathbb{N}, C^k[n] \in F(C)\}. \quad (3.12)$$

Em outras palavras, se trata do conjunto de raízes de C nas quais há uma iteração finita que atinge uma folha.

Nota. Este comportamento de iteração de raiz para uma folha é análogo à computação de uma entrada em uma máquina de Turing (especificamente, ao comportamento das configurações induzida pela função de transição). Se formos capazes de traduzir qualquer palavra e montar uma função que nos garanta esta iteração, temos o resultado esperado nesta seção. Faremos isso adiante.

Lema 3.2.2. Seja $A \neq \emptyset$ um alfabeto finito e A^* seu fecho, isto é, o conjunto de todas as palavras formadas pelos símbolos de A . Existe uma bijeção entre A^* e $\mathbb{N}$.

Demonstração. Note que A^* é o conjunto de todas as palavras de tamanho finito (incluindo tamanho zero). Em particular, podemos escrever A^* como

$$A^* = \bigcup_{t \in \mathbb{N}} A^t \quad (3.13)$$

onde A^t é o conjunto de todas as palavras de tamanho t . Como A é finito, qualquer A^t será também finito, e em particular enumerável. Como estamos fazendo uma união enumerável de conjuntos enumeráveis, sabemos pela seção 2.1.2 que A^* é enumerável. Por definição, um conjunto é enumerável se, e somente se, existe uma bijeção deste conjunto aos naturais. Temos então a garantia da bijeção entre A^* e $\mathbb{N}$. ■

Nota. O resultado acima é uma aplicação direta do que foi mencionado na parte de fundamentação teórica acerca teoria de conjuntos.

Apesar de não construirmos explicitamente a bijeção, aproveitaremos sua existência e usaremos dela para ‘traduzir’ as máquinas de Turing.

Proposição 3.2.3. Seja $T = (Q, \Sigma, \Gamma, q_0, \delta, H)$ uma máquina de Turing determinística e $L(T)$ sua linguagem. Existe um computador C tal que há uma bijeção $\phi : L(T) \rightarrow N(C)$.

Demonstração. Iremos observar o comportamento das configurações de T quando dadas as entradas de $L(T)$. Note que podemos escrever estas configurações como uma palavra do alfabeto $Q \cup \Sigma \cup \Gamma$, a saber, colocando o estado q_x logo à esquerda do símbolo lido pelo cabeçote, e o conteúdo da fita ao redor de q_x (esta é a forma que o [Sipser 2006](#) escreve, por exemplo. Ver páginas 140, 141 e 142).

O histórico de computação de uma máquina de Turing, para $w \in L(T)$, $u_i, v_i \in (\Sigma \cup \Gamma)^*$ e $q_i \in Q$, é algo da forma

$$q_0 w \vdash u_1 q_{x_1} v_1 \vdash u_2 q_{x_2} v_2 \vdash \dots \vdash u_k q_{x_k} v_k \quad (3.14)$$

onde $q_{x_k} \in H$ é um estado de parada. Utilizando do lema anterior [\(3.2.2\)](#), podemos assumir uma bijeção entre o conjunto de configurações de T , escritas como palavras no alfabeto $Q \cup \Sigma \cup \Gamma$, e os naturais. Seja ξ esta bijeção. Note que $L(T) \subseteq (Q \cup \Sigma \cup \Gamma)^*$, uma vez que $\Sigma^* \subseteq (Q \cup \Sigma \cup \Gamma)^*$, e portanto ξ também enumera as palavras da linguagem de T .

Para cada $w \in L(T)$, defina f_w como

$$\begin{aligned} f_w(\xi(w)) &= \xi(q_0 w) \\ f_w(\xi(q_0 w)) &= \xi(u_1 q_{x_1} v_1) \\ f_w(\xi(u_i q_{x_i} v_i)) &= \xi(u_{i+1} q_{x_{i+1}} v_{i+1}) \end{aligned} \quad (3.15)$$

observando que f_w está bem definida como função em algum subconjunto X_w dos naturais, pois T é determinística e assim cada configuração terá apenas uma próxima configuração no histórico. Temos que $1 \leq i < k$.

Desta forma, obtemos uma família de funções $\{f_w\}_{w \in L(T)}$, uma para cada palavra $w \in L(T)$. Usaremos esta família de funções para definir uma única função, a qual usaremos como função do computador.

Primeiramente, observamos que se $u, v \in L(T)$ e $k \in X_u \cap X_v$, então temos que $f_u(k) = f_v(k)$. Isso pois a máquina de Turing em questão é determinística, e nossa escolha de k nos diz que $\xi^{-1}(k)$ é alguma configuração, que por sua vez terá apenas uma próxima configuração resultante da transição da máquina de Turing. Assim, não importa se $\xi^{-1}(k)$ é atingida no histórico de aceitação de u ou de v , pois neste ponto as funções f_u e f_v devem concordar.

O argumento anterior nos diz que a função definida em

$$F : \bigcup_{w \in L(T)} X_w \rightarrow \mathbb{N} \quad (3.16)$$

dada por

$$F(n) = f_w(n), \forall w \in L(T) : n \in X_w \quad (3.17)$$

está bem definida, pois vimos que as funções f_w com $w \in L(T)$ devem concordar nas enumerações em comum, isto é, concordam em todos os $k \in X_{w_1} \cap X_{w_2} \cap \dots$ quando $\xi^{-1}(k)$ é alguma configuração que aparece no histórico de computação de $w_1, w_2, \dots \in L(T)$ (note que $\xi^{-1}(k)$ pode ser a própria palavra).

Obtemos então um computador $C = \{F\}$. Resta observarmos que existe uma bijeção entre $N(C)$ e $L(T)$. Da nossa construção da família de funções $\{f_w\}_{w \in L(T)}$, obtemos de imediato que $\xi(w) \in D(C)$ para todo $w \in L(T)$. Além disso, nessa mesma construção observamos que nenhuma palavra de $L(T)$ aparece como imagem se não for uma palavra no alfabeto $Q \cup \Sigma \cup \Gamma$, e portanto $\xi(w) \notin I(C)$. Como toda outra configuração utilizada aparece ou como imagem da palavra ou como imagem de outra configuração, temos que $\xi(L(T)) = R(C)$. Um argumento similar nos diz que as folhas de C se tratam dos números, por ξ , das configurações escritas com um símbolo $q_h \in H$, ou seja, a enumeração das configurações de parada (rigorosamente, de suas escritas no alfabeto $Q \cup \Sigma \cup \Gamma$).

Diretamente das nossas construções de f_w observamos que toda raiz também acaba, por definição, sendo parte do núcleo, pois a função percorre, por f_w (e todas as demais f_v que concordarem), precisamente os números por ξ das configurações intermediárias, até atingir a de aceitação.

Finalmente temos que $R(C) = N(C)$ e assim $\xi(L(T)) = N(C)$. Como ξ é uma bijeção, a restrição de ξ dada por

$$\begin{aligned} \phi : L(T) &\rightarrow N(C) \\ w &\mapsto \xi(w) \end{aligned} \tag{3.18}$$

é também uma bijeção. ■

Nota. A demonstração acima visa conectar o modelo aqui apresentado com as máquinas de Turing para que haja conformidade com a tese de Church–Turing. Esta conformidade é importante pois justifica as próximas seções.

Citamos duas referências principais para máquinas de Turing e teoria da computação no geral. São elas (Sipser 2006) e (Lewis e Papadimitriou 1998). Em particular, nossa forma de escrever as configurações vem de (Sipser 2006), tal como nosso uso do nome *linguagem recursivamente enumerável*. A definição de máquina de Turing também é basicamente extraída de (Sipser 2006).

3.3 Não-determinismo

Esta seção irá discutir alguns pontos acerca do não-determinismo. Não será tão formal nem rigorosa quanto as demais seções, e funcionará como o único tratamento do tema neste trabalho.

Retomamos nossa definição de não-determinismo. Se um computador C for tal que existam duas (ou mais) funções $f, g \in C$ tais que $D_f \cap D_g \neq \emptyset$ (isto é, a interseção de seus domínios for não vazia), e se $n \in D_f \cap D_g$ for tal que $f(n) \neq g(n)$, temos um caso de não-determinismo para o computador como definido aqui neste trabalho. Esse não-determinismo torna mal definida nossa notação $C[n]$ para um passo de computação.

A principal observação é que podemos reintroduzir o determinismo da seguinte forma: dado um computador não-determinístico C , montaremos um computador $\bar{C}$, com funções em $\mathcal{P}(\mathbb{N})$ (no conjunto das partes de $\mathbb{N}$, ou seja, funções que levam subconjuntos de $\mathbb{N}$ em subconjuntos de $\mathbb{N}$) que simula C . Este processo é análogo à determinização feita em outros modelos de computação.

O principal ingrediente é uma re-interpretação de $C[n]$. No caso de $\bar{C}$, temos que $\bar{C}[S] = \{f(n) : n \in S \wedge f \in C \wedge n \in D_f\}$. Note que estamos considerando todas as funções para as quais n está no domínio (do ponto de vista de C , o computador nos naturais), aplicando-as e obtendo então um novo conjunto. Se n for o único

elemento de S e também for um ponto determinístico em C , podemos escrever por extenso

$$\bar{C}[\{n\}] = \{C[n]\}. \quad (3.19)$$

E nesse sentido, observamos que $\bar{C}$ age como uma “cópia” de C .

Neste ponto, poderíamos então considerar a dinâmica dos computadores $\bar{C}$ definidos em $\mathcal{P}(\mathbb{N})$. Novamente, o caso não-determinístico para esse tipo de computador pode ser determinizado considerando $\mathcal{P}(\mathcal{P}(\mathbb{N}))$. Esta escalada pode ser repetida à vontade.

4 DINÂMICA COMPUTACIONAL

Tendo definido elementos de nossos computadores (3.1), e demonstrada sua potencial capacidade expressiva (3.2.3), partimos primeiro para um estudo da dinâmica interna de um dado computador, isto é, de como a iteração se comporta. Ressaltamos estar apenas tratando do caso determinístico.

4.1 Órbitas e Componentes

Definição 4.1.1. Seja C um computador e $n \in E(C)$. A **órbita** de n será o conjunto

$$\theta(n, C) = \{m \in E(C) : \exists p \in \mathbb{N}, C^p[n] = m\}. \quad (4.1)$$

A **pré-órbita** de n será

$$V(n, C) = \{m \in E(C) : \exists p \in \mathbb{N}, C^p[m] = n\}. \quad (4.2)$$

Em outras palavras, a órbita de um número n é o conjunto de todos os números m que são atingidos através iterações iniciadas em n . A pré-órbita são todos os números m que atingem n através de iterações de C , iniciadas em m .

Nota. As propriedades abaixo seguem diretamente da definição acima e portanto suas demonstrações foram omitidas.

Proposição 4.1.2. Para um computador C e $n \in E(C)$, valem:

1. $n \in F(C) \implies \theta(n, C) = \{n\}$;
2. $n \in R(C) \implies V(n, C) = \{n\}$;
3. $n \in N(C) \iff n \in R(C) \text{ e } \theta(n, C) \cap F(C) \neq \emptyset$;
4. Para todo $m \in E(C)$, temos que $m \in \theta(n, C) \iff n \in V(m, C)$;

□

Nota. A proposição abaixo nos garante que as órbitas não têm buracos, ou seja, se atingimos a p -ésima iteração de um n , a órbita tem todas as k -ésimas iterações anteriores, para $k \in \{0, 1, \dots, p-1\}$;

Proposição 4.1.3. Seja C um computador e $n \in E(C)$. Se $C^p[n] \in \theta(n, C)$, então para todo $k \in \{0, 1, \dots, p-1\}$, temos $C^k[n] \in \theta(n, C)$;

Demonstração. Faremos indução em p . Para o caso base $p = 0$, temos que $C^p[n] = C^0[n] = n \in \theta(n, C)$. Suponha que, para p qualquer, vale

$$C^p[n] \in \theta(n, C) \implies C^k[n] \in \theta(n, C), \forall k \in \{1, \dots, p-1\} \quad (4.3)$$

e assumamos ainda que $C^{p+1}[n] \in \theta(n, C)$. Como $C^{p+1}[n] = C[C^p[n]]$ temos que existe $C^p[n] \in \theta(n, C)$. Portanto, $C^k[n] \in \theta(n, C)$ para todo $k \in \{0, 1, \dots, p-1\}$. Disso e de $C^p[n] \in \theta(n, C)$ temos que $C^k[n] \in \theta(n, C)$ para todo $\{0, 1, \dots, p\}$. ■

Proposição 4.1.4. Se C é um computador e $n \in E(C)$ for tal que existe $x \in E(C)$ com $x \neq n$ mas $x \in V(n, C) \cap \theta(n, C)$ então n e x são pontos de um mesmo ciclo.

Demonstração. Como $x \in V(n, C) \cap \theta(n, C)$ e $x \neq n$, obtemos $p, q \in \mathbb{N}$ tais que $C^p[x] = n$ e $C^q[n] = x$. Desta forma,

$$x = C^q[n] = C^q[C^p[x]] = C^{p+q}[x] \quad (4.4)$$

e portanto x tem período divisor de $p+q$, estando portanto em um ciclo. Analogamente,

$$n = C^p[x] = C^p[C^q[n]] = C^{p+q}[n]. \quad (4.5)$$

■

Proposição 4.1.5. Dados C computador e $n, m \in E(C)$ tais que nenhum destes estão num mesmo ciclo, são equivalentes:

1. $m = n$;
2. $\theta(n, C) = \theta(m, C)$; e
3. $V(n, C) = V(m, C)$.

Demonstração. As implicações $(1) \implies (2)$ e $(1) \implies (3)$ são imediatas. Veremos então $(2) \implies (1)$ e $(3) \implies (1)$.

Para $(2) \implies (1)$, temos que $n \in \theta(n, C) = \theta(m, C) \implies n = C^k[m]$ para algum $k \in \mathbb{N}$. Assim, $m \in V(n, C)$ e portanto $m \in V(n, C) \cap \theta(n, C)$. Vejamos que $V(n, C) \cap \theta(n, C) = \{n\}$. De fato, se existir $x \in V(n, C) \cap \theta(n, C)$ diferente de n , a Proposição 4.1.4 nos diz que n e x são pontos de um mesmo ciclo, contrariando nossa hipótese. Portanto, $V(n, C) \cap \theta(n, C) = \{n\}$. Daí $m \in \{n\} \implies m = n$.

Para $(3) \implies (1)$, $n \in V(n, C) = V(m, C)$ nos diz que $C^k[n] = m$. Desta forma, $m \in \theta(n, C)$ e o resto da demonstração é análogo ao caso anterior. ■

Proposição 4.1.6. Sejam C um computador e $n, m \in E(C)$. Valem:

1. $m \in \theta(n, C) \iff \theta(m, C) \subseteq \theta(n, C)$; e
2. $m \in V(n, C) \iff V(m, C) \subseteq V(n, C)$.

Demonstração. Para (1), se $m \in \theta(n, C)$, obtemos $p \in \mathbb{N}$ tal que $C^p[n] = m$. Tome $x \in \theta(m, C)$ e $q \in \mathbb{N}$ tal que $C^q[m] = x$. Assim, $x = C^q[m] = C^q[C^p[n]] = C^{p+q}[n]$ e logo $x \in \theta(n, C)$. Portanto, $\theta(m, C) \subseteq \theta(n, C)$. Reciprocamente, se $\theta(m, C) \subseteq \theta(n, C)$, então $m \in \theta(m, C) \subseteq \theta(n, C)$ e temos $m \in \theta(n, C)$.

Para (2), se $m \in V(n, C)$, obtemos p tal que $C^p[m] = n$. Tome $x \in V(m, C)$ e separe q tal que $C^q[x] = m$. Temos então que $C^p[C^q[x]] = C^p[m] = n$ e assim $C^{p+q}[x] = n$, de onde obtemos $x \in V(n, C)$. Como escolhemos x arbitrário, $V(m, C) \subseteq V(n, C)$. Se $V(m, C) \subseteq V(n, C)$, temos analogamente ao caso anterior que $m \in V(n, C)$. ■

Definição 4.1.7. Sejam C um computador e $n \in E(C)$. Ao conjunto

$$\Delta(n, C) = \bigcup_{k \in \theta(n, C)} V(k, C) \quad (4.6)$$

daremos o nome de **componente** de n por C . Quando o computador em questão estiver claro, chamaremos apenas de componente de n .

Proposição 4.1.8. Sejam C um computador e n, m folhas de C . Temos que se $\Delta(n, C) = \Delta(m, C)$ então $m = n$.

Demonstração. Primeiramente, observamos que se $x \in F(C)$ então

$$\Delta(x, C) = V(x, C). \quad (4.7)$$

Desta forma, como $V(n, C) = V(m, C)$. Como n e m são folhas, não estão em um mesmo ciclo, e da Proposição 4.1.5 temos que $n = m$. ■

Proposição 4.1.9. Seja C um computador e $(x_1, \dots, x_p), (y_1, \dots, y_q)$ ciclos distintos (e disjuntos) de C . Temos que

$$\Delta(x_1, C) = \Delta(x_2, C) = \dots = \Delta(x_p, C) \quad (4.8)$$

e que

$$\Delta(x_1, C) \cap \Delta(y_1, C) = \emptyset. \quad (4.9)$$

Demonstração. Para a primeira série de igualdades, tome $i, j \in \{1, \dots, p\}$ e observe que $\theta(x_i, C) = \theta(x_j, C) = \{x_1, \dots, x_p\}$. Disso, temos que $\Delta(x_i, C) = V(x_1, C) \cup V(x_2, C) \cup \dots \cup V(x_p, C) = \Delta(x_j, C)$.

Agora suponha que existe $k \in \Delta(x_1, C) \cap \Delta(y_1, C)$. Daí, $k \in V(x_i, C)$ para algum $i \leq p$ e, analogamente, $k \in V(y_j, C)$ para algum $j \in \{1, \dots, q\}$. Obtemos então $r, s \in \mathbb{N}$ tais que $C^r[k] = x_i$ e $C^s[k] = y_j$. Temos três opções: $r = s$, $r < s$ ou $s < r$.

No caso $r = s$, o determinismo de C nos garante $x_i = C^r[k] = C^s[k] = y_j$. Ainda pelo determinismo, sabemos que $C[x_i] = C[y_j]$ e, seguindo por este raciocínio, vemos que os ciclos devem ser iguais, contradizendo nossa hipótese.

No caso $r < s$, teríamos que $0 < s - r$ e que

$$C^{r+(s-r)}[k] = C^{s-r}[C^r[k]] = C^{s-r}[x_i] = x_{i_1} \quad (4.10)$$

para algum $i_1 \in \{1, \dots, p\}$ uma vez que x_i está em um ciclo. No entanto, $C^{r+s-r}[k] = C^s[k] = y_j$ e assim obtemos

$$y_j = C^s[k] = C^{r+s-r}[k] = C^{s-r}[C^r[k]] = x_{i_1} \quad (4.11)$$

de onde o determinismo nos diz que os ciclos devem ser o mesmo, novamente contrariando nossa hipótese. O caso $s < r$ é análogo ao caso $r < s$. Portanto, não deve existir $k \in V(x_1, C) \cap V(y_1, C)$, ou seja, $\Delta(x_1, C) \cap \Delta(y_1, C) = \emptyset$. ■

Nota. A definição acima descreve um conjunto que, na visualização por grafos funcionais dos computadores, se trata de um componente conexo a n . No entanto, não é fácil observar a conexidade dos nós neste conjunto, então usaremos para isto uma relação de equivalência.

Definição 4.1.10. Seja C um computador e $n, m \in E(C)$. Diremos que n é **conexo** a m se

$$\exists k \in E(C) : k \in \theta(n, C) \cap \theta(m, C) \quad (4.12)$$

e denotaremos por $n \equiv m$.

Nota. Da Proposição 4.1.6 obtemos que $\theta(k, C) \subseteq \theta(n, C) \cap \theta(m, C)$.

Proposição 4.1.11. Conexidade, como definida acima, é uma relação de equivalência.

Demonstração. Reflexividade é imediata, pois por definição $n \in \theta(n, C)$. Se $n \equiv m$, o mesmo k nos garante que $m \equiv n$, e então temos a simetria.

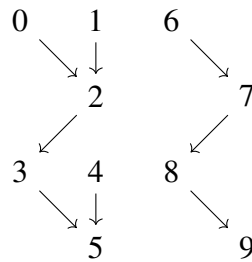
Para a transitividade, tome $n, m, p \in E(C)$ tais que $n \equiv m$ e $m \equiv p$. Separe k_1 que faz $n \equiv m$ e k_2 que faz $m \equiv p$. Como $k_1, k_2 \in \theta(m, C)$, temos três opções:

1. $k_1 = k_2$;
2. $k_1 \in \theta(k_2, C)$; ou
3. $k_2 \in \theta(k_1, C)$.

O caso (1) nos garante $n \equiv p$ de imediato. Os casos (2) e (3) são análogos então veremos apenas um deles. Se $k_1 \in \theta(k_2, C)$, como $\theta(k_2, C) \subseteq \theta(p, C)$ temos que $k_1 \in \theta(n, C) \cap \theta(p, C)$ e daí temos $n \equiv p$. ■

Exemplo 4.1.12. O nome conexidade não é a toa. Usamos esta relação para capturar, via a dinâmica, a conexidade de grafos. Ilustramos que de fato a definição aqui nos diz que n e m estão na mesma componente conexa dos grafos funcionais possíveis para nossos computadores.

Considere um computador simples, digamos o exibido abaixo (as funções específicas estão novamente omitidas):



Observamos que $4 \equiv 0$ pois $5 \in \theta(4, C) \cap \theta(0, C)$. Vemos também que $6 \not\equiv 0$.

Proposição 4.1.13. As classes de equivalência induzidas pela conexidade são precisamente as componentes dos $n \in E(C)$.

Demonstração. Seja $\bar{n}$ a classe de equivalência de $n \in E(C)$. Tome $x \in \bar{n}$ e separe k tal que $x \equiv n$. Como $k \in \theta(x, C) \cap \theta(n, C)$, em particular temos $k \in \theta(n, C)$ e $x \in V(k, C)$, de onde obtemos $x \in \Delta(n, C)$ pela definição de componente. Logo, $\bar{n} \subseteq \Delta(n, C)$.

Tomando $x \in \Delta(n, C)$, então $x \in V(r, C)$ para algum $r \in \theta(n, C)$. Desta forma, $r \in \theta(n, C) \cap \theta(x, C)$. Logo $x \equiv n$, ou seja, $x \in \bar{n}$ e temos $\Delta(n, C) \subseteq \bar{n}$. Finalmente, $\bar{n} = \Delta(n, C)$. ■

Nota. A proposição anterior nos garante que as componentes que definimos particionam o espaço, pois corresponde às classes de equivalência da conexidade. Neste sentido, estaremos satisfeitos em dizer que podemos observar, num dado computador C , uma combinação de componentes conexas que então formam seu grafo funcional.

Proposição 4.1.14. Seja C um computador, $n \in E(C)$ e $\Delta(n, C)$ sua componente. Vale uma e apenas uma das condições a seguir:

1. Para todo $x \in \Delta(n, C)$, existe uma única folha $p \in F(C)$ tal que $p \in \theta(x, C)$;
2. Para todo $x \in \Delta(n, C)$, existe um único ciclo $(y_1, \dots, y_q)$ tal que $y_1, \dots, y_q \in \theta(x, C)$; ou
3. Para todo $x \in \Delta(n, C)$, temos que $\theta(x, C)$ é infinita;

Demonstração. Temos duas opções para $\theta(n, C)$: ou é finita, ou é infinita. Suponha que $\theta(n, C)$ é infinita e seja $x \in \Delta(n, C)$. Obtemos $k \in E(C)$ que torna $n \equiv x$. Sabemos também que $k \in \Delta(n, C)$ e $k \in \Delta(x, C)$. Como $\theta(n, C)$ é infinita, $\theta(k, C)$ também deve ser.

Assuma então que $\theta(x, C)$ é finita. Temos duas opções: $\theta(x, C) \cap F(C)$ é vazia ou é não-vazia.

Se $\theta(x, C) \cap F(C)$ for vazia, sabemos que $C^k[x]$ deve estar definida para todo $k \in \mathbb{N}$. Assim, tome $k_1 > |\theta(x, C)|$. Pelo princípio da casa dos pombos, devemos ter que $C^{k_1}[x] \in \theta(x, C)$ deve ser igual a algum $C^{k_0}[x]$ para $k_0 < k_1$. Assim, temos um período e portanto um ciclo. A proposição 4.1.9 nos garante que este ciclo é único.

Se $\theta(x, C) \cap F(C)$ for não-vazia, temos (1) de imediato, e a proposição 4.1.8 nos diz que esta folha é única.

Vejam agora que apenas uma das condições ocorre por vez. De imediato, temos que (1) $\implies \neg(3)$ e (2) $\implies \neg(3)$, que pela contrapositiva nos diz que (3) $\implies \neg(2)$ e (3) $\implies \neg(1)$.

Vejam os que $(1) \implies \neg(2)$. De 4.1.8 sabemos que $\Delta(p, C) = V(p, C)$. De 4.1.13, sabemos também que $\Delta(p, C) = \Delta(x, C)$. Daí, $\Delta(x, C) = V(p, C)$. Suponha que existe um ciclo $y_1, \dots, y_q$ em $\Delta(x, C)$. Assim, devemos ter $k_i \in \mathbb{N}$ tal que $C^{k_i}[y_i] = p$ para $i \leq q$. Isso contradiz ou o fato de p ser folha ou o fato de $(y_i)_q$ ser um ciclo, e portanto não podem existir ciclos em $\Delta(x, C)$. A contrapositiva desta condição nos dá $(2) \implies \neg(1)$. ■

Nota. A proposição anterior nos dá uma certa classificação das componentes conexas que podemos ter num grafo de um computador C qualquer. Em outras palavras, todo computador C tem por grafo funcional uma combinação de uma ou mais componentes conexas de um dos três tipos acima. Gostaríamos de poder contar estas componentes.

Corolário 4.1.15. Seja C um computador e N o número de ciclos distintos em C . O número de componentes conexas do grafo de C é no mínimo $|F(C)| + N$.

Demonstração. Temos isto diretamente das Proposições 4.1.9, 4.1.8, e 4.1.14, pois toda componente conexa deve ser de um dos três tipos, e folhas e ciclos estão em componentes distintas. Note que isto também funciona para o caso infinito: se temos infinitas folhas ou ciclos em C , temos no mínimo infinitas componentes conexas. ■

Nota. Como sabemos que o comportamento orbital de componentes conexas é apenas um de três tipos, iremos dar nome à estes tipos.

Definição 4.1.16. Seja C um computador e $n \in E(C)$. Se, para $\Delta(n, C)$, vale a condição (1) de 4.1.14, então diremos que $\Delta(n, C)$ é de **tipo 1**. Desta mesma forma, definimos os tipos 2 e 3.

5 SIMULAÇÃO

Tendo explorado um pouco o funcionamento interno dos computadores, iremos estudar formas de compará-los. De agora em diante, estaremos lidando usualmente com dois ou mais computadores. Denotaremos estes computadores por C e K .

A próxima definição inaugura uma ferramenta que iremos utilizar amplamente no estudo dos elementos algébricos dos computadores.

5.1 Simulação Computacional

Definição 5.1.1. Sejam C e K computadores. Uma função $\phi : E(C) \rightarrow E(K)$ tal que para todo $n \in D(C)$ vale

$$\phi(C[n]) = K[\phi(n)] \quad (5.1)$$

será chamada de **simulação** de C em K .

Exemplo 5.1.2. Seja X um computador tal que $E(X) = \{1\}$ e $X[1] = 1$, ou seja, um computador cuja única computação é permanecer no ponto fixo 1. Qualquer outro computador C pode ser simulado em X fazendo simplesmente $\phi(n) = 1$ para todo $n \in E(C)$.

Exemplo 5.1.3. Iremos exibir dois computadores C e K por seus grafos. As funções específicas de C e K não são importantes para o exemplo e estão omitidas. Seja C da forma

$$4 \longrightarrow 5$$

e K da forma

$$0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 3$$

e considere a função $\phi : E(C) \rightarrow E(K)$ dada por

$$\begin{aligned}\phi(4) &= 2, \\ \phi(5) &= 3.\end{aligned}\tag{5.2}$$

Verificamos que ϕ é uma simulação de C em K pois

$$\phi(C[4]) = \phi(5) = 3 = K[2] = K[\phi(4)].\tag{5.3}$$

Proposição 5.1.4. Sejam C e K computadores, $p \in \mathbb{N}$ e ϕ uma simulação de C em K . Para todo $n \in D(C)$, se $C^p[n] \in E(C)$ então temos que

$$\phi(C^p[n]) = K^p[\phi(n)].\tag{5.4}$$

Demonstração. Tome $n \in D(C)$. Faremos por indução em p . O caso base de $p = 0$ é imediato: $\phi(C^0[n]) = \phi(n) = K^0[\phi(n)]$.

Se vale para p , temos que

$$\phi(C^{p+1}[n]) = \phi(C[C^p[n]]) = K[\phi(C^p[n])] = K[K^p[\phi(n)]] = K^{p+1}[\phi(n)]\tag{5.5}$$

e concluímos a indução. Como escolhemos n genérico, temos que vale para qualquer $n \in D(C)$. ■

Proposição 5.1.5. Sejam C , K e R computadores e $\phi : E(C) \rightarrow E(K)$ e $\psi : E(K) \rightarrow E(R)$ simulações. Temos que $\psi \circ \phi : E(C) \rightarrow E(R)$ é também uma simulação.

Demonstração. Tome $n \in D(C)$, temos

$$\psi \circ \phi(C[n]) = \psi(\phi(C[n])) = \psi(K[\phi(n)]) = R[\psi(\phi(n))] = R[\psi \circ \phi(n)].\tag{5.6}$$

■

Proposição 5.1.6. Sejam C e K computadores e ϕ uma simulação de C em K . Temos que $\phi^{-1}(R(K)) \subseteq R(C)$.

Demonstração. Seja $x \in \phi^{-1}(R(K)) \iff \phi(x) \in R(K)$. Se $x \notin R(C)$, então existe algum $n \in D(C)$ tal que $C[n] = x$. Como ϕ é simulação, temos que $\phi(C[n]) = K[\phi(n)]$. Assim,

$$\phi(x) = \phi(C[n]) = K[\phi(n)]\tag{5.7}$$

o que contradiz nossa escolha de x tal que $\phi(x) \in R(K)$. Deve ser o caso então que $x \in R(C)$, e assim $\phi^{-1}(R(K)) \subseteq R(C)$. ■

Nota. Em outras palavras, a proposição acima nos diz que as raízes de K são imagens de uma ou mais raízes de C , quando de fato forem imagens de alguém pela simulação.

Proposição 5.1.7. Sejam C e K computadores e ϕ uma simulação. Se $(x_1, \dots, x_n)$ é um ciclo em C (com $n > 0$), então $(\phi(x_1), \dots, \phi(x_n))$ é um ciclo em K .

Demonstração. Como $(x_1, \dots, x_n)$ é um ciclo, temos que $C[x_i] = x_{i+1}$ para $i < n$ e que $C[x_n] = x_1$. Da definição de simulação, obtemos

$$\phi(x_{i+1}) = \phi(C[x_i]) = K[\phi(x_i)], \forall i \in \{1, \dots, n-1\} \quad (5.8)$$

e também

$$\phi(x_1) = \phi(C[x_n]) = K[\phi(x_n)] \quad (5.9)$$

ou seja,

$$\begin{aligned} K[\phi(x_1)] &= \phi(x_2) \\ K[\phi(x_2)] &= \phi(x_3) \\ &\vdots \\ K[\phi(x_n)] &= \phi(x_1) \end{aligned} \quad (5.10)$$

de onde vemos que $(\phi(x_1), \dots, \phi(x_n))$ é também um ciclo. ■

Lema 5.1.8. Seja C um computador e $x \in E(C)$ tal que x tem período p . Temos que para todo $s \in \mathbb{N}$, $C^{ps}[x] = x$.

Demonstração. Faremos indução em s . Se $s = 0$, temos $C^0[x] = x$. Suponha que vale para s . No caso $s + 1$ temos

$$C^{p(s+1)}[x] = C^{ps+p}[x] = C^{ps}[C^p[x]] = C^{ps}[x] = x. \quad (5.11)$$

■

Corolário 5.1.9. Se $x_1 \in E(C)$ tem período p e $\phi(x_1) \in E(K)$ tem período q , então q divide p .

Demonstração. Se $q = p$, a divisibilidade é imediata. Se $q > p$ temos que $p + r = q$ para algum $0 < r < q$, e daí

$$\phi(x_1) = K^q[\phi(x_1)] = \phi(C^q[x_1]) = \phi(C^r[C^p[x_1]]) = \phi(C^r[x_1]) = K^r[\phi(x_1)] \quad (5.12)$$

que contradiz nossa hipótese de que q é período de $\phi(x_1)$, ou seja, não pode ocorrer de $q > p$. Resta então $q < p$. Pelo algoritmo Euclidiano da divisão, obtemos $p = qs + t$ para $s > 0$ e $0 \leq t < q$. Disso,

$$\phi(x_1) = \phi(C^p[x_1]) = \phi(C^{qs+t}[x_1]) = \phi(C^t[C^{qs}[x_1]]) = K^t[\phi(x_1)] \quad (5.13)$$

de onde concluímos que $t = 0$ e assim $p = qs$, ou seja, q divide p . ■

Proposição 5.1.10. Sejam C e K computadores, ϕ uma simulação de C em K e $n \in E(C)$. Valem:

1. $\phi(\theta(n, C)) \subseteq \theta(\phi(n), K)$;
2. $\phi(V(n, C)) \subseteq V(\phi(n), K)$; e
3. $\phi(\Delta(n, C)) \subseteq \Delta(\phi(n), K)$.

Demonstração. A propriedade (1) decorre diretamente de 5.1.4. Tome $\phi(x) \in \phi(\theta(n, C))$, ou seja, $x \in \theta(n, C)$. Disso, obtemos p tal que $C^p[n] = x$. Como ϕ é simulação,

$$\phi(x) = \phi(C^p[n]) = K^p[\phi(n)] \quad (5.14)$$

de onde obtemos que $\phi(x) \in \theta(\phi(n), K)$. Assim, $\phi(\theta(n, C)) \subseteq \theta(\phi(n), K)$.

Para (2), tome $\phi(x) \in \phi(V(n, C))$ para $x \in V(n, C)$. Obtemos $p \in \mathbb{N}$ tal que $C^p[x] = n$. Daí,

$$\phi(n) = \phi(C^p[x]) = K^p[\phi(x)] \quad (5.15)$$

e portanto $\phi(x) \in V(\phi(n), K)$.

Para (3), tome $\phi(x) \in \phi(\Delta(n, C))$. Usaremos o fato de que a componente é a classe de equivalência de n . Assim, obtemos $k \in \theta(x, C) \cap \theta(n, C)$. Do visto em (1), temos que $\phi(k) \in \theta(\phi(x), K) \cap \theta(\phi(n), K)$, e portanto $\phi(x) \in \Delta(\phi(n), K)$. ■

Nota. Relembramos que sendo C um computador e $n \in E(C)$, $\bar{n}$ denota a classe de equivalência de n pela relação de conexidade. Isso será importante para a próxima definição.

Definição 5.1.11. Ao conjunto

$$\bar{C} = \{\bar{n} : n \in E(C)\} \quad (5.16)$$

daremos o nome de **componentes conexas** de C .

Lema 5.1.12. Seja C um computador. Para cada $S \in \bar{C}$ existe um computador Z tal que $Z[s] = C[s]$ para todo $s \in S \cap D(C)$.

Demonstração. Se restringimos as funções de C para o domínio $S \cap D(C)$, obtemos as funções do computador Z . ■

Nota. O que se espera de fato do lema anterior é de que percebamos que as componentes conexas de computador podem ser entendidas também como computadores. Dessa forma, podemos falar de simulação entre componentes. Abusando da notação, serão confundidas as componentes $S \in \bar{C}$ com o computador obtido pela construção do lema acima.

Proposição 5.1.13. Sejam C e K computadores. Temos que existe uma simulação $\phi : E(C) \rightarrow E(K)$ se, e somente se, para todo $S \in \bar{C}$ existe $B_S \in \Delta(K)$ com uma simulação $\psi_S : E(S) \rightarrow E(B)$.

Demonstração. ($\implies$) Seja $\phi : E(C) \rightarrow E(K)$ uma simulação, e tome $n_0 \in E(C)$. Sabemos de 5.1.10 que $\phi(\Delta(n_0, C)) \subseteq \Delta(\phi(n_0), K)$. Faça $S = \Delta(n_0, C)$ e defina

$$\begin{aligned} \psi_S : S &\rightarrow \Delta(\phi(n_0), K) \\ s &\mapsto \phi(s). \end{aligned} \quad (5.17)$$

O fato de ψ_S ser uma simulação decorre do fato de ϕ ser uma simulação de C em K . Para $x \in S \cap D(C)$, temos do lema anterior que

$$\psi_S(S[x]) = \psi_S(C[x]) \quad (5.18)$$

e assim

$$\psi_S(C[x]) = \phi(C[x]) = K[\phi(x)] \quad (5.19)$$

e então novamente de 5.1.10 temos que $K[\phi(x)] \in \Delta(\phi(n_0), K)$.

($\impliedby$) Suponha que para todo $S \in \bar{C}$ existe $Z \in \Delta(K)$ onde temos uma simulação $\psi_S : E(S) \rightarrow E(Z)$. Como

$$\bigcup_{S \in \bar{C}} S = E(C) \quad (5.20)$$

e $S_1 \cap S_2 = \emptyset$ para todos $S_1, S_2 \in \bar{C}$, a função dada por

$$\phi(x) = \psi_S(x), \forall x \in E(S) \quad (5.21)$$

está bem definida para todo $x \in E(C)$. Vejamos que ϕ é simulação de C em K . Tome $n_0 \in E(C)$. Sabemos que $n_0 \in S$ para algum $S \in \bar{C}$. Temos

$$\phi(C[n_0]) = \phi(S[n_0]) = \psi_S(S[n_0]) = Z[\psi_S(n_0)] = Z[\phi(n_0)] = K[\phi(n_0)] \quad (5.22)$$

para algum $Z \in \Delta(K)$. Assim, ϕ é simulação de C em K . ■

Proposição 5.1.14. Seja C um computador e $S \in \bar{C}$ uma componente de tipo 3. Podemos simular S em uma componente de tipo 2.

Demonstração. Iremos construir um computador B com apenas uma componente e apresentar uma simulação de $S \in \bar{C}$ em B . Faremos $D(B) = \{0, 1\}$ e $I(B) = \{1\}$. A única função de B será

$$b(n) = \begin{cases} 1, & n = 0 \\ 1, & n = 1. \end{cases} \quad (5.23)$$

B tem raiz em 0 e ponto fixo em 1. A função $\phi : E(S) \rightarrow E(B)$ dada por

$$\phi(x) = \begin{cases} 0, & x \in R(S) \\ 1, & x \notin R(S) \end{cases} \quad (5.24)$$

simula S em B . De fato, tome $x \in E(S)$. Temos duas opções: $x \in R(S)$ ou $x \notin R(S)$.

Se $x \in R(S)$, temos que $\phi(x) = 0$. Como S é do tipo 3, deve existir $S[x] \in E(S)$ com $S[x] \notin R(S)$. Portanto,

$$\phi(S[x]) = 1 = B[0] = B[\phi(x)]. \quad (5.25)$$

Se $x \notin R(S)$, sabemos que existe $S[x] \in E(S)$ e que $\phi(x) = 1$. Além disso, $S[x] \notin R(S)$, e portanto

$$\phi(S[x]) = 1 = B[1] = B[\phi(x)]. \quad (5.26)$$

Desta forma, ϕ é de fato uma simulação de S em B . ■

Nota. Podemos demonstrar, de uma forma basicamente análoga ao feito na proposição anterior, que toda componente de tipo 2 pode ser simulada em uma componente como o B apresentado. Em outras palavras, um ponto fixo e uma raiz bastam para simularmos qualquer comportamento infinito, seja ciclo ou não.

Proposição 5.1.15. Seja C um computador. Existe um computador K , com no máximo duas componentes, tal que podemos simular C em K .

Demonstração. Considere $P = \{2^n : n \in \mathbb{N}\}$ e $Q = \{3, 5\}$. Montaremos o computador K com a seguinte função:

$$f(n) = \begin{cases} n/2, & n \in P - \{1\} \\ 3, & n \in Q \end{cases} \quad (5.27)$$

Dessa forma, obtemos que $D(K) = (P - \{1\}) \cup Q$ e $I(K) = (P \cup Q) - \{5\}$. Seja $S \in \bar{C}$. Se S é do tipo 3 ou do tipo 2, sabemos pela proposição anterior que existe uma simulação de S em $\Delta(5, K) = \Delta(3, K)$.

Se S é de tipo 1, seja $f \in F(S)$ a folha de S e defina ϕ_S como

$$\phi_S(n) = \begin{cases} 1, & n = f \\ 2^k & \exists k \in \mathbb{N} : k \neq 0 \wedge C^k[n] = f. \end{cases} \quad (5.28)$$

Como todo elemento $x \in E(S)$ deve ser tal que $C^p[x] = f$ para algum $p \in \mathbb{N}$, sabemos que $E(S)$ é de fato o domínio de ϕ_S . Além disso, se $a, b \in E(S)$ forem tais que $a = b$, então se temos $k \in \mathbb{N}$ com $C^k[a] = f$, o determinismo nos diz que $C^k[b] = f$ e assim $\phi(a) = \phi(b)$. Logo, ϕ_S está bem definida.

Tome $x \in D(S)$ e seja $r \in \mathbb{N}$ com $C^r[x] = f$. Temos que $\phi(x) = 2^r$ e $\phi(C[x]) = 2^{(r-1)}$. Logo,

$$\phi(C[x]) = 2^{(r-1)} = 2^r / 2 = K[2^r] = K[\phi(x)] \quad (5.29)$$

e de fato ϕ_S é uma simulação. Como tomamos $S \in \bar{C}$ arbitrário, qualquer componente de tipo 1 pode ser simulada em K . Como qualquer componente de C pode ser simulada em alguma componente de K , por [5.1.13](#) sabemos que existe uma simulação de C em K . Da nossa construção de K , vemos que há apenas duas componentes de fato: $\Delta(1, K)$ e $\Delta(3, K)$. ■

6 COMPLEXIDADE ORBITAL DOS NÚCLEOS

Começaremos definindo o que se apresenta como complexidade para o nosso caso e então encerraremos com uma primeira definição de classe de complexidade. Antes, iremos construir a notação e algumas intuições. Para este capítulo, assuma sempre um computador C dado com núcleo não vazio.

6.1 Complexidade Orbital

Considere o conjunto $T(C)$ dos tamanhos das órbitas dos núcleos de C , ou seja

$$T(C) = \{|\theta(n, C)| : n \in N(C)\}. \quad (6.1)$$

Note que ignoramos multiplicidade de tamanhos de órbita. A partir de $T(C)$, obtemos uma sequência de conjuntos $T(C)_0, T(C)_1, \dots, T(C)_n, \dots$ tais que

$$\begin{aligned} T(C)_0 &= T(C) \\ T(C)_1 &= T(C)_0 - \{\min T(C)_0\} \\ T(C)_2 &= T(C)_1 - \{\min T(C)_1\} \\ &\vdots \\ T(C)_n &= T(C)_{n-1} - \{\min T(C)_{n-1}\} \\ &\vdots \end{aligned} \quad (6.2)$$

exceto nos casos onde existir $k \in \mathbb{N}$ tal que $T(C)_k = \emptyset$. Sem perda de generalidade, assumimos que este é o menor k tal que isso acontece. Nesse caso, faremos $T(C)_n = T(C)_{k-1}$ para todo $n \geq k$. Esta sequência de conjuntos está bem definida devido à propriedade da boa ordem dos naturais, bastando observar que $T(C)_n \subseteq \mathbb{N}$ para todo $n \in \mathbb{N}$, e portanto faz sentido pensar em um elemento mínimo.

Com esta sequência de conjuntos, obtemos uma sequência $t(n)$ de números naturais dada por $t(n) = \min T(C)_n$. Com esta sequência $t(n)$, estamos prontos

para definir a complexidade orbital do núcleo de C . Apesar de omitida na notação, $t(n)$ ainda depende de C .

Definição 6.1.1. Seja C um computador e considere sua sequência de tamanhos de órbitas $t(n)$ como descrita anteriormente. Diremos que uma função *contínua* f é **assíntota** de $N(C)$ se, e somente se, existir $c \in \mathbb{R}$ tal que

$$\lim_{n \rightarrow \infty} \frac{t(n)}{f(n)} = c. \quad (6.3)$$

Em outras palavras, se o limite acima existir. Existindo este limite, diremos que f é **não dominante** de $N(C)$ se $c \neq 0$. Se $c = 1$, então diremos que f é **canônica**.

Nota. Funções dominantes de um núcleo dizem pouco sobre o crescimento de suas órbitas. Por exemplo, observamos que se (por alguma coincidência), tivermos $t(n) = n$, então e^n , $n \log n$, n^a para $a > 1$ são todas dominantes.

Proposição 6.1.2. Se existe assíntota não dominante para $N(C)$, então existe assíntota canônica.

Demonstração. Como

$$\lim_{x \rightarrow \infty} \frac{t(x)}{f(x)} = c \quad (6.4)$$

por uma propriedade dos limites temos que

$$\frac{1}{c} \lim_{x \rightarrow \infty} \frac{t(x)}{f(x)} = \lim_{x \rightarrow \infty} \frac{t(x)}{cf(x)} = \frac{c}{c} = 1 \quad (6.5)$$

e portanto $cf(x)$ é assíntota canônica de $N(C)$. ■

Nota. Observamos que podem existir mais de uma assíntota canônica. De fato, se x^2 é assíntota canônica de algum $N(C)$ então $x^2 + x$ e $x^2 + x + 1$ também serão. Falaremos então de **classe** de complexidade orbital como sendo o conjunto de funções assíntotas canônicas de um dado $N(C)$.

Definição 6.1.3. A **classe de complexidade orbital** de um dado núcleo $N(C)$ será o conjunto de funções assíntotas canônicas de $N(C)$.

A terminologia similar (e mesmo a formulação) de fato buscam remeter às classes de complexidade usuais. Uma diferença fundamental, no entanto, é que não temos noção de tamanho de entrada aqui desenvolvida.

Um resgate das propriedades do limite pode ser feita em (Guidorizzi 2001).

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, propusemos uma abordagem para o estudo da computação a partir de uma perspectiva dinâmica. Partindo de uma estrutura simples, dada por um conjunto de funções em $\mathbb{N}$, buscamos compreender como históricos de computação podem ser descritos, relacionados e analisados por meio de suas órbitas. O objetivo central não foi substituir os modelos clássicos de computação, mas oferecer uma linguagem na qual certos comportamentos computacionais, usualmente compreendidos de forma intuitiva em modelos como as máquinas de Turing, apareçam como consequências formais das definições adotadas.

Os resultados apresentados ao final do capítulo de dinâmica computacional constituem uma das principais contribuições do trabalho. Eles mostram que a dinâmica observada intuitivamente em máquinas de Turing pode ser recuperada de maneira formal no modelo proposto. Em particular, a decomposição dos históricos em comportamentos de parada, ciclos e trajetórias infinitas evidencia que a formulação adotada é capaz de capturar aspectos fundamentais da computação determinística sem exigir, de início, uma estrutura excessivamente complexa.

Outro ponto importante da construção é a noção de simulação. Essa definição expressa a ideia de que um computador pode representar a dinâmica de outro quando seus passos de computação são preservados por uma função adequada. A inspiração em homomorfismos é natural: assim como homomorfismos preservam a estrutura relevante em outros contextos matemáticos, a simulação preserva a evolução computacional. Essa escolha permitiu comparar históricos e órbitas de forma compatível com a dinâmica do modelo, abrindo caminho para uma análise mais sistemática das relações entre diferentes computadores.

A principal qualidade do modelo, a nosso ver, está em sua simplicidade. A partir de uma estrutura inicial bastante elementar, foi possível obter naturalmente noções de histórico, órbita, componente, simulação e complexidade orbital. Essa simplicidade não elimina as dificuldades do modelo, mas torna mais claro o papel de cada definição e permite que as propriedades dinâmicas surjam progressivamente ao longo do texto. Assim, o trabalho contribui ao organizar uma linguagem

na qual fenômenos computacionais conhecidos podem ser revisitados por meio de ferramentas dinâmicas.

Trabalhos Futuros

Há diversos caminhos naturais para o desenvolvimento posterior desta abordagem. Um primeiro ponto diz respeito à computabilidade dos passos de computação. Como a formulação apresentada permite funções genéricas, não há, em princípio, garantia de que cada passo computacional seja efetivamente computável. Embora isso não impeça a análise dinâmica dos históricos, compreender melhor os limites dessa generalidade é uma direção importante para trabalhos futuros.

Relacionado a isso, permanece a questão de delimitar quais históricos de computação correspondem de fato a procedimentos efetivos. Neste trabalho, mostramos que os históricos de computação de máquinas de Turing estão contemplados pela análise proposta. No entanto, isso não implica que toda dinâmica admitida pelo modelo possa ser induzida por uma máquina de Turing ou por outro procedimento efetivo equivalente. Investigar essa fronteira entre dinâmica formal e computação efetiva parece ser um dos desdobramentos mais importantes da teoria.

Outra direção possível é o desenvolvimento de noções mais refinadas de equivalência entre históricos de computação. A simulação captura de maneira interessante a preservação da dinâmica, mas os resultados obtidos indicam que ela pode ser forte demais para definir, por si só, uma noção satisfatória de equivalência. Uma teoria mais robusta de equivalências dependeria, provavelmente, de uma compreensão mais precisa de quais históricos representam procedimentos efetivos e de quais aspectos da dinâmica devem ser preservados.

Por fim, há também espaço para aprofundar a noção de complexidade orbital. A definição apresentada sugere conexões com cálculo e análise, o que já possui paralelos no estudo clássico da complexidade temporal, mas ganha aqui uma formulação compatível com a perspectiva dinâmica adotada no trabalho. Investigar essas conexões pode revelar novas formas de comparar históricos, medir crescimento orbital e relacionar propriedades dinâmicas com noções tradicionais de complexidade computacional.

REFERÊNCIAS BIBLIOGRÁFICAS

Blum 1967 BLUM, M. A machine-independent theory of the complexity of recursive functions. *J. ACM*, Association for Computing Machinery, New York, NY, USA, v. 14, n. 2, p. 322–336, abr. 1967. ISSN 0004-5411. Disponível em: <https://doi.org/10.1145/321386.321395>.

Diestel 2017 DIESTEL, R. *Graph Theory*. 5. ed. [S.l.]: Springer, 2017.

Enderton 1977 ENDERTON, H. B. *Elements of Set Theory*. [S.l.]: Academic Press, 1977.

Guidorizzi 2001 GUIDORIZZI, H. L. *Um Curso de Cálculo*. 5. ed. Rio de Janeiro: LTC, 2001. v. 1.

Halmos 1960 HALMOS, P. R. *Naive Set Theory*. [S.l.]: Van Nostrand, 1960.

Harary e Palmer 1973 HARARY, F.; PALMER, E. M. A survey of graphical enumeration problems. In: SRIVASTAVA, J. N. (Ed.). *A Survey of Combinatorial Theory*. Amsterdam: North-Holland, 1973. p. 259–275.

Immerman 1999 IMMERMANN, N. *Descriptive Complexity*. New York, NY: Springer, 1999. (Graduate Texts in Computer Science). ISBN 978-1-4612-0539-5.

Jech 2003 JECH, T. *Set Theory*. [S.l.]: Springer, 2003.

Lagarias 2010 LAGARIAS, J. C. (Ed.). *The Ultimate Challenge: The $3x+1$ Problem*. Providence, RI: American Mathematical Society, 2010.

Lewis e Papadimitriou 1998 LEWIS, H. R.; PAPADIMITRIOU, C. H. *Elements of the Theory of Computation*. [S.l.]: Prentice-Hall, 1998.

Milnor 2006 MILNOR, J. *Dynamics in One Complex Variable*. 3. ed. [S.l.]: Princeton University Press, 2006.

Sandefur 1990 SANDEFUR, J. T. *Discrete Dynamical Systems: Theory and Applications*. New York: Oxford University Press, 1990.

Sipser 2006 SIPSER, M. *Introduction to the Theory of Computation*. 2nd. ed. [S.l.]: International Thomson Publishing, 2006. ISBN 053494728X.

West 2001 WEST, D. B. *Introduction to Graph Theory*. 2. ed. [S.l.]: Prentice Hall, 2001.

APÊNDICE A — ARTIGO EM FORMATO SBC

Históricos de Computação como Dinâmica Discreta

Gian Carlo Figueiredo Ferrari¹

¹Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

`giancarlofferrari@gmail.com`

Abstract. *This paper proposes a model in which computation histories are interpreted as discrete dynamical systems. A computer is defined as a finite family of functions on subsets of the natural numbers, and successive computational steps induce orbits in a discrete state space. The model represents recursively enumerable languages through deterministic Turing machines and supports a structural analysis based on functional graphs, connected components and periodic behavior. A notion of simulation that preserves computational steps is also introduced, together with an exploratory measure of orbital complexity. The resulting framework provides a unified language for describing termination, periodicity and infinite evolution in computational processes.*

Resumo. *Este artigo propõe um modelo no qual históricos de computação são interpretados como sistemas dinâmicos discretos. Um computador é definido como uma família finita de funções sobre subconjuntos dos números naturais, e passos sucessivos de computação induzem órbitas em um espaço discreto. O modelo representa linguagens recursivamente enumeráveis por meio de máquinas de Turing determinísticas e permite uma análise estrutural baseada em grafos funcionais, componentes conexas e comportamento periódico. Introduz-se ainda uma noção de simulação que preserva passos computacionais e uma medida exploratória de complexidade orbital. A estrutura resultante oferece uma linguagem unificada para descrever parada, periodicidade e evolução infinita em processos computacionais.*

1. Introdução

A teoria da computação estuda processos computacionais por meio de modelos formais que permitem discutir computabilidade, reconhecimento de linguagens e complexidade. Máquinas de Turing, funções recursivas e outros formalismos descrevem como uma computação é realizada e quais problemas podem ser resolvidos por procedimentos efetivos [Sipser 2006, Lewis and Papadimitriou 1998]. Embora esses modelos enfatizem aspectos distintos, sua equivalência expressiva, usualmente associada à tese de Church–Turing, sugere a possibilidade de investigar propriedades da computação sem fixar um mecanismo operacional específico.

Uma execução computacional também determina uma evolução temporal. Partindo de uma entrada ou configuração inicial, a aplicação sucessiva de regras de transição produz uma sequência de estados, isto é, um histórico de computação. Essa sequência pode ser examinada não apenas como registro intermediário de um algoritmo, mas como

objeto matemático. Sob essa perspectiva, a computação se aproxima de uma dinâmica discreta: uma função, ou uma família compatível de funções, é iterada em um espaço de estados, produzindo órbitas, ciclos e trajetórias que podem terminar ou prosseguir indefinidamente.

Este artigo apresenta um modelo abstrato destinado ao estudo dessa estrutura. Um computador é definido como uma família finita de funções cujos domínios são subconjuntos não vazios de $\mathbb{N}$ e cujas imagens também pertencem a $\mathbb{N}$. No caso determinístico, essas funções induzem um passo de computação bem definido. A iteração desse passo produz os históricos analisados ao longo do trabalho e, ao mesmo tempo, determina um grafo funcional no qual os elementos do espaço do computador são vértices e cada computação possível é representada por uma aresta dirigida.

A simplicidade da definição não pretende substituir modelos clássicos de computação. Seu objetivo é fornecer uma representação em que propriedades dinâmicas sejam explícitas. Para justificar sua expressividade, mostra-se que, para toda máquina de Turing determinística, pode-se construir um computador cujo núcleo está em bijeção com a linguagem reconhecida pela máquina. Assim, linguagens recursivamente enumeráveis podem ser observadas no modelo sem que a análise dependa diretamente da estrutura interna da máquina de Turing.

A partir dessa construção, definem-se órbitas, pré-órbitas e componentes. As componentes do grafo funcional são classificadas segundo três comportamentos: trajetórias que alcançam uma folha, trajetórias que alcançam um ciclo e trajetórias infinitas sem repetição. Essa classificação reúne, em uma única estrutura, fenômenos computacionais correspondentes à parada, à periodicidade e à evolução indefinida.

O artigo também considera relações entre computadores. Uma simulação é definida por uma aplicação entre seus espaços que preserva passos de computação. Essa condição corresponde a uma relação de semiconjugação entre as dinâmicas e permite investigar quais propriedades orbitais são preservadas quando um computador representa o comportamento de outro. Por fim, introduz-se uma noção de complexidade orbital associada ao comprimento dos históricos que partem do núcleo, como uma aproximação inicial entre crescimento orbital e complexidade computacional. A proposta se relaciona ao objetivo de estudar complexidade em estruturas abstratas, sem depender exclusivamente de um modelo particular de máquina [Blum 1967].

As contribuições centrais são: (i) a formulação de computadores e históricos como uma dinâmica em $\mathbb{N}$; (ii) a representação de linguagens recursivamente enumeráveis no núcleo do modelo; (iii) a caracterização das componentes induzidas pela dinâmica determinística; (iv) a definição de simulação por preservação dos passos de computação; e (v) a proposição de uma medida de complexidade orbital.

O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta as definições do modelo e sua relação com máquinas de Turing. A Seção 3 estuda órbitas e componentes. A Seção 4 introduz simulações entre computadores e apresenta a complexidade orbital. Por fim, a Seção 5 reúne as conclusões e indica trabalhos futuros.

2. Modelo Proposto

Seja $\mathbb{N} = \{0, 1, 2, \dots\}$ o conjunto dos números naturais. Neste trabalho, a computação é modelada por meio de funções definidas em subconjuntos de $\mathbb{N}$.

Definição 2.1. Sejam $D_1, \dots, D_k \subseteq \mathbb{N}$ conjuntos não vazios. Um *computador* é uma coleção finita

$$C = \{f_i : D_i \rightarrow \mathbb{N} \mid 1 \leq i \leq k\}.$$

Define-se

$$D(C) = \bigcup_{i=1}^k D_i,$$

denominado domínio do computador, e

$$I(C) = \bigcup_{i=1}^k \text{Im}(f_i),$$

denominado imagem do computador. O espaço do computador é

$$E(C) = D(C) \cup I(C).$$

Quando duas funções possuem interseção em seus domínios, pode haver mais de uma computação possível para um mesmo elemento.

Definição 2.2. Um computador é dito determinístico quando

$$n \in D_i \cap D_j \implies f_i(n) = f_j(n),$$

para quaisquer $f_i, f_j \in C$.

Caso contrário, o computador é denominado não determinístico.

Neste trabalho restringimo-nos ao caso determinístico. Nessa situação, o passo de computação está bem definido.

Definição 2.3. Seja C um computador determinístico. O *passo de computação* de um elemento $n \in D(C)$ é definido por

$$C[n] = f_i(n),$$

onde $n \in D_i$.

A aplicação sucessiva dos passos de computação induz naturalmente uma dinâmica discreta.

Definição 2.4. A iteração de um elemento n é a sequência

$$(n, C[n], C^2[n], \dots),$$

onde $C^0[n] = n$ e

$$C^{k+1}[n] = C[C^k[n]].$$

Essa sequência será denominada iteração de n .

A interpretação das iterações como órbitas de uma dinâmica constitui o elemento central do modelo proposto. Dessa forma, conceitos clássicos de sistemas dinâmicos podem ser empregados para estudar propriedades estruturais da computação.

Além dos elementos pertencentes ao domínio e à imagem do computador, destacam-se dois subconjuntos de interesse.

Definição 2.5. O conjunto

$$R(C) = D(C) \setminus I(C)$$

é denominado conjunto das raízes.

Analogamente,

$$F(C) = I(C) \setminus D(C)$$

é denominado conjunto das folhas. Finalmente,

$$N(C) = \{n \in R(C) : \exists p \in C^p[n] \in F(C)\}$$

será o núcleo de C , ou seja, as raízes cuja computação termina em folha.

As raízes representam estados que não podem ser alcançados por nenhuma computação anterior, enquanto as folhas correspondem a estados sem continuação computacional. Essa interpretação permite visualizar computadores como grafos funcionais, nos quais raízes e folhas assumem papéis análogos aos de árvores direcionadas.

2.1. Expressividade do Modelo

Apesar da simplicidade de sua definição, o modelo preserva a capacidade de representar computações arbitrárias.

A ideia consiste em associar cada configuração de uma máquina de Turing a um número natural por meio de uma enumeração das configurações possíveis. A função induzida pelas transições da máquina passa então a definir um computador segundo a Definição 1.

Esse resultado é formalizado pela proposição seguinte.

Proposição 2.6. *Para toda máquina de Turing determinística T existe um computador C tal que o conjunto das entradas aceitas por T está em bijeção com o núcleo de C .*

A construção utiliza uma enumeração das configurações da máquina de Turing, permitindo interpretar cada transição como um passo de computação no modelo proposto. Como consequência, linguagens recursivamente enumeráveis podem ser representadas

por computadores definidos neste trabalho, preservando a expressividade computacional do modelo clássico.

A partir desse resultado, as propriedades estudadas nas próximas seções podem ser interpretadas como propriedades gerais dos históricos de computação, independentemente do modelo concreto utilizado para sua implementação.

3. Dinâmica Computacional

Uma vez definido o passo de computação, a iteração induz naturalmente uma dinâmica discreta sobre o espaço do computador. Esta seção introduz os principais objetos associados a essa dinâmica e apresenta uma caracterização estrutural de seus componentes.

3.1. Órbitas e Pré-Órbitas

A iteração de um computador permite descrever tanto a evolução futura de um estado quanto os estados capazes de alcançá-lo.

Definição 3.1. Seja C um computador e $n \in E(C)$.

A órbita de n é definida por

$$\theta(n, C) = \{m \in E(C) : \exists k \in \mathbb{N}, C^k[n] = m\}.$$

A pré-órbita de n é

$$V(n, C) = \{m \in E(C) : \exists k \in \mathbb{N}, C^k[m] = n\}.$$

Enquanto a órbita descreve todos os estados atingidos a partir de n , a pré-órbita reúne todos os estados cuja computação eventualmente alcança n . Esses conjuntos desempenham papel análogo às órbitas direta e inversa estudadas em sistemas dinâmicos discretos.

As órbitas preservam naturalmente a estrutura induzida pela iteração.

Proposição 3.2. Sejam $m, n \in E(C)$. Então

$$m \in \theta(n, C) \iff \theta(m, C) \subseteq \theta(n, C).$$

De forma dual,

$$m \in V(n, C) \iff V(m, C) \subseteq V(n, C).$$

Essas propriedades mostram que órbitas e pré-órbitas são fechadas pela própria dinâmica, constituindo os blocos básicos da estrutura computacional.

3.2. Componentes

A combinação entre órbitas e pré-órbitas permite definir regiões maximais do espaço nas quais todos os elementos participam da mesma estrutura dinâmica.

Definição 3.3. Seja $n \in E(C)$.

A componente de n é o conjunto

$$\Delta(n, C) = \bigcup_{x \in \theta(n, C)} V(x, C).$$

Intuitivamente, uma componente contém todos os estados que pertencem ao mesmo histórico computacional, seja por evolução futura, seja por retrocesso através das pré-órbitas.

Essa definição coincide naturalmente com as componentes conexas do grafo funcional induzido pelo computador.

Para tornar essa relação explícita, introduz-se a seguinte noção de conexidade.

Definição 3.4. Sejam $m, n \in E(C)$.

Dizemos que m é conexo a n quando

$$\theta(m, C) \cap \theta(n, C) \neq \emptyset.$$

Denotaremos essa relação por

$$m \equiv n.$$

Proposição 3.5. A relação $\equiv$ é uma relação de equivalência.

Como consequência, o espaço do computador é particionado em classes de equivalência, cada uma delas coincidindo com exatamente uma componente da dinâmica.

3.3. Classificação das Componentes

Uma consequência importante da estrutura funcional dos computadores é que toda componente admite apenas um número limitado de comportamentos assintóticos.

Teorema 3.6. *Toda componente de um computador determinístico pertence exatamente a um dos seguintes tipos:*

1. *componentes cuja dinâmica termina em uma folha;*
2. *componentes contendo exatamente um ciclo;*
3. *componentes cujas órbitas são infinitas.*

O primeiro caso corresponde a computações finitas. No segundo, a computação torna-se periódica após um número finito de iterações. O terceiro descreve computações infinitas sem repetição periódica.

Essa classificação fornece uma descrição completa do comportamento qualitativo das componentes de um computador determinístico.

Além disso, componentes distintas não podem compartilhar folhas nem ciclos, implicando que a estrutura global de um computador pode ser interpretada como uma união disjunta desses três tipos fundamentais de dinâmica.

Essa caracterização constitui um dos principais resultados estruturais do modelo proposto, pois reduz o estudo de históricos arbitrários ao estudo dessas três formas básicas de evolução computacional.

4. Simulação e Complexidade Orbital

Além da análise da dinâmica interna de um computador, é natural perguntar quando dois computadores podem ser considerados equivalentes do ponto de vista de sua evolução temporal.

A noção introduzida nesta seção procura capturar exatamente essa ideia. Em vez de exigir igualdade entre computadores, considera-se suficiente que um deles reproduza a dinâmica do outro por meio de uma transformação entre seus espaços de estados.

4.1. Simulação entre Computadores

Definição 4.1. Sejam C e K computadores.

Uma função

$$\phi : E(C) \rightarrow E(K)$$

é denominada uma *simulação* quando

$$\phi(C[n]) = K[\phi(n)]$$

para todo

$$n \in D(C).$$

Equivalentemente,

$$\phi \circ C = K \circ \phi.$$

Essa igualdade mostra que a aplicação ϕ preserva cada passo da computação. Em outras palavras, executar um passo em C e depois aplicar ϕ produz o mesmo resultado que primeiro transformar o estado por ϕ e, em seguida, executar um passo em K .

Do ponto de vista da teoria de sistemas dinâmicos, essa condição corresponde precisamente a uma semiconjugação entre dinâmicas discretas.

A principal consequência é que toda a evolução computacional é preservada.

Proposição 4.2. Se $\phi : C \rightarrow K$ é uma simulação, então

$$\phi(C^m[n]) = K^m[\phi(n)]$$

para todo $m \in \mathbb{N}$.

Demonstração. A demonstração segue imediatamente por indução sobre m .

Para $m = 0$ a igualdade é trivial.

Supondo válida para m , temos

$$\phi(C^{m+1}[n]) = \phi(C[C^m[n]]) = K[\phi(C^m[n])] = K[K^m[\phi(n)]] = K^{m+1}[\phi(n)].$$

□

Como consequência imediata, propriedades orbitais são preservadas por simulações.

Corolário 4.3. *Se ϕ é uma simulação, então*

$$\phi(\theta(n, C)) \subseteq \theta(\phi(n), K).$$

Assim, ciclos, computações finitas e estruturas periódicas permanecem compatíveis com a transformação entre computadores.

Essa interpretação permite comparar computadores segundo sua estrutura dinâmica, independentemente da implementação específica de cada um.

4.2. Complexidade Orbital

A classificação dinâmica apresentada anteriormente descreve o tipo de comportamento de uma computação. Entretanto, ela não fornece informações sobre a velocidade com que a dinâmica se desenvolve.

Com esse objetivo introduz-se uma medida denominada complexidade orbital.

Tomando elementos das raízes de um computador C , consideramos o conjunto

$$T(C) = \{|\theta(n, C)| : n \in N(C)\} \quad (1)$$

de tamanhos das órbitas dos elementos do núcleo. Ou seja, os tamanhos das órbitas das computações que terminam. Podemos colocar estes tamanhos em ordem crescente em uma sequência de números naturais $t(n)$, repetindo o último número caso $T(C)$ seja finito.

Assim, definimos que uma função contínua f é assíntota de t se

$$\lim_{n \rightarrow \infty} \frac{t(n)}{f(n)} = c \in \mathbb{R}.$$

Diremos que f é não-dominante se $c \neq 0$. Uma função assíntota será canônica se $c = 1$. Em particular, toda $t(n)$ com assíntota não-dominante admite assíntota canônica. Como podemos ter mais de uma assíntota canônica, chegamos na seguinte definição:

Definição 4.4. A classe de complexidade orbital de C será o conjunto de funções assíntotas canônicas da sequência $t(n)$ associada a $N(C)$.

5. Conclusões

Este trabalho apresentou um modelo abstrato de computação baseado em famílias finitas de funções sobre os números naturais, no qual a execução de um algoritmo é interpretada como uma dinâmica discreta.

A formulação proposta permitiu descrever históricos de computação por meio de conceitos como órbitas, componentes conexas e grafos funcionais, obtendo uma classificação estrutural dos possíveis comportamentos de um computador determinístico.

Também foi introduzida uma noção de simulação entre computadores baseada na preservação dos passos computacionais. Essa construção estabelece uma conexão direta com o conceito de semiconjugação em sistemas dinâmicos e fornece um mecanismo para comparar computadores segundo sua estrutura dinâmica.

Por fim, foi proposta uma medida de complexidade orbital destinada a descrever o crescimento das órbitas produzidos pela dinâmica computacional. Embora exploratória, essa medida sugere novas possibilidades para o estudo da relação entre dinâmica discreta e teoria da computação.

Como trabalhos futuros, pretende-se aprofundar a teoria da complexidade orbital, investigar invariantes preservados por simulações e estender o modelo para o caso não determinístico.

Referências

- Blum, M. (1967). A machine-independent theory of the complexity of recursive functions. *J. ACM*, 14(2):322–336.
- Lewis, H. R. and Papadimitriou, C. H. (1998). *Elements of the Theory of Computation*. Prentice-Hall.
- Sipser, M. (2006). *Introduction to the Theory of Computation*. International Thomson Publishing, 2nd edition.