UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PROPRIEDADES DECIDÍVEIS DE AUTÔMATOS CELULARES FINITOS, HÍBRIDOS, NÃO-LINEARES, SENSÍVEIS E REVERSÍVEIS

LEONARDO DE SÁ ALT

Uberlândia - Minas Gerais
2013

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LEONARDO DE SÁ ALT

PROPRIEDADES DECIDÍVEIS DE AUTÔMATOS CELULARES FINITOS, HÍBRIDOS, NÃO-LINEARES, SENSÍVEIS E REVERSÍVEIS

Dissertação de Mestrado apresentada à Faculdade de Ciência da Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Inteligência Artificial.

Orientadora:
Prof[a]. Dr[a]. Gina Maira Barbosa de Oliveira

Uberlândia, Minas Gerais
2013

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

A465p
2013

Alt, Leonardo de Sá, 1989-
Propriedades decidíveis de autômatos celulares finitos, híbridos, não-lineares, sensíveis e reversíveis / Leonardo de Sá Alt. - 2013.
73 f. : il.

Orientadora: Gina Maira Barbosa de Oliveira.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Criptografia - Teses. 3. Robôs - Teses. I. Oliveira, Gina Maira Barbosa de. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Ciência da Computação a aceitação da dissertação intitulada "**Propriedades Decidíveis de Autômatos Celulares Finitos, Híbridos, Não-Lineares, Sensíveis e Reversíveis**" por **Leonardo de Sá Alt** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 25 de Fevereiro de 2013

Orientadora: _____

Profª. Drª. Gina Maira Barbosa de Oliveira
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Profª. Drª. Sandra Aparecida de Amo
Universidade Federal de Uberlândia

_____

Prof. Dr. Pedro Paulo Balbi de Oliveira
Universidade Presbiteriana Mackenzie

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Fevereiro de 2013

Autor:	**Leonardo de Sá Alt**
Título:	**Propriedades Decidíveis de Autômatos Celulares Finitos, Híbridos, Não-Lineares, Sensíveis e Reversíveis**
Faculdade:	**Faculdade de Ciência da Computação**
Grau:	**Mestrado**

_____

Autor

Agradecimentos

Aos meus pais, Jairo e Raquel, e ao meu irmão Miguel pelo apoio (e leve pressão, saudável e sempre necessária para mim) em todas as fases deste trabalho.

À professora Dr^{a}. Gina Maira Barbosa de Oliveira, minha orientadora desde a metade do meu bacharelado, pela oportunidade que me deu há alguns anos atrás de trabalhar com ela, por compartilhar seus conhecimentos comigo, por sua companhia, por ser a responsável pelos conhecimentos que adquiri ao decorrer deste trabalho e pela confiança que sempre teve em mim.

À professora Dr^{a}. Sandra de Amo e à professora Dr^{a}. Márcia Fernandes, que junto com a professora Gina formam, para mim, o trio de melhores professores desta faculdade. Tive a enorme felicidade de aprender sobre os assuntos mais importantes da Ciência da Computação com elas. O conhecimento que têm e a forma como conseguem expressá-lo é simplesmente impressionante, e devo a elas o conhecimento que adquiri. Aprendi muito com as três, e por isso sou grato.

Ao professor Dr. Lásaro Camargos, *coach* do meu (aposentado) time da Maratona de Programação ICPC, com quem também aprendi muito. Infelizmente não tive a oportunidade de fazer alguma disciplina com este honrado, nobre, íntegro, digno e imaculado professor, que se tornou um amigo.

Aos meus amigos e colegas de computação Enrique Fynn, Thadeu Tucci, Rodrigo Saramago, Lucas Vella, Tuanir Rezende e Lord Spy pelas conversas e discussões com muito sentido que já tivemos. Ao Lucas Vella agradecimento especial pelas discussões sobre este trabalho.

Aos meus amigos (de fora da computação) Álvaro, Jordana, Rafael, Ludmila, Igor, Lílian, Lucas, Bárbara, Fernando, Tainá, Felipe, Miriã, Lucas, Pio, Cidão e Fábio, pelo apoio e motivação.

Aos meus amigos do mestrado e de laboratório Thiago, Danielli, Giordano Bruno e Murillo, pelo apoio, motivação e pelo tempo que passamos juntos em nossa "2^{a} casa".

À CAPES, pelo apoio financeiro.

E finalmente, agradeço ao *Heavy Metal*.

Resumo

Nós investigamos a decidibilidade e complexidade dos problemas do Predecessor e da Alcançabilidade em Autômatos Celulares Finitos, Híbridos, Reversíveis, Sensíveis e Não-Lineares. Demonstramos a reversibilidade do modelo, aqui definido como HSR, resolvendo assim o Problema do Predecessor. Utilizando a Forma Normal Disjuntiva para representar as funções de transição, conseguimos por derivadas parciais booleanas transformá-las para a Forma Normal Algébrica. Mostramos que utilizando a forma matricial e também as derivadas parciais booleanas é possível calcular vários passos da evolução temporal do modelo HSR em tempo polinomial; com isso demonstramos que o Problema da Alcançabilidade pertence à classe "Arthur-Merlin" AM_2 e por isso não pode ser NP-Completo (a não ser que a hierarquia colapse). Também propusemos um novo método criptográfico baseado no modelo de AC HSR, cujas chaves criptográficas são combinações de funções de transição elementares, o que aumenta a eficiência do método sem abrir mão da segurança, já que mesmo tamanhos pequenos de reticulado fazem a cardinalidade do espaço de chaves ser muito grande.

Palavras chave: autômatos celulares, problema do predecessor, problema da alcançabilidade, pep, crep, criptografia, reversibilidade, sensitividade.

Abstract

We investigated the decidability and complexity of the Predecessor and the Configuration Reachability problems in Non-Linear, Sensitive, Reversible, Hybrid and Finite Cellular Automata. We demonstrated the model's reversibility (defined here as HSR, *Híbrido Sensível Reversível*, or Hybrid Reversible Toggle), which, in turn solves the Predecessor's Problem. Using Disjunctive Normal Form to represent transition functions, by Boolean partial derivatives, we could transform them to the Algebraic Normal Form. We show that using matrix form and Boolean partial derivatives it is possible to calculate several HSR evolution steps in polynomial time; so we demonstrated that the Configuration Reachability Problem belongs to the complexity class "Arthur-Merlin" AM_2 and cannot be NP-Complete (unless the hierarchy collapses). We also proposed a new cryptographic method based on the model HSR, whose cryptographic keys are combinations of elementary transition functions, what increases the method's eficiency, without compromising security, since even small lattice sizes make the key space cardinality very large.

Keywords: cellular automata, predecessor problem, configuration reachability problem, pep, crep, cryptography, reversibility, sensitiviy.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Um autômato celular (AC) consiste de um *array* de autômatos (de estados) finitos (também chamados de células) conectados localmente que evoluem de modo síncrono e discreto. As interações locais determinam uma função global F que atua no espaço C de todas as configurações possíveis do *array* de células e o comportamento do sistema pode ser descrito pela equação $X^{i+1} = F(X^i), X^i \in C$ para todo $t \geq 0$ [16].

Autômatos celulares (ACs) vêm sendo cada vez mais estudados e utilizados em aplicações de vários tipos [7, 45–47], tais como: reconhecimento de linguagens [35, 36], processamento de imagens [14, 32], escalonamento [5, 6, 31, 39, 40], modelagem de sistemas biológicos [4, 44], modelagem de sistemas físicos [8, 11] e criptografia [7, 13, 19, 22, 25, 27, 28, 33, 34, 46, 50].

Em 1986, Wolfram sugeriu um método criptográfico baseado em ACs [46]. Após este início, vários outros pesquisadores aderiam à idéia e criaram cada vez mais diversificados métodos de criptografia baseados em autômatos celulares [7, 13, 19, 22, 27–30, 33, 34, 41, 50]. Alguns deles, utilizando a mesma idéia básica de Wolfram: usar o AC para gerar sequências pseudoaleatórias que posteriormente são usadas como chaves criptográficas [34, 46]. Outros utilizam propriedades algébricas de ACs lineares e criam ACs reversíveis com diagrama de transição de estados conhecido [7, 33]. Já outros utilizam o conceito de pré-imagens de ACs como base para cifragem em seus métodos [13, 19, 27, 28].

O método de criptografia HCA [19] pertence ao terceiro tipo e é baseado no cálculo de pré-imagens de autômatos celulares; ele foi proposto e avaliado em [19]. O HCA utiliza duas regras diferentes sensíveis a uma mesma célula no extremo da vizinhança (*toggle rules*), uma chamada regra principal e outra regra de borda, que é puramente sensível. A cifragem é feita através de múltiplos cálculos de pré-imagem a partir de uma configuração inicial que é o texto a ser cifrado. A decifragem é feita através da evolução temporal do AC pelo mesmo número de passos da cifragem, tendo como configuração inicial o texto cifrado e configuração final o texto original. Em [19] foi mostrado experimentalmente que este método possui bom desempenho e nível de segurança, mas uma análise de caráter teórico ainda não havia sido feita. Nesta dissertação este trabalho foi feito, estudando a dinâmica

global de um modelo mais genérico no qual o modelo de AC utilizado no método HCA está incluso. O estudo da dinâmica global do AC é muito importante pois pode fornecer alguns indícios de como aquele modelo se comporta em relação ao tempo: ele pode cair em um padrão uniforme e nunca mais sair, ficar em ciclos (e nesse caso o tamanho do ciclo é importante), gerar resultados caóticos e até mesmo ter comportamentos complexos, como sair de um padrão e entrar rapidamente em modo caótico. A dificuldade deste tipo de estudo fica evidente em [10], onde Culik e Yu demonstram que é indecidível conhecer o comportamento de um AC algoritmicamente.

Dentre as combinações das várias características que definem um modelo genérico de AC, vários submodelos foram criados com finalidades diferentes, e por isso possuem dinâmicas diferentes. Este trabalho estuda a dinâmica de um novo modelo de AC, sendo, dentre suas especificações, finito, híbrido, não-linear que emprega funções sensíveis ao extremo (conhecidas como *toggle rules* [13,25]), e tem em sua principal aplicação o método de criptografia HCA [19].

O modelo investigado nessa dissertação foi chamado de AC Híbrido Sensível Reversível (HSR) (ou *Hybrid Reversible Toggle*) pois ele define as propriedades que um AC híbrido composto exclusivamente de regras sensíveis ao extremo (*toggle rules*) deve possuir para que o modelo seja reversível com o uso de uma condição de contorno periódica. Nesse trabalho, foi caracterizado mais precisamente o papel das funções de transição sensíveis genéricas na reversibilidade do modelo, que equivalem à definição de uma *toggle rule* genérica e que no modelo HCA é representada pela regra principal, e o papel as funções de transição puramente sensíveis, um sub-caso das *toggle rules* no qual a função de transição é exclusivamente sensível a uma célula e que no modelo HCA é representada pela regra de borda. Basicamente, a constatação realizada a partir da análise de reversibilidade do modelo HCA é de que um AC híbrido com reticulado de N células e fronteira periódica será sempre reversível se forem utilizadas N funções de transição de raio R sensíveis a uma mesma célula extrema (à direita ou à esquerda), desde que em pelo menos $2R$ células consecutivas do reticulado sejam utilizadas funções de transição puramente sensíveis. O modelo HCA passa a ser um caso específico do modelo HSR, no qual as $2R$ células consecutivas aplicam uma mesma regra puramente sensível (chamada regra de borda [19]) e as demais $N - 2R$ células do reticulado aplicam uma mesma regra sensível genérica (a chamada regra principal [19]). Dessa forma, toda a análise teórica feita sobre o modelo HSR é plenamente aplicável ao modelo HCA.

Para tal análise, nesta dissertação são investigados dois problemas que se destacam na literatura [37, 38]:

- Problema da Existência do Predecessor (PEP - *Predecessor Existence Problem*). Dados um AC com função global F e uma configuração X, existe Y tal que $F(Y) = X$?

- Problema da Alcançabilidade (CREP - *Configuration REachability Problem*). Dados um AC com função global F, uma configuração fonte X e uma configuração destino Y, existe t tal que $F^t(X) = Y$?

O PEP, como está relacionado à reversibilidade de ACs, está complemente ligado ao desempenho do método, pois assim é possível que um algoritmo $O(N)$ encontre cada pré-imagem, sendo N o tamanho do reticulado do AC. O $CREP$, que também é influenciado pela reversibilidade, está mais relacionado com a topologia do diagrama de transição de estados do AC, que está ligada à segurança do método principalmente quando consideramos os ciclos que formam o diagrama de transição de estados do modelo de AC estudado nessa dissertação.

A investigação de ambos problemas foi feita respondendo sobre a decidibilidade e complexidade de ambos, e analisando como isso pode influenciar na dinâmica global do modelo.

Para o PEP, é sabido que ele é $NLog - Completo$ para ACs finitos unidimensionais, P para ACs infinitos unidimensionais e $NP - Completo$ para ACs com dimensões maiores [38]. Para modelos reversíveis, além do PEP ser decidível a resposta já é conhecida, já que neste tipo de AC para toda configuração X existe uma e somente uma configuração Y tal que $F(Y) = X$. Demonstra-se no Capítulo 3 que o modelo estudado nessa dissertação é reversível.

O $CREP$ é, em geral, $PSPACE - Completo$ [37], e por isso os ACs são considerados, em geral, não previsíveis. Mas a complexidade do $CREP$ pode variar dependendo da função global F. Um AC é chamado de fracamente previsível se $F^t(X)$ pode ser calculado em tempo polinomial, e previsível se $CREP$ é decidível em tempo polinomial. Para ACs fracamente previsíveis, $CREP$ é $NP - Completo$ [37]. Em [9] foram usados conceitos de Sistemas de Prova Interativos, e foi dado um protocolo interativo para o complemento de $CREP$ demonstrando que $CREP \in CoAM_2$, onde AM_K é a classe de complexidade "Arthur-Merlin" definida em [1, 12]. Foi concluído então que $CREP$, para ACs fracamente previsíveis reversíveis não pode ser $NP - Completo$ (a menos que a hierarquia colapse). Utilizando cálculo diferencial booleano, álgebra booleana e álgebra matricial demonstramos também no Capítulo 3 que o modelo estudado nessa dissertação é fracamente previsível, e por ser reversível, $CREP$ não pode ser $NP - Completo$.

O estudo de um modelo de AC mais genérico que o HCA investigado em [19] levou à idelização de um novo método criptográfico, similar ao HCA, mas mais genérico: ao invés de utilizar apenas duas regras, ele pode utilizar até N funções locais, sendo N o tamanho do reticulado. Isto faz com que mesmo no raio 1 o espaço de chaves criptográficas tenha uma cardinalidade alta viabilizando o emprego do mesmo no menor raio possível para se definir uma vizinhança local, o que permite uma implementação mais eficiente.

Definições formais no contexto de ACs e trabalhos anteriores sobre criptografia baseada em ACs se encontram no Capítulo 2. O Capítulo 3 contém uma formalização do modelo

HSR, nossas análises sobre o PEP, $CREP$, e a descrição do novo método proposto. O Capítulo 4 contém nossas conclusões a respeito do trabalho e a última seção contém as referências bibliográficas.

Capítulo 2

Definições e Estado da Arte

2.1 Autômatos Celulares

2.1.1 Definições

Autômatos celulares (ACs) são sistemas dinâmicos totalmente discretos (estado, tempo e espaço). Eles são formados por um reticulado (circular) de máquinas de estados finitas, chamadas células, e uma função de transição. As células estão localizadas nos pontos inteiros do reticulado no espaço Euclidiano d-dimensional. Nos referimos às células por suas coordenadas, isto é, as células são endereçadas pelos elementos de $\mathbb{Z}^d$. O conjunto finito de células (reticulado) evolui sincronamente de acordo com a função (também chamada de regra) local de transição descrita por um autômato finito. O próximo estado de uma célula x depende dos estados dos vizinhos de x, e de seu próprio estado. No caso de ACs unidimensionais, uma das formas de se olhar para os vizinhos é considerando um raio R, onde as células vizinhas são todas à esquerda de x até $x - R$, a própria célula x e todas à direita até $x + R$. Outra forma (mais genérica) é definir um vetor vizinhança, onde cada elemento diz a posição relativa à célula x que deve ser olhada. Para $R = 1$, o vetor vizinhança V seria $(-1, 0, 1)$. Para $R = 2$, $V = (-2, -1, 0, 1, 2)$, e assim sucessivamente. Seja Ω o AC de reticulado L e função de transição local f. Seja $S = \{0, 1\}$ o conjunto de estados possíveis, e a configuração $c = 0110100$ no instante $t = 0$, definindo um reticulado de sete células. Suponha $R = 1$. Assim, o número de vizinhanças possíveis é $|S|^{2R+1} = 8$. Seja $f = 01011011$, representando os bits relativos às saídas das oito vizinhanças possíveis, ordenadas lexicograficamente de forma decrescente. Esta maneira de representar a função de transição se chama tabela de transição, e diz-se que o número de vizinhanças é o tamanho da função. Para cada vizinhança possível temos o novo estado da célula central. A Figura 2.1 mostra todas as vizinhanças possíveis de f e o respectivo estado de saída, que representa o novo estado da célula central.

Para definir o valor de L no próximo passo de tempo devemos aplicar f a todas as células de L de modo sincronizado. A Figura 2.2 mostra como é feita a evolução

111	110	101	100	011	010	001	000
0	1	0	1	1	0	1	1

Figura 2.1: Exemplo de tabela de transição ($f = 01011011$).

passo a passo, desde a configuração inicial $c = 0110100$ em $t = 0$, até a obtenção da configuração $c = 0110011$ em $t = 1$, aplicando-se a função de transição $f = 01011011$, detalhada na Figura 2.1. Quando todas as células utilizam a mesma função local, como acontece no exemplo da Figura 2.2, o AC é dito homogêneo. Caso contrário, é chamado de não-homogêneo ou híbrido. Quando a primeira célula é considerada vizinha da última (fazendo do reticulado uma estrutura circular) o AC é chamado de fronteira periódica. Caso contrário, células nulas são consideradas vizinhas das extremas. Embora a Figura 2.2 apresente o cálculo de uma célula do reticulado L em $t = 1$ de cada vez, é possível obter-se os valores das sete células de uma só vez, aplicando-se a função f em todas as sete células em $t = 0$ de forma síncrona. Por isso, diz-se que o modo de atualização é síncrono. Embora os ACs síncronos sejam mais usuais, existem outros modos de atualização possíveis, como a atualização sequencial, na qual a atualização da 2ª célula do reticulado já leva em consideração o novo estado da 1ª célula e assim por diante [5].

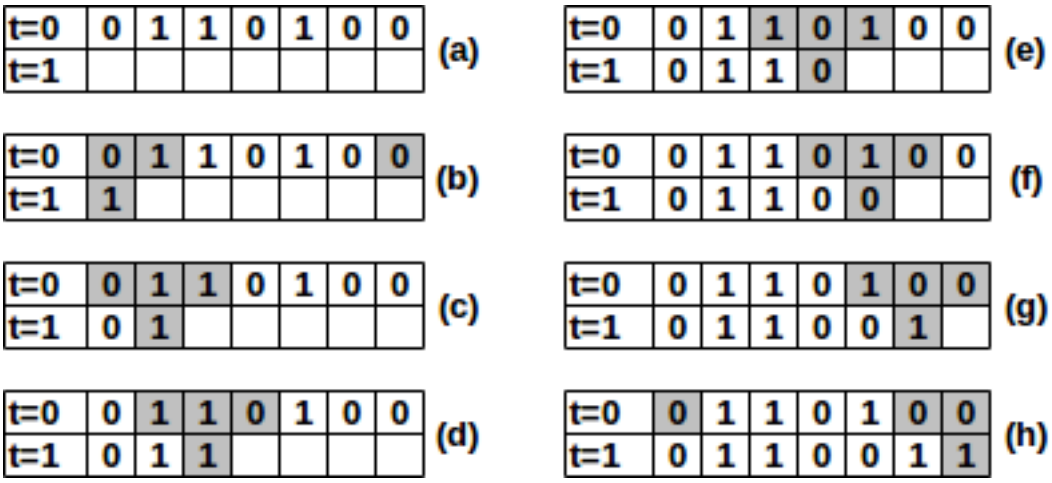

Figura 2.2: Exemplo de um passo de evolução de um AC elementar ($c = 0110100$, $f = 01011011$).

Por simplicidade, durante todo este trabalho, se o contexto do AC analisado não estiver especificado, considere que é um AC elementar: homogêneo com função de transição unidimensional f de raio 1, reticulado com 2 estados por célula e atualização síncrona.

Formalmente, as definições seguem.

Definição 2.1. Um AC é uma tupla (d, S, N, V, f) onde $d \in \mathbb{N}$ é a dimensão do espaço celular, S é o conjunto finito de estados, $N \in \mathbb{N}$ é a quantidade de células por dimensão no

reticulado (quando finito), $V \subseteq \mathbb{Z}^d$ é a vizinhança e $f : S^{|V|} \to S$ é a função de transição local do AC.

Definição 2.2. Uma configuração $c \in S^{N^d}$ é uma coloração do espaço celular por estados. A configuração também pode ser definida como uma função $c : N^d \to S$ onde $c(x)$ é o estado atual da célula x. O conjunto S^{N^d} de todas configurações é denotado por $C(d, S)$, ou simplesmente C uma vez especificados os valores de d e S [17].

A função local é da forma $f : S^{|V|} \to S$. O estado $f(s_1, s_2, \ldots, s_v)$ é o novo estado da célula cujos v vizinhos definidos pelo vetor de vizinhança $V = (x_1, x_2, \ldots, x_v)$ estavam nos estados $s_1, s_2, \ldots, s_v$ um passo de tempo antes. Uma configuração c_1 se torna em um passo de tempo a configuração c_2 onde, $\forall x \in N^d$,

$$c_2(x) = f(c_1(x + x_1), c_1(x + x_2), \ldots, c_1(x + x_v)).$$

A função é definida rigorosamente como segue [17].

Definição 2.3. A função global de transição $F : C \to C$ é definida como:

$$\forall c \in S^{N^d}\, \forall x \in N^d\, ((F(c))(x) = f(c(x + x_1), \ldots, c(x + x_v))), \text{ onde } V = (x_1, \ldots, x_v).$$

Definição 2.4. Um autômato celular híbrido é aquele que permite células diferentes utilizarem funções diferentes. Se utiliza apenas uma função para todas as células, é chamado uniforme ou homogêneo [17].

Geralmente identificamos um AC por sua função de transição global F, e falamos sobre a função F de AC, ou simplesmente AC F. Podemos referenciar F também pela representação decimal do número binário formado pelas saídas das vizinhanças. Por exemplo, podemos dizer que o AC homogêneo que utiliza a função 00111010 é o AC 58. Nessa forma de representação decimal, proposta por Wolfram [45], os bits de saída da função de transição devem estar ordenados pelas vizinhanças em ordem decrescente (111 a 000) e a função de transição é interpretada como um número na base 2 que depois é convertido para a base 10. Quando o AC é híbrido, podemos utilizar a mesma idéia, mas com um vetor de funções. Por exemplo, um AC híbrido com reticulado de quatro células que utiliza a função 00001111 na primeira célula, a função 00000010 na segunda, a função 11001001 na terceira célula e novamente a função 00000010 na quarta célula pode ser representado pelo vetor $< 15, 2, 201, 2 >$.

Definição 2.5. Um autômato celular é dito linear quando suas funções locais (ou a única, se for uniforme) podem ser representadas por operações lógicas $OUEXCLUSIVO(XOR)$ entre as células da vizinhança [22]. Por exemplo, dado o vetor vizinhança $V = (-1, 0, 1)$, a função $f = 01011010$ pode ser representada alternativamente por $\forall s_1, s_2, s_3(f(s_1, s_2, s_3) = s_1 \oplus s_3)$. A Figura 2.3 dá as saídas das vizinhanças para a função f.

111	110	101	100	011	010	001	000
1 ⊕ 1	1 ⊕ 0	1 ⊕ 1	1 ⊕ 0	0 ⊕ 1	0 ⊕ 0	0 ⊕ 1	0 ⊕ 0
0	1	0	1	1	0	1	0

Figura 2.3: Exemplo de função de transição linear (função de transição elementar 90).

Definição 2.6. Um autômato celular é chamado binário quando $|S| = 2$, pois é comum usar $S = \{0, 1\}$ [45]. Neste caso, as células também podem ser chamadas de bits.

Definição 2.7. Uma função de transição local f de um autômato celular binário é dita sensível [13] à célula i quando

$$\forall s_1, \ldots, s_v (f(s_1, s_2, \ldots, s_i, \ldots, s_v) = x \rightarrow f(s_1, s_2, \ldots, \bar{s_i}, \ldots, s_v) = \bar{x}).$$

Ou seja, quando f é sensível à célula i, para uma vizinhança qualquer de f, quando complementarmos apenas a célula i, sua saída será obrigatoriamente invertida. A Figura 2.4 contém um exemplo de função sensível ao terceiro bit, também dita sensível à direita. A Figura 2.4b contém um exemplo de uma função sensível ao primeiro bit, também dita sensível à esquerda. Também é possível definir funções de transição elementares sensíveis ao bit central. Em raios maiores, podemos definir funções de transição sensíveis a qualquer célula de uma vizinhança $(-R, \ldots, -1, 0, +1, \ldots, +R)$, mas nessa dissertação estamos interessados nas funções de transição sensíves às células nos extremos das vizinhanças. Ou seja, as funções de transição sensíveis à direita e sensíveis à esquerda, também conhecidas por *toggle rules* [13].

11**1**	11**0**	10**1**	10**0**	01**1**	01**0**	00**1**	00**0**	(a)
1	0	1	0	0	1	1	0	

111	**1**10	**1**01	**1**00	**0**11	**0**10	**0**01	**0**00	(b)
1	0	1	0	0	1	0	1	

Figura 2.4: Exemplo de função de transição sensível a direita (a) (função de transição elementar 166) e sensível à esquerda (b) (função de transição elementar 165).

Definição 2.8. Uma função de transição local f de autômato celular binário é definida neste trabalho como puramente sensível à célula i se

$$\forall s_1, \ldots, s_v (f(s_1, s_2, \ldots, s_i, \ldots, s_v) = s_i)$$

or

$$\forall s_1, \ldots, s_v (f(s_1, s_2, \ldots, s_i, \ldots, s_v) = \bar{s_i}).$$

Ou seja, se a representação Booleana da função de transição é formada por apenas 1 variável. Ou seja, são os *shifts* e a identidade eventualmente complementados. A Figura 2.5 contém dois exemplos de funções puramente sensíveis ao terceiro bit, ou puramente sensíveis à direita. As funções 11110000 e 00001111 são exemplos de funções puramente sensíveis à esquerda.

111	110	101	100	011	010	001	000
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0

Figura 2.5: Exemplos de funções de transição puramente sensíveis à direita.

Claramente, no espaço de funções de transição unidimensionais binárias de qualquer raio, existem apenas 2 funções de transição puramente sensíveis a cada célula extrema: uma que dá como saída o próprio valor da célula extrema, e outra que dá como saída o complemento dela (pois a função só depende desta célula).

Definição 2.9. Um AC elementar é dito de fronteira nula quando o vizinho à esquerda da célula mais à esquerda e o vizinho à direita da célula mais à direita são considerados 0 [17].

Definição 2.10. Um AC elementar é dito de fronteira periódica quando a célula mais à esquerda é a vizinha esquerda da célula mais à direita e vice-versa [17]. Por exemplo, na evolução do AC da Figura 2.2, a fronteira foi considerada periódica.

Definição 2.11. Uma configuração c_1 do AC de função global F é uma pré-imagem de c_2 se $c_2 = F(c_1)$ [13]. Por exemplo, no AC da Figura 2.2, a configuração 0110100 é uma pré-imagem da configuração 0110011, uma vez que a segunda configuração foi obtida a partir da aplicação da função de transição sobre a primeira.

Definição 2.12. Uma configuração de AC c_1 é dita *Jardim do Éden* se $\nexists c_2 \in C | c_1 = F(c_2)$ [17]. Ou seja, c_1 não possui pré-imagens e esta configuração só pode surgir no reticulado se imposta como configuração inicial e nunca durante a evolução temporal.

Definição 2.13. Um autômato celular é reversível [16] se

$$\forall c_1 \in C(\exists c_2 \in C(c_1 = F(c_2) \wedge \forall c_3 \in C(c_1 = F(c_3) \rightarrow c_2 = c_3))).$$

Ou seja, cada configuração possui uma e apenas uma pré-imagem, portanto F é bijetora.

Definição 2.14. Um Diagrama de Transição de Estados (DTE) de um AC é um dígrafo $G = (V, E)$, sendo $V = C$ e $(X, Y) \in E$ se $F(X) = Y$ [17]. Por exemplo, as Figuras 2.6 e 2.7 apresentam dois exemplos de DTE para duas funções de transição de ACs elementares distintas aplicadas em um reticulado binário de quatro células.

Perceba que se o AC é reversível, o DTE é formado por um conjunto de ciclos desconexos, onde cada ciclo pode ter tamanho (quantidade de estados) $[1, |S|^N]$. Por exemplo, a Figura 2.6 apresenta o DTE de uma função reversível. Note que o DTE é formado por ciclos desconexos.

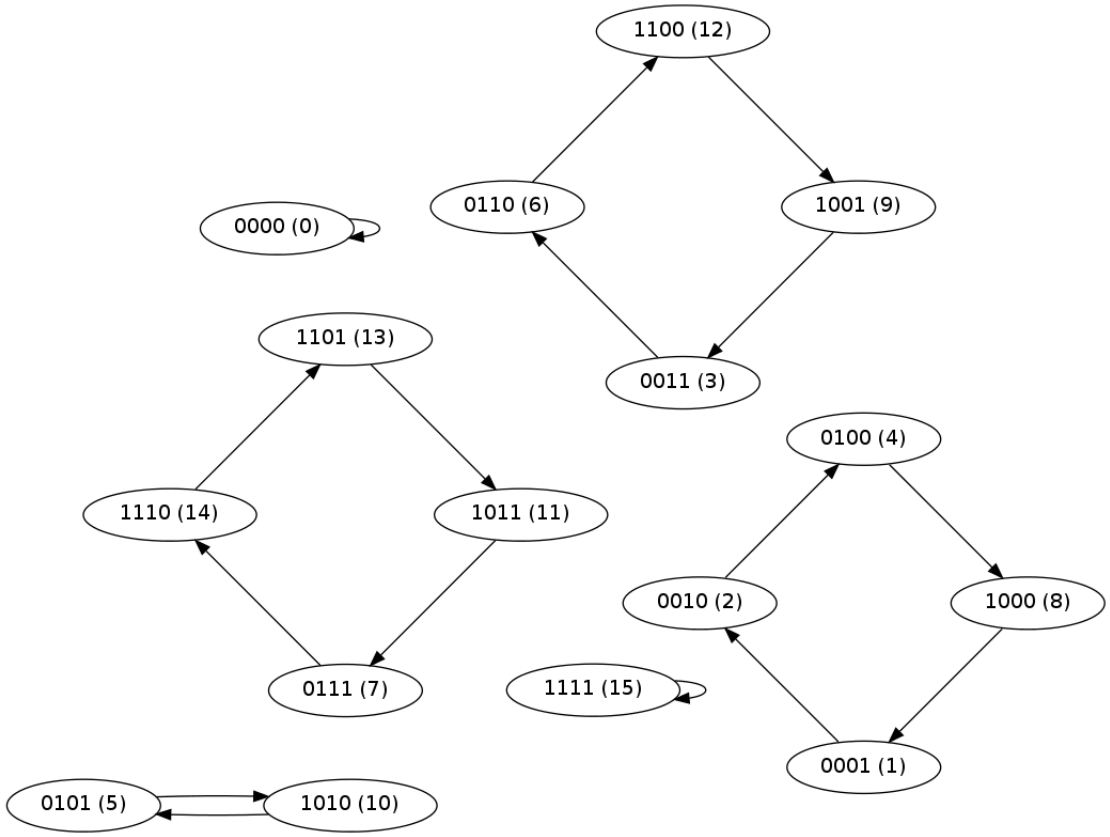

Figura 2.6: DTE da função 01010101 (reversível).

Definição 2.15. Um AC é dito fracamente previsível quando o tempo para calcular $F^t(X)$ é um polinômio de t [9].

2.1.2 Caracterização de ACs Lineares em Álgebra Matricial

Um AC linear (Definição 2.5) de reticulado de N células pode ser representado por uma matriz de transformação $N \times N$, operando sobre $GF(2)$ [7]. A matriz de transformação T é construída de tal forma que:

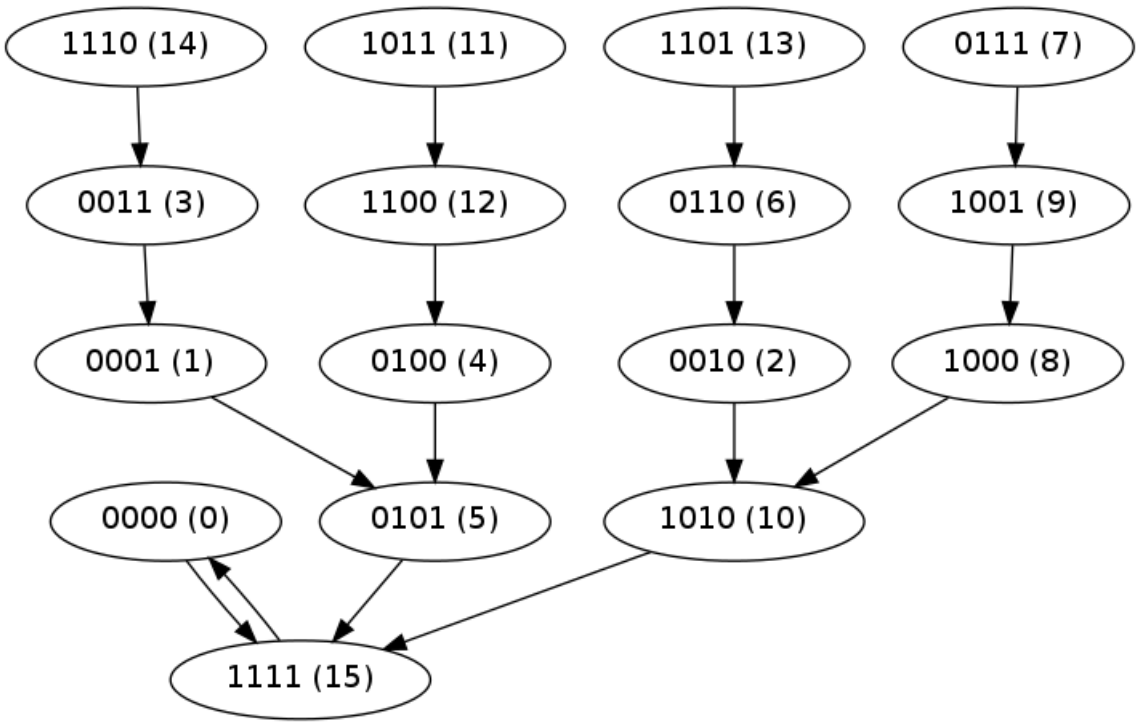

Figura 2.7: DTE da função 10100110 (irreversível).

$$T(i,j) = \begin{cases} 1 & \text{se o próximo estado da célula } i \text{ depende do estado atual da célula } j \\ & \text{na combinação de xor da função } f_i \\ 0 & \text{caso contrário} \end{cases}$$

Exemplo 2.1. Um AC híbrido de fronteira nula com o vetor de funções de transição $< 150, 150, 90, 150 >$ é representado pela seguinte matriz de transformação:

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

O polinômio característico associado pode ser obtido construindo a matriz $T + xI$ e calculando o determinante correspondente.

$$T + xI = \begin{bmatrix} 1+x & 1 & 0 & 0 \\ 1 & 1+x & 1 & 0 \\ 0 & 1 & x & 1 \\ 0 & 0 & 1 & 1+x \end{bmatrix}$$

O polinômio característico é $P(x) = det(T + xI) = 1 + x^3 + x^4$.

A próxima configuração do AC é obtida multiplicando-se a matriz de transformação pelo vetor configuração L (um vetor de tamanho N que representa a configuração do

AC, uma analogia ao reticulado). Seja F a função global do AC. Utilizando a matriz de transformação, temos que:

$$F(X) = T.L(X) \text{ e } F^2(X) = T.F^1(X) = T.T.L(X) = T^2.L(X).$$

Similarmente, após t passos,

$$F^t(X) = T^t.L(X).$$

Exemplo 2.2. Para exemplificar o uso da matriz de transformação na evolução temporal de ACs lineares, vamos calcular o segundo passo da evolução de um AC homogêneo de reticulado de quatro células com configuração inicial $X = 0011$ utilizando a função linear 01100110. Se aplicarmos a função de transição de forma síncrona em todas as 4 células, considerando-se uma fronteira periódica, da mesma forma que foi exemplificada na Figura 2.2, obteremos a próxima configuração que é 0101. Se aplicarmos a função de transição novamente obteremos a configuração 1111. Alternativamente, como esta função de transição é linear, onde $f(s_1, s_2, s_3) = s_2 \oplus s_3$, podemos fazer a evolução por 2 passos de tempo através do cálculo de T^2. Assim, temos que

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Temos também que o vetor que representa a configuração é

$$L(X) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Calculando T^2 chegamos em

$$T^2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

Portanto, temos que

$$F^2(X) = T^2.L(X) = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Desta forma, é possível calcular $F^t(X)$ em tempo $O(log(t))$. Como $t \leq |S|^N$, $O(log(t)) = O(log(|S|^N)) = O(N)$. Portanto, é possível calcular $F^t(X)$ em tempo polinomial, fazendo dos ACs lineares fracamente previsíveis.

Além disso, a caracterização da evolução temporal de ACs lineares através de matrizes de transformação permitiu a utilização de ferramentas de álgebra linear para análise deste tipo de AC [7]. Um dos resultados importantes desse tipo de análise foi a caracterização de modelos de ACs híbridos lineares que possuem reversibilidade com ciclo conhecido a priori e que foram empregados na proposição do método de criptografia [22] discutido na seção 2.2.3.

2.2 Métodos Criptográficos Baseados em ACs

Wolfram foi o primeiro a sugerir o uso de ACs em criptografia [46]. Desde sua idéia, vários estudos têm sido realizados [7, 13, 19, 22, 27–30, 33, 34, 41, 50]. Basicamente, os modelos de criptografia baseados em ACs podem ser divididos em três classes: *(i)* modelos que usam autômatos celulares para gerar seqüências binárias com bom nível de pseudo-aleatoriedade, as quais são usadas como chaves de criptografia, mas a cifragem efetiva é feita por uma outra função [34,46]; *(ii)* modelos baseados em autômatos celulares lineares, híbridos e reversíveis, que utilizam propriedades algébricas deste tipo de regras para gerar autômatos celulares de ciclo máximo e/ou conhecido [7, 33]; *(iii)* modelos baseados no cálculo de pré-imagens de autômatos celulares, os quais usam iterações reversas de ACs no processo de cifragem e iterações para frente para decifrar [13, 19, 27, 28].

2.2.1 AC Geradores de Códigos Pseudo-Aleatórios

Wolfram [46] propôs um modelo que emprega a habilidade do AC em gerar seqüências pseudo-aleatórias para a criação das chaves criptográficas de um tamanho arbitrário a partir de uma chave de tamanho limitado. A idéia principal é utilizar a regra 30, que possui comportamento dinâmico caótico, como gerador da chave secundária, a partir de um reticulado inicial conhecido apenas pelo emissor e receptor (chave primária). Para isso, é necessário evoluir o autômato celular por um número de passos igual ao tamanho da chave secundária desejada. A chave secundária é dada pela evolução temporal de uma única célula do reticulado (cuja posição no reticulado também é conhecida a priori pelo emissor e receptor), equivalente a uma coluna da evolução temporal apresentada na Figura 2.8. Wolfram mostrou que a evolução de uma única célula gera uma sequência pseudoaleatória com as propriedades estatísticas desejadas. Nesse caso, o processo de criptografia propriamente dito pode ser feito por qualquer outro modelo já conhecido como, por exemplo, a submissão da chave e do texto original a sucessivas operações de ou-exclusivo (XOR). Diversos outros métodos refinaram o método original de Wolfram

[15, 34, 42].

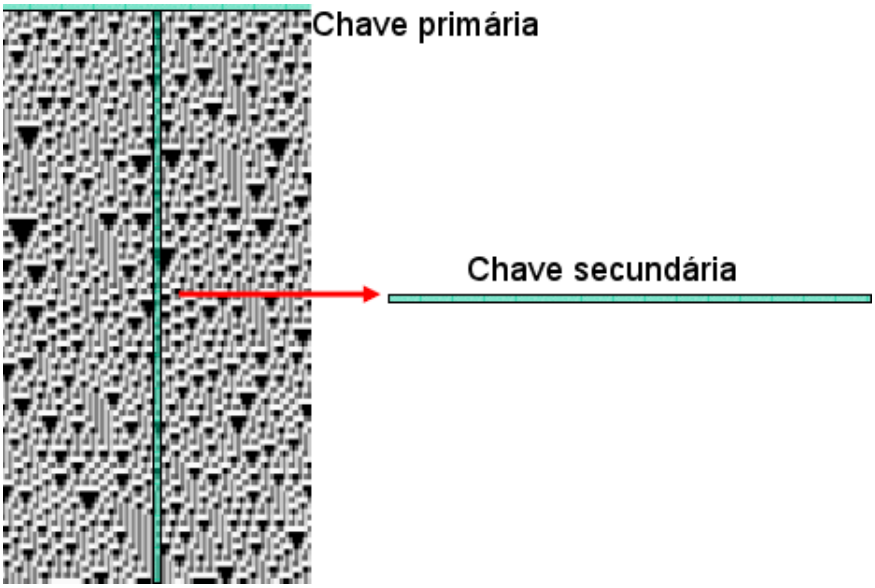

Figura 2.8: Visualização do método criptográfico de Wolfram.

2.2.2 Modelo de Gutowitz

Gutowitz propôs e patenteou o primeiro modelo criptográfico baseado na evolução para trás de ACs irreversíveis [13]. Esse modelo utiliza funções de transição de AC sensíveis (Definição 2.7) a um dos extremos (direita ou esquerda) como chave secreta.

O uso de funções sensíveis possibilita o cálculo de uma pré-imagem de qualquer reticulado a partir de uma pré-imagem parcial aleatória. Uma pré-imagem do reticulado é calculada adicionando R bits extras em cada lado do reticulado, onde R é o raio do AC. Assim, dado um reticulado de N células e uma função de transição sensível à esquerda com raio 1, a pré-imagem resultante terá $N+2$ células. Suponha que dois bits iniciais a e b são escolhidos aleatoriamente para começar o cálculo da pré-imagem e eles são posicionados na borda direita do reticulado. O estado da próxima célula da esquerda (a terceira) é deterministicamente obtida porque as duas únicas transições possíveis são $0ab \rightarrow v$ e $1ab \rightarrow v'$. O valor da primeira célula na configuração inicial é comparado a v e v', e então determina-se o estado da terceira célula da pré-imagem. Com o estado da terceira célula determinado, o próximo passo é determinar o estado da quarta célula e das outras células sucessivamente até completar a pré-imagem inteira. A propriedade da função de transição ser sensível à esquerda garante que as $N+2$ células da pré-imagem podem ser obtidas, passo-a-passo, de um modo determinístico. O mesmo processo pode ser feito para funções sensíveis à direita e neste caso as células iniciais devem ser posicionadas do lado esquerdo.

Uma característica fundamental no modelo de Gutowitz para garantir a existência de uma pré-imagem para qualquer reticulado é o uso de uma fronteira não periódica.

Quando a pré-imagem é obtida, os bits extras não são descartados: eles são incorporados ao reticulado. Por isso, é sempre possível obter uma pré-imagem para qualquer escolha inicial de bits. Como o tamanho inicial de bits é $2R$, é possível obter 2^{2R} pré-imagens diferentes para cada reticulado. Portanto, o AC é irreversível. Este cálculo de pré-imagem é específico para o método de cifragem de Gutowitz e pode ser utilizado somente quando aplicadas funções sensíveis a um dos extremos (esquerda ou direita).

Dados o texto original (configuração inicial do AC) e a chave secreta T (uma função de transição sensível), P pré-imagens são calculadas sucessivamente, começando com bits iniciais aleatórios ($2R$ células) em cada passo. O texto cifrado é dado pela última pré-imagem obtida após P passos. O processo de decifragem é baseado no fato de que o agente receptor conhece tanto T quanto P. A partir do texto cifrado o receptor precisa apenas aplicar para frente a função de transição T por P passos e o reticulado final será o texto original. As Figuras 2.9 e 2.10 ilustram, respectivamente, os processos de cifragem e decifragem do texto 00100101 através do modelo criptográfico de Gutowitz, utilizando-se a regra sensível à direita 01010101.

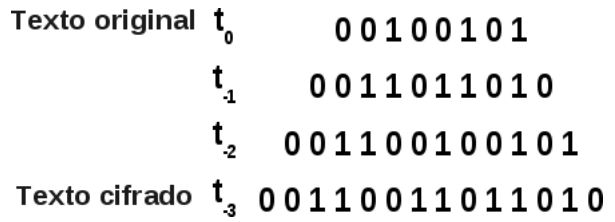

Figura 2.9: Cifragem no modelo de Gutowitz.

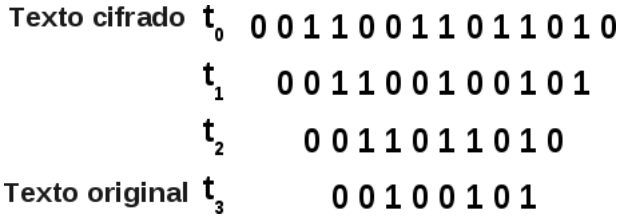

Figura 2.10: Decifragem no modelo de Gutowitz.

Seja P o número de passos de pré-imagem, N o tamanho do reticulado original e R o raio do AC. Como o método adiciona $2R$ bits para cada pré-imagem calculada, o tamanho do texto cifrado (reticulado final) será de $N + (P \text{ x } 2R)$. Por exemplo, se 50 pré-imagens forem calculadas para um texto de 128 bits a partir de uma função de raio 1, o tamanho do texto cifrado resultante será de 228 bits. Claramente, esse incremento é significante, sendo apontado como a grande falha no modelo de Gutowitz.

O modelo de Gutowitz foi refinado em [25] com o uso de regras com sensitividade bidirecional; entretanto, o aumento no tamanho do texto cifrado não foi corrigido. Em [27] e [50] foi avaliada a possibilidade de se usar o cálculo de pré-imagem genérico proposto

em [48] em conjunto com ACs homogêneos e de fronteira periódica para que fosse obtido um método similar ao de Gutowitz, no qual a cifragem é feita pela iteração reversa, porém mantendo o tamanho de pré-imagem igual ao do texto original. O desafio desta abordagem, entretanto, é caracterizar quais regras poderiam ser utilizadas como chaves e que garantissem a existência de pré-imagens para qualquer texto plano. Embora os dois trabalhos tenham sido independentes, chegaram a uma constatação similar: o uso do parâmetro Z [49] poderia ajudar na especificação de tais regras. Entretanto, a principal conclusão em [27] é que a simples utilização do modelo de ACs mais usual (homogêneos e com fronteira periódica) não seria possível de ser adotada em um método de criptografia baseado em cálculo de pré-imagens, visto que as únicas regras com 100% de garantia de inexistência de estados Jardim-do-Éden (Definição 2.12) são regras que promovem um simples deslocamento do reticulado e não possuem dinâmica caótica, sendo impróprias à cifragem.

Assim, dois caminhos foram propostos. O primeiro deles é o uso de um método alternativo que aumentasse o tamanho da pré-imagem apenas quando um estado Jardim-do-Éden fosse identificado. Essa abordagem foi avaliada em [28–30], onde foi possível obter um método viável para a criptografia que, com uma especificação adequada das chaves, retorna um texto cifrado com tamanho bem próximo ao texto original. Entretanto, o tamanho final do texto cifrado é variável. O segundo caminho é a utilização de uma variação do modelo usual de ACs que permitisse a garantia de existência de pré-imagem para qualquer texto plano, com uso de fronteira periódica. Essa segunda abordagem levou à proposição do método HCA, que utiliza um modelo de ACs híbrido que é avaliado nesta dissertação.

2.2.3 Modelo de Nandi e colaboradores

Este modelo [22] é baseado em ACs lineares (Definição 2.5) não-homogêneos reversíveis, com funções específicas tal que o tamanho do ciclo seja conhecido [22]. A idéia chave do modelo é utilizar combinações de funções lineares reversíveis para se conhecer o tamanho do ciclo, e então aplicar a evolução do AC para a frente. Isto pode ser feito porque os ACs lineares possuem propriedades algébricas específicas que permitem que os mesmos sejam caracterizados por uma matriz de transformação T, conforme apresentado na seção 2.1.2.

Utilizando os polinômios gerados pelas matrizes, os criadores deste modelo mostram que é possível calcular a quantidade e o tamanho dos ciclos que formam o DTE do AC. Assim, a parte principal do modelo envolve evoluir a configuração inicial X por t passos de tempo, sendo t a metade do tamanho do ciclo que contém X, chegando numa configuração Y. Para decifrar, basta evoluir Y por mais t passos, chegando de volta em X (dando a volta no ciclo). Como estas evoluções são para frente e o modelo é fracamente

previsível, cifragem e decifragem são extremamente rápidas. Porém, o método falha na segurança, porque é baseado em permutações no espaço de vetores V_N sobre $GF(2)$, e foi criticado em [3]. Segundo os autores, as permutações efetuadas são do tipo *affine*, um pequeno subgrupo no espaço de vetores V_N, onde todas as funções E_i geradas pelo método produzem a propriedade *affine group*, e podem ser quebradas facilmente por um ataque de texto plano escolhido ou um ataque de texto plano conhecido.

2.2.4 Modelo HCA

Este método criptográfico [19] surgiu da necessidade de um modelo que não tivesse as falhas de segurança existentes nos modelos baseados em ACs lineares e também não aumentasse o tamanho do texto cifrado, como nos métodos anteriores baseados no cálculo de pré-imagens.

O processo é feito assim como nos outros modelos baseados em pré-imagens de ACs: a cifragem é feita através da aplicação de P passos consecutivos do cálculo de pré-imagem, e a última pré-imagem gerada é o texto cifrado. A decifragem é feita pela evolução do AC para frente por P passos. A última configuração gerada é o texto decifrado.

Assim como o método de Gutowitz, o HCA utiliza funções de transição sensíveis aos extremos. Entretanto, diferentemente dos métodos anteriores, o modelo do AC é híbrido, pois utiliza duas funções: uma chamada de regra principal e outra regra de borda. Além disso, a fronteira é periódica, o que permite que o tamanho do reticulado seja preservado no cálculo de pré-imagem. A regra principal deve ser sensível ao extremo e é responsável pelo comportamento caótico do AC, fazendo com que a cifragem seja de boa qualidade. Isto foi comprovado em [19] pela análise da entropia do texto cifrado e pelas análises de perturbação de bit, tanto no texto plano quanto na chave criptográfica. A regra de borda é puramente sensível ao mesmo extremo e apresenta uma propriedade que permite que a condição de contorno periódica do reticulado sempre seja satisfeita para encontrar uma pré-imagem, descartando os bits extras. Desta forma, o texto cifrado tem o mesmo tamanho do texto original, corrigindo a principal falha dos modelos apresentados em [13, 25, 28]. Embora seja um método baseado em ACs híbridos, como o método de Nandi e colaboradores [22], as funções de transição não são lineares (Definição 2.5) e portanto não apresentam as falhas de segurança apontadas em [3].

A descoberta de cada pré-imagem é feita da seguinte maneira: como $2R$ células utilizam uma função de transição puramente sensível (regra de borda), $2R$ bits já são conhecidos, pois os estados destas células são determinados apenas pelo bit mais à direita da vizinhança que o gera. Assim, dependendo da regra de borda, basta copiar ou complementar o estado das $2R$ células que utilizam funções puramente sensíveis para as células mais á direita de suas vizinhanças. A Figura 2.11 mostra este processo inicial utilizando-se a função de transição puramente sensível à direita 10101010 (170) para encontrar a

pré-imagem da configuração 1001101 em um AC de Raio 1, sendo que as duas últimas as células do reticulado utilizam a regra de borda.

t=-1	1						0
t=0	1	0	0	1	1	0	1

Figura 2.11: Processo inicial do cálculo de uma pré-imagem no método HCA.

Perceba que quando estas $2R$ células da pré-imagem já foram descobertas, falta apenas um bit para termos a vizinhança completa da próxima célula da configuração que está sendo cifrada. Esta célula utiliza uma função de transição sensível à direita (regra principal), e por isso o estado desta nova célula na pré-imagem é definido de forma determinística: utilizando-se os $2R$ bits iniciais da vizinhança (01), o próximo bit será obtido verificando-se as vizinhanças 010 e 011 na regra principal. Portanto, basta verificar na configuração que está sendo cifrada o estado da célula central da vizinhança (no exemplo, o valor da célula central é 1), e decidir se o bit final da vizinhança que o gera é 0 ou 1. A Figura 2.12 ilustra o cálculo da pré-imagem iniciado na Figura 2.11 para a regra principal 01100101. Nesse caso, vemos que o valor da 3ª célula calculada na pré-imagem é 0 (2.12a), pois a regra principal estabelece 010 $\rightarrow$ 1. O restante da Figura 2.12 ilustra o cálculo dos próximos bits da pré-imagem, sempre utilizando-se a propriedade da sensitividade à direita da regra principal 01100101 para se calcular deterministicamente o valor de cada célula.

t=-1	1	0					0	(a)
t=0	1	0	0	1	1	0	1	

111	110	101	100	011	010	001	000
0	1	1	0	0	1	0	1

t=-1	1	0	0				0	(b)
t=0	1	0	0	1	1	0	1	

111	110	101	100	011	010	001	000
0	1	1	0	0	1	0	1

t=-1	1	0	0	1			0	(c)
t=0	1	0	0	1	1	0	1	

111	110	101	100	011	010	001	000
0	1	1	0	0	1	0	1

t=-1	1	0	0	1	0		0	(d)
t=0	1	0	0	1	1	0	1	

111	110	101	100	011	010	001	000
0	1	1	0	0	1	0	1

t=-1	1	0	0	1	0	1	0	(e)
t=0	1	0	0	1	1	0	1	

111	110	101	100	011	010	001	000
0	1	1	0	0	1	0	1

Figura 2.12: Cálculo final de uma pré-imagem no método HCA.

Em [19], onde este método foi apresentado, foram dadas 3 formas de se executar o cálculo de pré-imagem. A primeira e mais básica consiste simplesmente no cálculo sucessivo de pré-imagens, como explicado anteriormente. A segunda forma conta com um procedimento de rotação da borda a cada novo cálculo de pré-imagem. As células que utilizam a regra de borda são alteradas, e isto permite que os P passos de pré-imagem sejam executados de modo paralelo. A terceira forma possui a característica de evoluir

com regras diferentes a cada passo de execução do cálculo, além de usufruir do mesmo paralelismo da segunda forma. Em [19] foi constatado que os métodos referentes à primeira e à segunda versão eram muito sensíveis a alguns tipos de configurações iniciais, pelo fato de apenas uma regra ser empregada a cada passo do cálculo de pré-imagem, tornando-os vulneráveis a um ataque onte o texto plano é escolhido. Esta fraqueza foi corrigida na terceira forma. As três versões do método HCA foram submetidas ao INPI para solicitação de patente [26]. Posteriormente, versões bidimensionais [20] e tridimensionais [18] do HCA foram investigadas.

Inicialmente, o modelo unidimensional de AC empregado no método HCA foi utilizado como objeto de estudo desta dissertação. As análises dos problemas propostos não incluem a segunda e a terceira formas, pois estas já fazem parte do processo de criptografia em si e não do modelo de AC propriamente dito. Posteriormente, a partir do estudo das propriedades do AC híbrido empregado no HCA, foi possível observar ele se insere em um modelo mais geral de AC híbrido, no qual um reticulado de N células pode empregar duas funções de transição diferentes de borda (considerando-se raio 1) e $N-2$ funções de transição principais diferentes. Esse modelo mais geral foi chamado de HSR (Híbrido Sensível Reversível) e ele é definido mais precisamente na seção 3.1. Os problemas *PEP* e *CREP* serão investigados para este modelo mais geral, mas se aplicam ao modelo HCA, por ser um caso específico do mesmo.

2.3 Problemas Investigados em Modelos de ACs

O objetivo principal deste trabalho é investigar a dinâmica de um sub-modelo de ACs. Dentre as várias formas de se realizar esta investigação, duas delas são os problemas *PEP* e *CREP*. O estudo destes problemas de modo separado é importante, e de modo combinado pode dar bons indícios de como o modelo se comporta de modo geral [37].

2.3.1 PEP - Predecessor Existence Problem

O *PEP* (Problema da Existência do Predecessor) é um dos problemas mais antigos investigados no contexto de autômatos celulares. Especialmente no contexto da criptografia baseada em ACs, o *PEP* é de extrema importância pois relaciona-se ao estudo de reversibilidade dos modelos. Este problema possui o seguinte enunciado:

Dados um autômato celular Ω de função global F e uma configuração X, $\exists Y | F(Y) = X$?

Para modelos reversíveis, a resposta é sempre sim. Mas perceba que este é um problema de decisão, basta responder "sim" ou "não", não sendo necessário mostrar qual Y gera X. Para encontrar Y, no modelo estudado, é dado no Capítulo 3 um algoritmo que diz qual é o predecessor. Também no Capítulo 3 se encontra a demonstração de que o modelo estudado é reversível.

Para modelos irreversíveis, variadas são as complexidades deste problema para os diferentes modelos de ACs [37]. A Tabela 2.1 mostra esta relação para os modelos híbridos (pode-se considerar que o conjunto dos homogêneos é subconjunto do conjunto dos híbridos).

Modelo	Complexidade
1D Finito	NLOG-Completo
$\geq$ 2D Finito	NP-Completo
1D Infinito	P
$\geq$ 2D Infinito	Indecidível

Tabela 2.1: Complexidade do *PEP* dependendo do modelo de AC [38].

A principal característica de ACs reversíveis é a topologia específica do DTE, que é formado por um conjunto de ciclos desconexos, já que cada configuração (vértice) tem exatamente uma aresta de entrada e uma outra de saída.

2.3.2 CREP - Configuration Reachability Problem

O *CREP* (Problema da Alcançabilidade de Configuração) é um problema um pouco mais complexo em relação ao *PEP*, e também não menos importante. Seu enunciado se dá por:

Dados um AC Ω de função global F e duas configurações X e Y, $\exists t | F^t(X) = Y$?

No caso dos ACs reversíveis, resolver o *CREP* corresponde a verificar se as configurações X e Y do enunciado se encontram em um mesmo ciclo no DTE.

A complexidade deste problema varia com o modelo de AC, mas também varia dependendo das características da função de transição F (se F é reversível ou fracamente previsível) [38]. A Tabela 2.2 contém estas relações para os modelos híbridos.

Modelo	Complexidade
1D Finito	PSPACE-Completo
1D Finito, FP*	NP-Completo
1D Finito, R*, FP*	AM e não pode ser NP-Completo
Infinito	Indecidível

Tabela 2.2: Complexidade do *CREP* dependendo do modelo de AC. R* = Reversível. FP* = Fracamente Previsível.

Como é possível perceber na Tabela 2.2, a reversibilidade de um modelo influencia na complexidade do *CREP*.

Decidir a complexidade do *CREP* sobre um modelo de AC diz muito sobre a previsibilidade daquele modelo. Isto é de muita importância, pois em [10] Culik e Yu já mostraram ser indecidível separar formalmente os ACs de acordo com suas dinâmicas.

Em [9], demonstra-se o seguinte teorema:

Teorema 2.1. *[9] Para ACs reversíveis fracamente previsíveis, $CREP \in coAM$, e por isso não pode ser NP-Completo.*

Chegou-se a esse resultado através de uma generalização da análise do *CREP* para ACs reversíveis aditivos, percebendo-se que a aditividade e a reversibilidade do modelo o faziam ser NP não podendo ser NP-Completo). Assim, podemos utilizar este resultado para demonstrar que o *CREP* para o modelo estudado neste trabalho também pertence a esta classe de complexidade, bastando demonstrar que este é reversível e fracamente previsível. O Capítulo 3 contém ambas as demonstrações.

Capítulo 3

Desenvolvimento

3.1 Modelo Investigado: AC Híbrido Sensível Reversível (HSR)

O modelo de AC aqui investigado é finito 1D (reticulado finito unidimensional), híbrido (pode possuir funções diferentes para células diferentes) e não-linear (todas as funções utilizadas são não-lineares). Este modelo também requer que todas as funções utilizadas sejam sensíveis a um dos extremos, e que pelo menos $2R$ células consecutivas utilizem funções puramente sensíveis a este mesmo extremo.

Este modelo é mais genérico em relação ao modelo híbrido utilizado no método HCA [19], pois pode utilizar até N funções diferentes (sendo N o tamanho do reticulado), enquanto o modelo de AC empregado no sistema criptográfico HCA utiliza apenas duas funções: uma sensível ao extremo aplicada em $N - 2R$ células do reticulado e outra puramente sensível ao mesmo extremo, aplicada em $2R$ células.

Formalmente, o modelo híbrido investigado nessa dissertação, doravante chamado de AC Híbrido Sensível Reversível (*Hybrid Reversible Toggle* ou HSR), é assim definido:

Definição 3.1. AC Híbrido Sensível Reversível (HSR) é um AC unidimensional binário de fronteira periódica Ω definido pela tupla (N, D, R, F) onde N é o número de células do reticulado unidimensional, D é a direção da sensitividade das funções de transição (D $\in$ {esquerda, direita}), R é o raio que define a vizinhança das células e F é um vetor de N funções locais de transição do tipo $< f_1, f_2, f_3, \ldots, f_{N-2R}, f_{N-2R+1}, \ldots, f_N >$, de tal forma que cada f_i $(1 \leq i \leq N-2R)$ deve ser uma função de transição de raio R sensível à célula extrema na direção D e f_j $(N-2R+1 \leq j \leq N)$ deve ser uma função de transição de raio R puramente sensível à célula extrema na direção D. Estamos colocando as $2R$ células que utilizam as funções puramente sensíveis no final do reticulado, mas perceba que como o reticulado é circular podemos colocá-las em qualquer posição do reticulado, basta fazer uma rotação no reticulado ou no vetor F.

Por exemplo, como as funções de transição elementares 150, 225, 60 e 165 são sensíveis

à esquerda e as funções de transição 15 e 240 são puramente sensíveis à esquerda, temos que o AC $\Omega = (8, \text{'e'}, 1, F)$ com $F =< 150, 60, 150, 225, 165, 165, 15, 240 >$ é um exemplo de um AC híbrido sensível reversível.

O modelo do AC utilizado no HCA [19] é um caso específico do modelo descrito acima onde $\forall i \forall j (f_i = f_j)$, $1 \leq i, j \leq N - 2R$ e $\forall i \forall j (f_i = f_j)$, $N - 2R + 1 \leq i, j \leq N$.

3.2 PEP - *Predecessor Existence Problem*

Enunciado: Seja Ω um AC Híbrido Sensível Reversível com uma função global F e configuração X, $\exists Y | F(Y) = X$?

Teorema 3.1. *Seja Ω um autômato celular finito, híbrido, de configuração atual c, raio R e reticulado de N células com fronteira periódica, as quais utilizam $K \leq N$ funções sensíveis à direção d. Se pelo menos $2R$ células consecutivas utilizam funções puramente sensíveis à direção d, Ω é reversível.*

Demonstração. Durante toda a demonstração, será utilizada sensitividade à direita, e depois mostrado que o resultado é análogo para sensitividade à esquerda.

Para que Ω seja reversível, é necessário que as duas afirmações a seguir sejam verdadeiras:

A1. Se c tem uma pré-imagem c_1, ela é única.
A2. c tem uma pré-imagem c_1.

Proposição 3.1. (A1) *Se c tem uma pré-imagem c_1, ela é única.*

Demonstração. Suponha que c possui pelo menos duas pré-imagens c_1 e c_2, e portanto Ω não é reversível.

1) Seja $I_0 = [i, j]$ o intervalo das $2R$ células consecutivas que utilizam funções puramente sensíveis no reticulado com configuração c. O intervalo $I_1 = [i + R, j + R]$ corresponde às células da pré-imagem mais à direita nas vizinhanças que calculam o novo estado das células de I_0. Por exemplo, suponha que Ω possui um reticulado de 8 células com configuração atual c sendo que a 4^{a} $(c(i))$ e 5^{a} $(c(j))$ células do reticulado são sujeitas a funções puramente sensíveis à direita de raio 1. Nesse caso, a 5^{a} $(c_1(i + 1))$ e a 6^{a} $(c_1(j + 1))$ células da pré-imagem c_1 definem os valores das células $c(i)$ e $c(j)$ da configuração c. A Figura 3.1 ilustra esse exemplo.

Considerando o caso genérico de um reticulado com configuração c para o qual se deseja encontrar duas pré-imagens diferentes $c_1 =< c_1(1), c_1(2), c_1(3), c_1(4), c_1(5), c_1(6), c_1(7), c_1(8) >$ e $c_2 =< c_2(1), c_2(2), c_2(3), c_2(4), c_2(5), c_2(6), c_2(7), c_2(8) >$, a demonstração será feita tentando construir duas pré-imagens diferentes c_1 e c_2 para uma mesma configuração c. A Figura 3.2 ilustra esse processo.

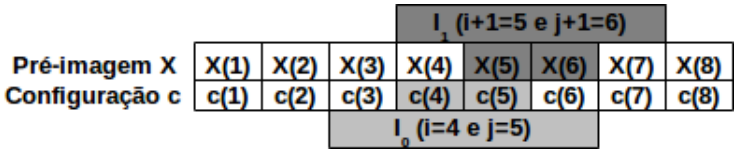

Figura 3.1: Intervalos I_0 e I_1 na configuração c e na pré-imagem X.

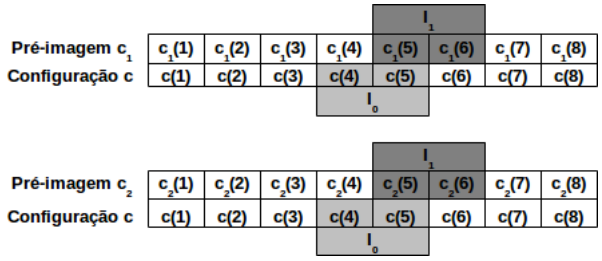

Figura 3.2: Início da construção das pré-imagens c_1 e c_2 a partir de c.

Se c_1 e c_2 são pré-imagems de c, então os estados das células no intervalo I_1 de c_1 são necessariamente os mesmos de c_2 no mesmo intervalo. Claramente nenhuma célula do intervalo I_1 de c_1 pode assumir um estado diferente das células no mesmo intervalo I_1 em c_2, já que as células do intervalo I_0 em c dependem exclusivamente dos estados das células no intervalo I_1 da pré-imagem, seja ela c_1 ou c_2. Assim, se o estado de qualquer célula em I_1 de c_1 for diferente do estado da célula em c_2 na mesma posição, necessariamente o valor de alguma célula de c em I_0 será diferente. Sendo assim, c_1 e c_2 não poderiam ser ambas pré-imagens de c. Este início está representado na Figura 3.2, onde os valores das células em c_2 no intervalo I_1 recebem os mesmos valores das células em c_1 no mesmo intervalo.

2) Uma vez estabelecido em (1) que as células no intervalo I_1 de c_1 e c_2 assumem exatamente os mesmos valores, podemos caminhar na configuração c_1 a partir da célula $i + R - 1$ (primeira à esquerda de I_1) até a célula $j + R + 1$ (primeira à direita de I_1), considerando-se o contorno periódico, para verificar a possibilidade de existir alguma célula de c_1 fora de I_1 com estado diferente da célula em c_2 na mesma posição.

Como todas as células utilizam funções sensíveis à direita, qualquer célula $c_1(k)$ de c_1 fora de I_1 cujo estado for diferente da célula $c_2(k)$, sem nenhuma outra alteração nas células de c_1 em relação a c_2, definiria algum estado diferente de uma célula $c(w)$, pois toda célula k de c_1 é a mais à direita da vizinhança de alguma célula w em c. Ou seja, $\exists k | c_1(k) \neq c_2(k), k \in I_1$. Mas como estas funções não são necessariamente puramente sensíveis, é possível que para algumas vizinhanças a função seja sensível à outras células. Assim, podemos tentar trocar algum estado $c_2(k)$ do intervalo $[i-1-R, i+R-2]$ (a vizinhança em c_2 que atualiza o estado $c(i-1)$, sem a célula mais à direita) para que a célula $c(i-1)$ volte a seu valor original. Esta troca e tentativa de reparo estão mostradas na Figura 3.3b. Mas se fizermos isso, alguma outra célula no intervalo $[i-1-R-R, i+R-2-R]$ de c será

alterada, pela sensitividade à direita, como na Figura 3.3c. Outra célula agora precisa ser alterada, utilizando a mesma idéia, para que o estado da célula 1 ($c(1)$) volte ao original. Escolhemos alterar a célula 8, como mostrado na Figura 3.3d, fazendo a célula 1 voltar ao estado correto, mas deixando a célula 7 em um estado errado. Nesse ponto, podemos alterar as células 6 e 7 para fazer a célula 7 voltar ao estado correto. Mas perceba que a célula 6 está dentro do intervalo I_1, portanto, por *1)* ela não pode ser alterada. Nos resta então alterar a célula 7, como na Figura 3.3e. A célula 7 de c' voltou ao estado correto, mas o estado da célula 6 agora se encontra errado. As duas únicas células de c_2 que poderíamos trocar para resolver este problema estão dentro de I_1, o que impossibilita qualquer ação. Podemos perceber que pela propriedade da sensitividade à direita, toda célula alterada em c_2 para consertar uma em c, vai gerar outro erro em c, sendo necessário alterar alguma das $2R$ células anteriores à última alterada em c_2 (tamanho da vizinhança menos a última célula). Como o tamanho da vizinhança é $2R + 1$, para consertar uma célula alterada pode-se olhar no máximo $2R$ células anteriores. Ou seja, cada passo desse procedimento pode pular no máximo $2R - 1$ células. Por isso, este ciclo acaba quando é alcançado (pela direita) o intervalo I_1. Como o tamanho do intervalo I_1 é $2R$, e não é possível pular $2R$ células, alguma de suas células precisaria ser alterada, o que por (1) é impossível.

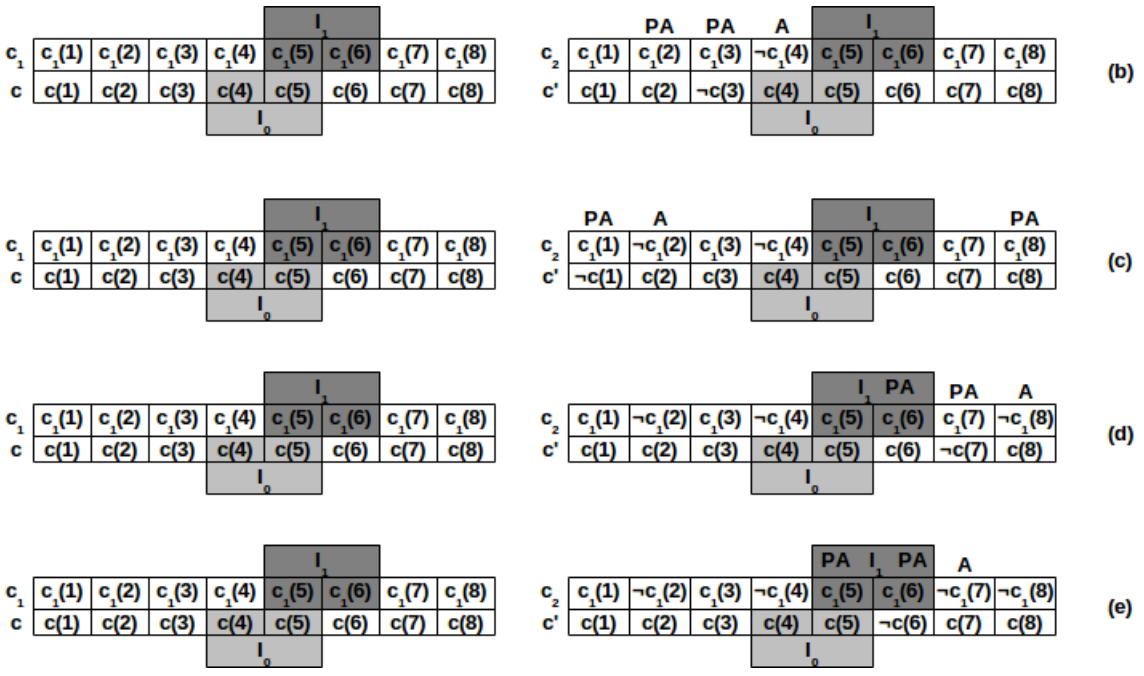

Figura 3.3: Visualização do procedimento que tenta encontrar duas pré-imagens diferentes para a configuração c.

Por contradição, é impossível que c_1 e c_2 sejam diferentes, fazendo com que a pré-imagem de c seja única, caso exista.

Esta mesma demonstração é válida para o caso do outro extremo, a sensitividade à

esquerda. Basta caminhar no sentido contrário, começando em $j + R + 1$ (primeira à direita de I_1) e terminando em $i + R - 1$ (primeira à esquerda de I_1).

Proposição 3.2. (A2) *c tem uma pré-imagem.*

Demonstração. Seja Ω um AC Híbrido Reversível Sensível (com reticulado de N células) sendo f o vetor de funções de transição de raio 1 sensíveis à direita e f_k a função aplicada à k-ésima célula do reticulado. O Algoritmo 1 encontra a pré-imagem X a partir de uma configuração c deterministicamente, mostrando que sempre é possível encontrá-la.

Algorithm 1 Calcula a pré-imagem X de uma configuração c

```
i, j ← intervalo das 2R células consecutivas puramente sensíveis
for k = i → j do
    X(k + 1)modN ← c(k) ⊕ f_k(0, 0, 0)
end for
for k = j + 1 → j + N − 2 do
    w ← kmodN
    if f_k(X(w − 1), X(w), 0) = c(w) then
        X(w + 1) ← 0
    else
        X(w + 1) ← 1
    end if
end for
```

Este algoritmo tem funcionamento similar ao algoritmo de cálculo de pré-imagens do método HCA [19], descrito em linhas gerais na seção 2.2.4. Entretanto, ele se aplica ao caso mais genérico, no qual em um reticulado de N células, até N funções sensíveis diferentes podem ser aplicadas.

O primeiro laço de repetição (*for*) do Algoritmo 1 determina o estado das células no intervalo $I_1 = [i + 1, j + 1]$ da pré-imagem, que se refere às células cujos estados são determinados pelas células da configuração c que utilizam funções puramente sensíveis e que estão no intervalo $I_0 = [i, j]$.

Como duas células (considerando-se $R = 1$) utilizam funções de transição puramente sensíveis, $2R$ bits da pré-imagem já são conhecidos, pois os estados destas células são determinados apenas pelo bit mais à direita da vizinhança que o gera. Assim, para cada célula que utiliza uma função puramente sensível, basta copiar ou complementar seu estado para as células mais à direita de suas vizinhanças. A decisão entre copiar ou complementar é feita analisando-se o primeiro bit da regra local aplicada sobre a célula k do intervalo I_0, ou seja, $f_k(000)$. Se $f_k(000) = 1$ a regra complementa o bit da direita; caso contrário, simplesmente copia o valor do bit da direita.

O segundo laço do Algoritmo 1 determina o estado das células da pré-imagem que estão fora do intervalo I_1, ou seja, das demais $N - 2$ células do reticulado (considerando-se $R = 1$).

Perceba que como duas células da pré-imagem já foram descobertas, falta apenas um bit para termos a vizinhança completa da próxima célula de c. Esta célula utiliza uma função de transição sensível à direita, e por isso o estado desta nova célula na pré-imagem é definido de forma determinística: se os $2R$ bits iniciais da vizinhança são XY, o próximo bit será obtido verificando-se as vizinhanças XY0 e XY1 na função aplicada nesta célula de c. Portanto, basta verificar em c o estado da célula central da vizinhança, e decidir se o bit final da vizinhança que o gera é 0 ou 1. Isto deve ser feito até que a pré-imagem seja inteiramente calculada.

Como o algoritmo consiste basicamente de dois loops que somados resultam em N operações, e as operações dentro dos loops tem tempo de execução constante, temos que este algoritmo é linear no tamanho do reticulado ($O(N)$).

O algoritmo para funções de transição sensíveis à esquerda é análogo, bastando caminhar para a esquerda ao invés da direita.

Como A1 e A2 são verdadeiras, Ω é reversível (Teorema 3.1).

Voltando ao enunciado do *PEP*, como o modelo estudado é reversível, temos que para toda configuração $X \in C$ sempre existe uma e somente uma configuração $Y \in C$ tal que $F(Y) = X$. Portanto, para este modelo de AC, existe uma Máquina de Turing que resolve o Problema do Predecessor, respondendo instantaneamente "Sim".

3.3 CREP - *Configuration Reachability Problem*

Enunciado do problema: seja Ω um AC híbrido sensível reversível de função global F e duas configurações X e Y, $\exists t | F^t(X) = Y$?

Estratégia de demonstração: em [9] foi demonstrado que se um AC for reversível e fracamente previsível, $CREP \in coAM$ (classe de complexidade Arthur-Merlin [1, 12]) e não pode ser NP-Completo. Como já demonstramos que o modelo de AC estudado neste trabalho é reversível, nesta seção será demonstrado que o modelo também é fracamente previsível. Para isto serão utilizadas derivadas parciais de funções booleanas e álgebra matricial.

3.3.1 Classe de Complexidade Arthur-Merlin

A classe de complexidade Arthur-Merlin (AM) é o conjunto de problemas de decisão para os quais uma resposta "sim" pode ser verificada em tempo BPP (*bounded polynomial time*) por um protocolo Arthur-Merlin, assim como problemas de decisão contidos em NP podem ter suas respostas verificadas em tempo P. O protocolo Arthur-Merlin funciona da seguinte maneira: Arthur, um verificador BPP, gera um desafio baseado na entrada do problema, e o envia a Merlin, junto a suas moedas aleatórias a Merlin. Merlin envia

de volta uma resposta, e então Arthur decide se aceita ou não. Dado um algoritmo para Arthur, é requerido que:

- Se a resposta é "sim", Merlin pode agir de tal maneira que Arthur aceita com probabilidade de ao menos 2/3;
- Se a resposta é "não", não importa a maneira como Merlin age, Arthur vai rejeitar com probabilidade de ao menos 1/3.

AM é equivalente à classe $BPNP$ (*bounded probabilistic non-deterministic polynomial*), e possui várias propriedades importantes relacionadas a outras classes de complexidade. Seja AM_k similar a AM, exceto que Arthur e Merlin usem k iterações. Para qualquer constante $k > 2$, $AM_k = AM_2 = AM$ [1]. Em [1] também foi provado que $AM \in PH$, onde PH é a hierarquia polinomial (*polynomial hierarchy*). Em [51], foi provado que se $coNP \subseteq AM_k$ para alguma constante k, então $PH \subseteq AM_k$, isto é, a hierarquia colapsa.

O resultado provado em [51] é forte sobre problemas e suas relações com a classe $NP-Completo$ e a hierarquia polinomial. Em [9], foi provado o teorema que utilizamos neste trabalho: se um AC é reversível e fracamente previsível, $CREP \in coAM$, e como $coAM \subseteq AM$, $CREP$ não pode ser NP-Completo sem que a hierarquia colapse. Seguindo o mesmo princípio, AM também contém o problema do não-isomorfismo, que é o complemento do problema do isomorfismo, um dos problemas mais importantes em aberto na teoria de grafos. O problema do isomorfismo é um problema de decisão que testa se dois grafos são isomórficos. Para o caso especial em que o grafo é uma árvore, este problema está contido em P. Mas para o caso geral, o problema do isomorfismo permanece sendo uma dúvida, ainda não tendo sido provado pertencer a $NP - Completo$ ou P.

As propriedades da classe AM mostram que ela está relacionada a questões fundamentais da teoria de complexidade que não foram resolvidas (como $P = NP$, $P = BPP$, por exemplo), e tem forte impacto em variados problemas, como o isomorfismo de grafos e $CREP$ para autômatos celulares.

A classe de complexidade AM é uma das principais que descrevem um Sistema de Prova Interativo (IPS, Interactive Proof System). A classe ISP contém todos os problemas que podem ser resolvidos por uma máquina abstrata que modela troca de mensagens entre duas entidades, o verificador e o provador, sem restrições de recursos computacionais. No caso de AM, Arthur e Merlin são, respectivamente, o verificador e o provador, e Arthur deve realizar seus cálculos em tempo BPP. O fato de AM ser um IPS a faz um importante protocolo para criptografia pois todos problemas em NP possuem provas de conhecimento zero. Um *sistema de prova de conhecimento zero* (ZKP, Zero Knowledge Proof) é um sistema de prova interativa em que o verificador é convencido de que a resposta está correta, sem nada descobrir a respeito de um certificado para esta solução. Este conceito é importante para segurança de sistemas em aplicações como por exemplo,

autenticação de usuários em um serviço, mesmo sem o serviço conhecer a senha do usuário. Este tipo de aplicação tem grande impacto em sistemas criptográficos, pois permite ao usuário mostrar que é confiável sem abrir mão de seus dados privados.

Mais detalhes sobre classes de complexidade estão em Apêndice A.

3.3.2 Representação das Funções de Transição de ACs na Forma Normal Disjuntiva (FND)

Se considerarmos um AC de raio 1, a função de transição local deste AC terá 3 variáveis: $f(x_{i-1}, x_i, x_{i+1}) = B$. Como o modelo de AC estudado é binário, temos que f é uma função booleana, onde x_{i-1}, x_i e x_{i+1} são variáveis booleanas e B é um valor booleano (0 ou 1). B terá valor 1 quando as variáveis formarem uma vizinhança cuja saída na tabela de transições for 1. Sendo assim,

$$f(x_{i-1}, x_i, x_{i+1}) = \bigvee_{k \in K} (x_{i-1}, x_i, x_{i+1}) = k,$$

onde K é o conjunto de todas vizinhanças cuja saída é 1. Separando as variáveis e os membros de cada vizinhança $k \in K$, temos que

$$f(x_{i-1}, x_i, x_{i+1}) = \bigvee_{k \in K} x_{i-1} = k_0 \wedge x_i = k_1 \wedge x_{i+1} = k_2,$$

fórmula que está na Forma Normal Disjuntiva (FND).

Por exemplo, para a função 00100110, cuja tabela de vizinhanças e saídas está representada na Figura 3.4, a fórmula da função de transição é dada por:

$$f(x_{i-1}, x_i, x_{i+1}) = (x_{i-1} = 0 \wedge x_i = 0 \wedge x_{i+1} = 1) \vee (x_{i-1} = 0 \wedge x_i = 1 \wedge x_{i+1} = 0) \vee (x_{i-1} = 1 \wedge x_i = 0 \wedge x_{i+1} = 1),$$

ou simplesmente,

$$f(x_{i-1}, x_i, x_{i+1}) = (\bar{x_{i-1}} \wedge \bar{x_i} \wedge x_{i+1}) \vee (\bar{x_{i-1}} \wedge x_i \wedge \bar{x_{i+1}}) \vee (x_{i-1} \wedge \bar{x_i} \wedge x_{i+1}).$$

111	110	101	100	011	010	001	000
0	0	1	0	0	1	1	0

Figura 3.4: Tabela de transição da função 00100110.

3.3.3 Representação das Funções de Transição de ACs na Forma Normal Algébrica (FNA)

As funções de transição podem ser convertidas da Forma Normal Disjuntiva (FND) para a Forma Normal Algébrica (FNA).

Derivadas Parciais Booleanas

Seguindo Vichniac [43], definimos a derivada parcial de uma função booleana f em relação a seu *i-ésimo* argumento x_i como

$$\frac{\partial f}{\partial x_i}\Big|x = f(x_1, \ldots, x_i, \ldots, x_n) \oplus f(x_1, \ldots, \bar{x}_i, \ldots, x_n) \tag{3.1}$$

Esta derivada (de primeira ordem) expressa a dependência do *i-ésimo* argumento pela função: $\partial f/\partial x_i$ é 1 se o valor de f é alterado quando o valor de x_i é alterado nos argumentos. Se a derivada de f em relação a x_i é 1, então a regra muda seu valor quando x_i muda. Se ela é 0, a alteração do valor de x_i nunca influencia f. Esta provê uma definição altenativa para as regras sensíveis à célula i (Definição 2.7).

Esta definição de derivada parcial booleana é consistente com as propriedades comuns das derivadas numéricas: a derivada da função identidade é 1, e a derivada de uma constante (0 ou 1) é 0. Além disso, a derivada é linear em relação à operação XOR, e segue a regra padrão para a derivada de um produto,

$$\frac{\partial(f \wedge g)}{\partial x} = \frac{\partial f}{\partial x} \wedge g \oplus f \wedge \frac{\partial g}{\partial x} \tag{3.2}$$

A definição pode ser estendida para derivadas de ordens maiores. Por exemplo, uma derivada de segunda ordem em relação a x_i e x_j é definida como

$$\frac{\partial^2 f}{\partial x_i \partial x_j}\Big|_x = f(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) \oplus f(x_1, \ldots, \bar{x}_i, \ldots, x_j, \ldots, x_n) \oplus \tag{3.3}$$

$$f(x_1, \ldots, x_i, \ldots, \bar{x}_j, \ldots, x_n) \oplus f(x_1, \ldots, \bar{x}_i, \ldots, \bar{x}_j, \ldots, x_n)$$

Note que a definição de derivada booleana também é consistente com a regra da cadeia para derivadas numéricas,

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial x}\right) \tag{3.4}$$

A derivada de segunda ordem em relação ao mesmo argumento é identicamente zero.

Em [2], Bagnoli introduz uma definição mais compacta para as derivadas booleanas. Indicando com x o vetor de *bits* dos argumentos $(x_1, \ldots, x_n)$ e com δ um vetor de *bits* (constante) de mesmo tamanho, ou seja:

$$\partial_\delta f(x) = \bigoplus_{\alpha \leq \delta} f(x \oplus \alpha) \tag{3.5}$$

É imediato verificar que $\partial_\delta f(x)$ é igual a derivada parcial de f em x em relação às variáveis que correspondem aos bits com valor um em δ. Por exemplo, indicando com $\delta^{(i)}$

um *bitarray* no qual apenas o *i-ésimo* bit está setado, temos

$$\partial_{\delta^{(i)}} f(x) = \frac{\partial f}{\partial x_i}\Big|_x \tag{3.6}$$

A partir daí, Bagnoli chega no resultado mais importante, a série de MacLaurin que representa f:

$$f(x) = \bigoplus_{\alpha \in B_n} x^\alpha \wedge f_\alpha, \tag{3.7}$$

onde

$$f_\alpha = \partial_\alpha f(0);$$

O gradiente da função de transição de um AC elementar $f(x_{i-1}, x_i, x_{i+1})$ é definido como um vetor com os elementos sendo as derivadas parciais das variáveis de f:

$$\nabla f = \left(\frac{\partial f}{\partial x_{i-1}}, \frac{\partial f}{\partial x_i}, \frac{\partial f}{\partial x_{i+1}} \right) \tag{3.8}$$

Podemos explicitamente escrever a fórmula para um AC elementar, cuja evolução depende da própria célula (x_i) e seus vizinhos mais próximos (x_{i-1} e x_{i+1}). Localmente,

$$x_i' = f(x_{i-1}, x_i, x_{i+1}) \tag{3.9}$$

onde x_i' indica o futuro valor desta célula. A expansão de MacLaurin de f é dada por

$$\begin{aligned} x_i' = f(0,0,0) \oplus \\ x_{i-1} \wedge \frac{\partial f}{\partial x_{i-1}}\Big|_{0,0,0} \oplus x_i \wedge \frac{\partial f}{\partial x_i}\Big|_{0,0,0} \oplus x_{i+1} \wedge \frac{\partial f}{\partial x_{i+1}}\Big|_{0,0,0} \oplus \\ x_{i-1} \wedge x_i \wedge \frac{\partial f^2}{\partial x_{i-1} \partial x}\Big|_{0,0,0} \oplus x_{i-1} \wedge x_{i+1} \wedge \frac{\partial f^2}{\partial x_{i-1} \partial x_{i+1}}\Big|_{0,0,0} \oplus x_i \wedge x_{i+1} \wedge \frac{\partial f^2}{\partial x_i \partial x_{i+1}}\Big|_{0,0,0} \oplus \\ x_{i-1} \wedge x_i \wedge x_{i+1} \wedge \frac{\partial f^3}{\partial x_{i-1} \partial x \partial x_{i+1}}\Big|_{0,0,0}. \end{aligned} \tag{3.10}$$

Esta expansão pode ser lida da seguinte maneira: como a derivada parcial de f em relação a uma variável x_j representa a sensitividade de f em relação a x_j, podemos considerar $f(0,0,0)$ como um estado inicial e sempre que x_j for 1 ela se torna uma variável potencialmente sensível para trocar o valor calculado até agora (que começou com $f(0,0,0)$). Mas ela só vai ser realmente sensível se sua condição de sensitividade for satisfeita, que é a derivada. Isto precisa ser feito para todas as combinações possíveis de variáveis, para que todas as alterações e potenciais senstividades sejam analisadas. Desta forma, é possível transformar a função de transição da FND para a FNA.

Esta conversão é para ACs elementares genéricos. Mas como estamos considerando ACs com regras sensíveis, nós reduzimos as fórmulas e as simplificamos utilizando dois teoremas de Vichniac em relação a derivadas parciais e regras sensíveis.

Teorema 3.2. *[43] Seja f uma função de transição de um AC binário,*

$$\left(\frac{\partial f}{\partial x_i} = 1\right) \equiv f \text{ é sensível à célula } i. \tag{3.11}$$

Isto acontece porque neste caso qualquer alteração da célula i faz o valor de f também ser alterado.

Teorema 3.3. *[43] Seja f uma função de transição de um AC elementar. Se a derivada parcial de f em relação a uma variável x_i é 1 ou 0, as outras duas variáveis terão no máximo 1 literal.*

Como no caso de ACs elementares a função de transição f é uma função de 3 variáveis, se f é sensível a apenas uma delas, as outras duas variáveis são parcialmente sensíveis, isto é, dependem uma da outra. Por exemplo, seja $f = 01011001$, representada pela tabela de transições da Figura 3.5. O gradiente de f é dado por:

$$\nabla f = (x_i, x_{i-1}^{-}, 1).$$

111	110	101	100	011	010	001	000
0	1	0	1	1	0	0	1

Figura 3.5: Tabela de transição da regra 01011001

Como a função de transição é sensível à célula da direita (x_{i+1}), a derivada parcial de f em relação a x_{i+1} é 1. Podemos perceber que a função só é sensível à célula da esquerda (variável x_{i-1}) quando o estado da célula do meio da vizinhança (x_i) é igual 1. Considerando-se as vizinhanças cujo bit central é 1, quando o bit da esquerda é alterado a saída também é necessariamente modificada. Já a sensitividade à célula do meio ocorre quando o estado da célula da esquerda é igual a 0, como é possível verificar pela derivada parcial de f em relação a x_i.

Como $\dfrac{\partial f}{\partial x_{i+1}} = 1$, qualquer derivada parcial de f de ordem ≥ 2 tal que uma das variáveis seja x_{i+1} será igual a 0, pois

$$\frac{\partial f^2}{\partial x_j \partial x_{i+1}} = \frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial x_{i+1}}\right) = \frac{\partial f'}{\partial x_j} = \tag{3.12}$$

$$= f'(x_1, \ldots, x_j, \ldots, x_n) \oplus f'(x_1, \ldots, \bar{x_j}, \ldots, x_n) = 1 \oplus 1 = 0.$$

Outra simplificação que podemos fazer é: como dito anteriormente, se duas variáveis x_{i-1} e x_i dependem uma da outra para ativar a sensitividade em f, a derivada parcial de primeira ordem de f em relação a x_{i-1} será igual a x_i ou $\bar{x_i}$. De forma análoga, a derivada parcial de primeira ordem de f em relação a x_i será x_{i-1} ou $\bar{x_{i-1}}$ (no máximo um literal). Portanto, a derivada parcial de segunda ordem de f em relação a estas duas variáveis será 1, pois:

$$\frac{\partial f^2}{\partial x_{i-1} \partial x_i} = \frac{\partial}{\partial x_{i-1}} \left(\frac{\partial f}{\partial x_i} \right) = \frac{\partial f'}{\partial x_{i-1}} = \tag{3.13}$$

$$= f'(x_1, \ldots, x_{i-1}, \ldots, x_n) \oplus f'(x_1, \ldots, \bar{x_{i-1}}, \ldots, x_n) = x_i \oplus \bar{x_i} = 1.$$

Desta forma, a expansão de MacLaurin para ACs elementares com funções sensíveis á direita se torna

$$x'_i = f(0,0,0) \oplus \tag{3.14}$$

$$x_{i-1} \wedge \frac{\partial f}{\partial x_{i-1}}\bigg|_{0,0,0} \oplus x_i \wedge \frac{\partial f}{\partial x_i}\bigg|_{0,0,0} \oplus x_{i+1} \wedge 1 \oplus$$

$$x_{i-1} \wedge x_i \wedge 1 \oplus x_{i-1} \wedge x_{i+1} \wedge 0 \oplus x_i \wedge x_{i+1} \wedge 0 \oplus$$

$$x_{i-1} \wedge x_i \wedge x_{i+1} \wedge 0.$$

e pode ser simplificada para

$$x'_i = f(0,0,0) \oplus \tag{3.15}$$

$$x_{i-1} \wedge \frac{\partial f}{\partial x_{i-1}}\bigg|_{0,0,0} \oplus x_i \wedge \frac{\partial f}{\partial x_i}\bigg|_{0,0,0} \oplus x_{i+1} \oplus$$

$$x_{i-1} \wedge x_i$$

3.3.4 Evolução por Álgebra Matricial

Podemos perceber por 3.15 que a função de transição se torna um XOR de 3 partes:

- 1) $f(0,0,0)$;
- 2) uma cláusula para cada variável x_i, considerando a derivada parcial de primeira ordem de f em relação a x_i;
- 3) uma cláusula para cada par de variáveis (x_i, x_j), considerando a derivada parcial de segunda ordem de f em relação a x_i e x_j.

Assim, podemos construir uma matriz quadrada $(N+1) \times (N+1)$, constante (para a função) para cada parte da fórmula, e pela comutatividade da operação XOR, o XOR das partes nos dará o valor de f. A matriz da parte 2 será chamada de J, pois é uma matriz Jacobiana. A matriz da parte 3 será H, pois é uma alteração da matriz Hessiana. A matriz da parte 1 será chamada de Z, pois representa a origem (zero) da função: $f(0,0,0)$. I seria um identificador melhor para ela, mas para evitar qualquer possível confusão com a matriz identidade é melhor não usá-lo.

Podemos utilizar as matrizes Z, J e H para fazer a evolução temporal do AC. Ou seja, dada a configuração atual do AC (X), vamos utilizar essas matrizes para obter F(X).

A configuração X do AC se torna uma matriz coluna R (de reticulado) de $N+1$ linhas, e agora F (a função de transição global do AC) pode ser calculada aplicando-se f a cada linha de R utilizando a notação matricial:

$$F(X) = Z.R \oplus J.R \oplus H.R = (Z \oplus J \oplus H).R \tag{3.16}$$

Por isso, podemos gerar uma única matriz de transformação constante $T = (Z \oplus J \oplus H)$ e simplificar F para

$$F(X) = T.R \tag{3.17}$$

Então, para calcularmos 2 passos a partir de X, fazemos

$$F^2(X) = T.F(X) = T.(T.R) \tag{3.18}$$

Analogamente, para 3 passos,

$$F^3(X) = T.F^2(X) = T.(T.(T.R)) \tag{3.19}$$

Podemos então, pela associatividade da multiplicação de matrizes, dizer que

$$F^t(X) = T^t.R \tag{3.20}$$

Ou seja, a evolução temporal do AC por t passos de tempo, a partir de uma configuração inicial X é dada por 3.20.

A matriz J é montada da seguinte maneira:

$$J(i,j) = \begin{cases} \nabla f_i(k) & \text{se a célula } j \text{ faz parte da vizinhança da célula } i, \\ & \text{sendo } k \text{ a posição de } j \text{ na vizinhança.} \\ 0 & \text{caso contrário} \end{cases} \tag{3.21}$$

Linha e coluna N também ficam zeradas.

A matriz H é montada da seguinte maneira:

$$H(i,j) = \begin{cases} \mid \frac{\partial f}{\partial x_i} \mid & \text{se } i = j, i < N \\ 0 & \text{se } i \neq j \end{cases} \tag{3.22}$$

A notação $|\cdot|$ adotada em 3.22 representa que o operador NOT é ignorado caso $\dfrac{\partial f}{\partial x_i}$ resulte em uma expressão com a negação.

Perceba que linha e coluna N também ficam zeradas na matriz H.

Já a matriz Z é montada desta forma:

$$Z(i,j) = \begin{cases} f_i(0,0,0) & \text{se } j = N \text{ e } i < N \\ 1 & \text{se } i = j = N \\ 0 & \text{caso contrário} \end{cases} \tag{3.23}$$

Perceba que na matriz Z apenas a coluna N é preenchida.

E por fim, o vetor R de $N+1$ linhas que representa uma configuração X fica nesta forma:

$$R(i) = \begin{cases} X(i) & \text{se } i < N \\ 1 & \text{se } i = N \end{cases} \tag{3.24}$$

Segue agora um exemplo onde é utilizado um AC homogêneo de função de transição 10101001, reticulado de 3 células e configuração inicial $X = 001$, onde queremos calcular diretamente o segundo passo da evolução temporal. Para esta função de transição, temos que o gradiente é:

$$\nabla f = (\bar{x_i}, \bar{x_{i-1}}, 1) \tag{3.25}$$

Assim, montamos as 3 matrizes (J, H e Z) de acordo com 3.21, 3.22 e 3.23:

$$J = \begin{bmatrix} \bar{x_2} & 1 & \bar{x_0} & 0 \\ \bar{x_1} & \bar{x_0} & 1 & 0 \\ 1 & \bar{x_2} & \bar{x_1} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.26}$$

$$H = \begin{bmatrix} x_2 & 0 & 0 & 0 \\ 0 & x_0 & 0 & 0 \\ 0 & 0 & x_1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.27}$$

$$Z = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.28}$$

Agora podemos calcular a matriz $T = J \oplus H \oplus Z$:

$$T = \begin{bmatrix} 1 & 1 & \bar{x_0} & 1 \\ \bar{x_1} & 1 & 1 & 1 \\ 1 & \bar{x_2} & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.29}$$

Agora precisamos calcular T^2, pois queremos o segundo passo a partir da configuração 001.

$$T^2 = \begin{bmatrix} (1 \oplus \bar{x_1} \oplus \bar{x_0}) & (\bar{x_0} \wedge \bar{x_2}) & 1 & x_0 \\ 1 & (\bar{x_1} \oplus \bar{x_2} \oplus 1) & (\bar{x_1} \wedge \bar{x_0}) & x_1 \\ (\bar{x_1} \wedge \bar{x_2}) & 1 & (\bar{x_0} \oplus \bar{x_2} \oplus 1) & x_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.30}$$

Basta agora multiplicar T^2 por R, para se obter $F^2(X)$:

$$F^2(X) = \begin{bmatrix} (1 \oplus \bar{x_1} \oplus \bar{x_0}) & (\bar{x_0} \wedge \bar{x_2}) & 1 & x_0 \\ 1 & (\bar{x_1} \oplus \bar{x_2} \oplus 1) & (\bar{x_1} \wedge \bar{x_0}) & x_1 \\ (\bar{x_1} \wedge \bar{x_2}) & 1 & (\bar{x_0} \oplus \bar{x_2} \oplus 1) & x_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \tag{3.31}$$

$$\begin{bmatrix} \bar{x_0} \\ (\bar{x_1} \wedge \bar{x_0}) \oplus x_1 \\ (\bar{x_0} \oplus \bar{x_2} \oplus 1) \oplus x_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Para o resultado, ignoramos a última linha (que serve para o inserir a parte da origem zero da função na fórmula) e temos que $F^2(001) = 111$.

Sabemos que exponenciação de matrizes pode ser calculada em tempo $O(log(t))$, sendo t o expoente, como mostrado no algoritmo *exponentiation by squaring* em [21]. Como t, o número de passos consecutivos de F sobre X que queremos calcular, é menor ou igual a 2^N, que é o maior tamanho possível de um ciclo para um reticulado de tamanho N, temos que o tempo para este cálculo é da ordem de $O(log(2^N)) = O(N)$. Portanto, podemos calcular $F^t(X)$ em tempo polinomial.

Teorema 3.4. *O modelo HSR que utiliza apenas funções sensíveis ao mesmo extremo tal que* $2R$ *células seguidas utilizem funções puramente sensíveis é fracamente previsível.*

Demonstração. O algoritmo *exponentiation by squaring* em [21] mostra como calcular $F^t(X)$ em tempo polinomial para este modelo, o que é suficiente para que ele seja fracamente previsível.

Teorema 3.5. *Para o modelo de AC que utiliza apenas funções sensíveis ao mesmo extremo tal que $2R$ células seguidas utilizem funções puramente sensíveis, $CREP \in AM$ (classe de complexidade Arthur-Merlin [1, 12]) e não pode ser $NP - Completo$.*

Demonstração. Como foi demonstrado que o modelo é reversível e fracamente previsível, utilizando o Teorema 2.1 (demonstrado em [9]) demonstramos este.

3.4 Método Criptográfico com Raio 1 baseado no modelo HSR

A partir da análise dos problemas *PEP* e *CREP* no modelo de AC Híbrido Sensível Reversível surge a idéia de um novo método criptográfico. Este método é similar ao HCA [19], mas ao invés de utilizar apenas duas regras, este método mais genérico pode utilizar quantas funções diferentes forem necessárias, desde que obedeça a definição do modelo na seção 3.1: todas devem ser sensíveis ao mesmo extremo, e pelo menos $2R$ consecutivas devem ser puramente sensíveis. Neste método, a chave criptográfica é formada pelo vetor de funções. Como a chave agora passa a ser uma combinação de $k \leq N$ (onde N é o tamanho do reticulado) funções ao invés de apenas duas funções, o espaço de chaves aumenta consideravalmente para um mesmo raio. Por isso, o novo modelo permite a utilização de funções de transição de AC de menor raio, o que aumenta o desempenho dos processos de cifragem/decifragem.

Para raio 1, existem 16 funções sensíveis à esquerda e 16 à direita. Destas, duas são sensíveis a ambos extremos, e de cada conjunto, duas são puramente sensíveis. A Tabela 3.1 apresenta esta lista. Assim, no HCA, temos 16 opções para a regra principal, uma vez definida a direção da sensitividade que será utilizada. Como é necessário que as funções sejam sensíveis ao mesmo extremo, quando escolhemos uma das funções para a regra principal temos apenas duas opções para a regra de borda. Por isso, o tamanho do espaço de chaves no raio 1 é 30 se definirmos a direção da sensitividade a priori (ou 60 se a direção for definida por um bit a mais na chave). No modelo proposto, temos 16 opções diferentes para cada célula que não utiliza uma função puramente sensível (ou 30 para a primeira escolha se a direção de sensitividade for escolhida pelo bit extra). Após este passo, para manter a sensitividade ao mesmo extremo, temos 2 opções para cada uma das $2R$ células que precisam ter funções puramente sensíveis. Assim, o espaço de chaves é igual a $2^{2R} \times 16^{N-2R} = 4 \times 16^{N-2}$. Para um bloco de 128 bits, o espaço de chaves tem tamanho 4×16^{126} (ou 2^{506}).

A cifragem é feita calculando-se P pré-imagens a partir de uma configuração inicial formada pelo texto a ser cifrado utilizando o algoritmo 1. A última pré-imagem será o texto cifrado.

Núcleo	Esquerda	Direita
0000	01010101 (85)**	00001111 (15)**
0001	01010110 (86)	00011110 (30)
0010	01011001 (89)	00101101 (45)
0011	01011010 (90)	00111100 (60)
0100	01100101 (101)	01001011 (75)
0101	01100110 (102)	01011010 (90)
0110	01101001 (105)	01101001 (105)
0111	01101010 (106)	01111000 (120)
1000	10010101 (149)	10000111 (135)
1001	10010110 (150)*	10010110 (150)*
1010	10011001 (153)	10100101 (165)*
1011	10011010 (154)	10110100 (180)
1100	10100101 (165)*	11000011 (195)
1101	10100110 (166)	11010010 (210)
1110	10101001 (169)	11100001 (225)
1111	10101010 (170)**	11110000 (240)**

Tabela 3.1: Lista de regras sensíveis à direita e à esquerda. * = sensível a ambos extremos. ** = puramente sensível.

A decifragem é feita evoluindo-se o AC por P passos tendo como configuração inicial o texto cifrado. A última configuração gerada representa o texto decifrado.

Capítulo 4

Conclusões

Foi investigado neste trabalho o modelo de autômato celular finito híbrido não-linear que emprega funções com sensitividade ao extremo e a seguinte especificação: todas as funções utilizadas devem ser sensíveis ao mesmo extremo e pelo menos $2R$ células devem ser puramente sensíveis. Esse modelo foi chamado de AC Híbrido Sensível Reversível, ou simplesmente HSR.

Nessa investigação foram estudados os problemas *PEP* e *CREP*. Para o *PEP*, foi demonstrado por meio de fundamentos de ACs e do algoritmo de pré-imagem que o modelo estudado é reversível. Para o *CREP*, utilizando cálculo diferencial booleano, álgebra booleana e álgebra matricial foi possível demonstrar que o problema pertence à classe AM, e não pode ser $NP - Completo$.

Disso podemos concluir que o modelo estudado pode ser considerado "mais" previsível que os modelos apenas reversíveis ou fracamente previsíveis. Claramente, isto o faz também ser "mais" previsível que ACs genéricos. Podemos dizer que ser "mais" previsível - mesmo sem formalmente quantificar esta previsibilidade - é uma grande característica do modelo HSR, já que em [10] foi demonstrado que o problema de classificar um AC por sua dinâmica global é indecidível. Esta maior previsibilidade pode influenciar tanto na segurança quanto no desempenho do HCA, método motivador deste trabalho.

Também foi proposto um novo método de criptografia baseado no cálculo de pré-imagens de ACs com a especificação definida pelo modelo HSR. A vantagem deste método em relação ao que o inspirou é a possibilidade de se utilizar várias funções, e não apenas duas como no HCA. Esta propriedade faz com que o espaço de chaves seja consideravelmente maior, possibilitando maior segurança mesmo com o uso de um raio pequeno (por exemplo, raio 1). A idéia deste modelo parece promissora, por isso uma melhor análise quanto à segurança, desempenho e viabilidade deste método fica como sugestão para trabalhos futuros. Dentro disso, algumas análises importantes relacionadas às funções de transição do raio 1 devem ser feitas, como: i) verificar quais destas regras são viáveis para a criptografia; ii) como elas se comportam quando combinadas umas com as outras; iii) qual o impacto de se utilizar funções de transição lineares como as $2R$ puramente sensí-

veis; iv) qual o impacto de se utilizar funções de transição lineares (algumas poucas do raio 1 são lineares) na combinação de funções que são de fato responsáveis pela cifragem (aplicadas às outras $N - 2R$ células).

Uma outra sugestão para futuras análises tanto sobre o HCA quanto sobre o novo modelo proposto é a análise dos DTE. Embora este problema pareça de dificuldade computacional elevada até mesmo para os ACs lineares, a sensitividade se mostrou uma característica muito importante para a dinâmica global dos ACs.

Referências Bibliográficas

[1] Babai, L. & Moran, S., *Arthur-Merlin games: a randomized proof system and a hierarchy of complexity classes*. Journal of Computer and System Sciences 36, 254-276 1985.

[2] Bagnoli, F., *Boolean derivatives and computation of cellular automata*. International Journal of Modern Physics C, 3, 307-320, 1992.

[3] Blackburn, S.R.; Murphy, S.; & Paterson, K.G., *Comments on "Theory and Applications of Cellular Automata in Cryptography"*. IEEE Transactions on Computers, 46, 637-638, 1997.

[4] Burks, C. & Farmer, D., *Towards modeling DNA sequences as automata*. Physica D, 10, 157-167, 1984.

[5] Carneiro, M.G. & Oliveira, G.M.B., *Synchronous cellular automata scheduler with construction heuristic to static task scheduling in multiprocessors*. GECCO'12, 1433-1434, 2012.

[6] Carneiro, M.G. & Oliveira, G.M.B., *SCAS-IS: Knowledge Extraction and Reuse in Multiprocessor Task Scheduling Based on Cellular Automata*. SBRN'12: 142-147, 2012.

[7] Chaudhuri, P. & Chowdury, D. & Nandi, S. & Chattopadhyay, S. *Additive Cellular Automata: Theory and Applications*. ISBN 0-8186-7717-1, 1997.

[8] Chopard, B. & Droz, M., *Cellular Automata Modelling of Physical Systems*. Cambridge University Press, Cambridge, 1998.

[9] Clementi, A. & Impagliazzo, R., *The Reachability Problem for Finite Cellular Automata*. Journal Information Processing Letters 53(1), 27-31, 1995.

[10] Culik, K.; Yu, S., *Undecidability of CA classification schemes*. Complex Systems 2(2), 177-190, 1988.

[11] Gerola, H. & Seiden P., *Stochastic star formation and spiral structure of galaxies*. Astrophysical Journal, 223, 129-135, 1978.

[12] Goldwasser, S.; Micali, M. & Rackoff, T., *The knowledge complexity of interactive proof systems*. SIAM Journal of Computing 18, 186-208 1987.

[13] Gutowitz, H., *Cryptography with Dynamical Systems*. Cellular Automata and Cooperative Phenomena, 1:237-274, 1995.

[14] Hernández, G. & Herrmann, J.H., *Cellular Automata for Elementary Image Enhancement*. Graphical Models and Image Processing, 58(1), 82-89, 1996.

[15] Hortensius, P.D., McLeod, R.D. & Card, H.C., *Parallel random number generation for VLSI systems using cellular automata*. IEEE Transactions on Computers, 38, 1466-1473, 1989.

[16] Kari, J., *Reversible Cellular Automata*. 2005.

[17] Kari, J., *Theory of cellular automata: A survey*. Theoretical Computer Science, 334(1–3), 3-33, 2005.

[18] Lima, D.A., *Modelo Criptográfico Baseado em Autõmatos Celulares Tridimensionais Híbridos*. Dissertação de Mestrado, Faculdade de Computação, Universidade Federal de Uberlândia, 2012.

[19] Macêdo, H.B., *Um novo método criptográfico baseado no cálculo de pré-imagens de autômatos celulares caóticos, não-homogêneos e não-aditivos*. Dissertação de Mestrado, Faculdade de Computação, Universidade Federal de Uberlândia, 2007.

[20] Magalhães, T.A, *Método criptográfico baseado em autômatos celulares bidimensionais para cifragem de imagens*. Dissertação de Mestrado, Faculdade de Computação, Universidade Federal de Uberlândia, 2010.

[21] Montgomery, P.L., *Speeding the Pollard and Elliptic Curve Methods of Factorization*. Mathematics of Computation, 48, 243-264, 1987.

[22] Nandi, S.; Kar, B. & Chaudhuri, P., *Theory and Applications of Cellular Automata in Cryptography*. IEEE Transactions on Computers, 43:1346-1357, 1994.

[23] Oliveira, G.M.B.; de Oliveira, P.P.B. & Omar N., *Definition and Application of a Five-Parameter Characterization of One-Dimensional Cellular Automata Rule Space*. Artificial Life, 7(3), 277-301, 2001.

[24] Oliveira, G.M.B., *Autômatos Celulares: aspectos dinâmicos e computacionais*. Capítulo de livro, III Jornada de Mini-cursos em Inteligência Artificial (MCIA), Editores: Ricardo de Oliveira Anido e Paulo César Masiero, SBC, 8, 297-345, 2003.

[25] Oliveira, G.M.B.; Coelho, A.R. & Monteiro, L.H.A., *Cellular Automata Cryptographic Model Based on Bi-Directional Toggle Rules*. International Journal of Modern Physics C, 15:1061-1068, 2004.

[26] Oliveira, G.M.B. & Macêdo, H.B., *Sistema criptográfico baseado no cálculo de pré-imagem em autômatos celulares não-homogêneos, não-aditivos e com dinâmica caótica*. Brasil patente PI0703188-2A2, 04/09/2007.

[27] Oliveira, G.M.B.; Macêdo, H.B.; Branquinho, A.A.B. & Lima, M.J.L., *A cryptographic model based on the pre-image calculus of cellular automata*. Automata-2008: Theory and Applications of Cellular Automata, ed: A. Adamatzky, R. Alonso-Sanz, A. Lawniczak, G. Juarez Martinez, K. Morita, T. Worsch, Luniver Press, 139-155, 2008.

[28] Oliveira, G.M.B.; Martins, L.G.A.; Alt, L.S. & Ferreira, G.B., *Investigating a Cellular Automata-Based Cryptographic Model with a Variable-Length Ciphertext*. CSC'10 - International Conference on Scientific Computing, Las Vegas, 2010, 2010.

[29] Oliveira, G.M.B.; Martins, L.G.A.; Alt, L.S. & Ferreira, G.B., *Exhaustive Evaluation of Radius 2 Toggle Rules for a Variable-Length Cellular Automata Cryptographic Model*. International Conference on Cellular Automata for Research and Industry, Ascoli Piceno, Lecture Notes in Computer Science (LNCS), 2010, v. 6350. p. 1-10, 2010.

[30] Oliveira, G.M.B.; Martins, L.G.A;, Ferreira, G.B. & Alt, L.S., *Secret Key Specification for a Variable-Length Cryptographic Cellular Automata-Based Model*. 11th International Conference on Parallel Problem Solving From Nature (PPSN2010), Krakow, Lecture Notes in Computer Science (LNCS), 2010, v. 6239. p. 1-10, 2010.

[31] Oliveira, G.M.B. & Vidica, P.M., *A Coevolutionary Approach to Cellular Automata-Based Task Scheduling*. ACRI'12, 111-120, 2012.

[32] Rosin, P.L., *Training cellular automata for image processing*. IEEE Transactions on Image Processing, 15(7), 2076-2087, 2006.

[33] Sen, S.; Shaw, C.; Chowdhuri, D.; Ganguly, N. & Chaudhuri, P., *Cellular Automata based Cryptosystem (CAC)*. LNCS, 2513: 303-314, 2002.

[34] Seredynski, F.; Bouvry, P. & Zomaya, A.Y., *Secret key cryptography with cellular automata*. Proceedings of Workshop on Nature Inspired Distributed Computing (IPDPS2003: International Parallel & Distributed Processing Symposium), 149-155, 2003.

[35] Smith III, A.R., *Real-time language recognition by one-dimensional cellular automata*. Journal of Computer and System Sciences, 6, 233-253, 1972.

[36] Sommerhalder, R. & van Westrhenen, S.C., *Parallel language recognition in constant time by cellular automata*. Acta Informatica, 19, 397-407, 1983.

[37] Sutner, K., *A note on Culik-Yu classes*. Complex Systems 3(1), 107-115, 1989.

[38] Sutner, K., *On the Computational Complexity of Finite Cellular Automata*. Technical Report, Stevens Institute of Technology, Hoboken, NJ 07030 USA 1989.

[39] Switalski, P., & Seredynski, F., *Multiprocessor scheduling by generalized extremal optimization*. Journal of Scheduling, 13(5), 531-543, 2010.

[40] Switalski, P., & Seredynski, F., *An Effective Multiprocessor Scheduling with Use of Geo Metaheuristic*. International Journal of Foundations of Computer Science 23(2), 465-481, 2012.

[41] Tomassini, M. & Perrenoud, M., *Stream Ciphers with One and Two-Dimensional Cellular Automata*. Proceedings of Parallel Problem Solving from Nature (PPSN VI), LNCS (Springer-Verlag), 1917:722-731, 1986.

[42] Tomassini, M. & Sipper, M., *On the Generation of High-Quality Random Numbers by Two-Dimensional Cellular Automata*. IEEE Transactions on Computers, 49(10), 1140-1151, 2000.

[43] Vichniac, G., *Boolean derivatives on cellular automata*. Physica D, 45, 63-74, 1990.

[44] Vitanni, P., *Sexually reproducing cellular automata*. Mathematical Biosciences, 18, 23-54, 1973.

[45] Wolfram, S., *Theory and Applications of Cellular Automata*. World Scientific, 1986.

[46] Wolfram, S., *Cryptography with cellular automata*. Proceedings of International Cryptology Conference (Crypto'85), LNCS (Springer-Verlag), 218:429-432, 1986.

[47] Wolfram, S., *Random Sequence Generation by Cellular Automata*. Advances in Applied Mathematics, 7: 123-169, 1986.

[48] Wuensche, A. & Lesser, M., *Global Dynamics of Cellular Automata*. ISBN: 0-201-55740-1, New Mexico: Addison-Wesley. 1992.

[49] Wuensche, A., *Classifying Cellular Automata Automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins and the Z parameter*. Complexity, 4(3):47-66. 1998.

[50] Wuensche, A., *Encryption using cellular automata chain-rules*. Automata-2008: Theory and Applications of Cellular Automata, ed: A. Adamatzky, R. Alonso-Sanz, A. Lawniczak, G. Juarez Martinez, K. Morita, T. Worsch, Luniver Press, 126-138, 2008.

[51] Boppana, R.B. & Hastad, J. & Zachos, S., *Does Co-NP have short interactive proofs?*. IPL 49, 127-132, 1987.

Apêndice

A Classes de Complexidade

A Teoria de Complexidade tem como objetivo saber a quantidade de recursos computacionais que um problema necessita para ser resolvido, e classificar os problemas computacionais de acordo com a dificuldade de resolvê-los. Os recursos computacionais mais utilizados nas classes de complexidade são tempo de computação e consumo de memória. Esta classificação é feita na forma de limites inferiores, ou seja, a mínima quantidade de um recurso específico necessária para resolver qualquer instância de certo problema (mesmo no pior caso). Uma classe de complexidade é definida por (i) um modelo computacional; (ii) um recurso computacional; e (iii) um limite inferior. Nesta seção nós usaremos (i) máquina de Turing (determinística e não-determinística); (ii) tempo e memória; e (iii) notação assintótica, ou seja, uma função que representa o crescimento do consumo de dado recurso em relação ao tamanho entrada do problema. Nós focamos esta seção entre classes de complexidade com limite inferior de ordem logarítmica até exponencial, dado que estas são as mais úteis para problemas reais. Abaixo nós definimos as classes fundamentais de tempo e espaço.

- DTIME[$f(n)$]: é a classe de problemas que podem ser resolvidos por máquinas de Turing determinísticas em tempo $f(n)$.
- NTIME[$f(n)$]: é a classe de problemas que podem ser resolvidos por máquinas de Turing não-determinísticas em tempo $f(n)$.
- DSPACE[$f(n)$]: é a classe de problemas que podem ser resolvidos por máquinas de Turing determinísticas com consumo de memória $f(n)$.
- NSPACE[$f(n)$]: é a classe de problemas que podem ser resolvidos por máquinas de Turing não-determinísticas com consumo de memória $f(n)$.

Nós abreviamos DTIME[$f(n)$] para DTIME[f] e NTIME[$f(n)$] para NTIME[f] quando está claro no contexto que f é uma função, e quando não há referências ao tamanho da entrada n.

As classes de complexidade consideradas canônicas são dadas abaixo.

- L = DSPACE[$logn$] (espaço logarítmico determinístico)
- NL = NSPACE[$logn$] (espaço logarítmico não-determinístico)
- P = DTIME[$n^{O(1)}$] = $\bigcup_{k\geq 1}$ DTIME[n^k] (tempo polinomial determinístico)
- NP = NTIME[$n^{O(1)}$] = $\bigcup_{k\geq 1}$ NTIME[n^k] (tempo polinomial não-determinístico)
- PSPACE = DSPACE[$n^{O(1)}$] = $\bigcup_{k\geq 1}$ DSPACE[n^k] (espaço polinomial)
- EXP = DTIME[$2^{n^{O(1)}}$] = $\bigcup_{k\geq 1}$ DTIME[2^{n^k}] (tempo exponencial determinístico)
- NEXP = NTIME[$2^{n^{O(1)}}$] = $\bigcup_{k\geq 1}$ NTIME[2^{n^k}] (tempo exponencial não-determinístico)

- EXPSPACE = DSPACE[$2^{n^{O(1)}}$] = $\bigcup_{k\geq 1}$ DSPACE[2^{n^k}] (espaço exponencial)

A classe P contém vários problemas comuns que podem ser resolvidos de maneira eficiente, como encontrar o menor caminho em um grafo, ordenação, multiplicação de matrizes, entre outros. Por definição, P contém todos os problemas que podem ser resolvidos por algoritmos que possuam complexidade razoável para o pior caso.

A classe NP pode ser definida de maneira diferente. NP contém problemas cujas soluções podem ser verificadas de forma rápida, por máquinas de Turing determinísticas em tempo polinomial. Por exemplo, um problema que está em NP é o do Circuito Hamiltoniano. O problema do Circuito Hamiltoniano visa, dado um grafo de V vértices, encontrar um caminho que comece em um vértice arbitrário v, visite todos os outros vértices uma única vez, e volte a v. Veja que, dado um caminho como uma possível solução, é possível rapidamente verificar se esta solução está correta: basta verificar se (i) o caminho visita todos os vértices do grafo unicamente, exceto v; (ii) o caminho começa e termina no mesmo vértice v; (iii) todas as arestas deste caminho existem no grafo dado como entrada.

Um outro problema importante que pertence à classe NP é o problema da satisfabilidade (SAT). Uma fórmula Booleana ϕ é satisfatível se é possível atribuir verdadeiro ($\top$) ou falso ($\bot$) para cara variável tal que o valor de ϕ é $\top$. Por exemplo, a fórmula $p \wedge (\neg p \vee \neg q)$ é satisfatível para $p = \top$ e $q = \bot$. Já a fórmula $p \wedge \neg q \wedge (\neg p \vee q)$ não é satisfatível. Perceba que uma solução para este problema pode ser verificada por uma máquina de Turing determinística tem tempo polinomial, enquanto resolver o problema em tempo polinomial requer uma máquina de Turing não-determinística. A diferença entre NP e P é a diferença entre resolver e verificar, ou entre encontrar uma prova de um teorema matemático e testar se uma prova candidata está correta. Essencialmente, NP representa todos os conjuntos com provas consideradas pequenas, enquanto P representa afirmações que podem ser provadas ou refutadas rapidamente.

Há um grande interesse geral nas classes P e NP, principalmente pela noção de completude de uma classe. Dada uma classe de complexidade C, esta classe possui um subconjunto $C - Completo$ tal que para quaisquer problemas de decisão p contido em $C - Completo$ e p' contido em C, p é redutível a p'. Um problema de decisão p é redutível a outro problema de decisão p' se para qualquer entrada s que p responde "sim", s pode ser transformado em s' em tempo polinomial tal que p' responde "sim" para s'; e para qualquer entrada n que p responde "não", n pode ser transformado em n' em tempo polinomial tal que p' responde "não" para n'. Os problemas contido no subconjunto $C - Completo$ de uma classe de complexidade C são considerados os mais difíceis daquela classe.

O subconjunto $NP - Completo$ da classe de complexidade NP contém vários problemas importantes para aplicações reais, como SAT, Circuito Hamiltoniano, Cobertura de Vértices, Cobertura de Conjuntos, Caixeiro Viajante, Problema da Mochila, entre outros. O subconjunto $NP \backslash NP - Completo$, dos problemas contidos em NP mas não em $NP - Completo$, é uma grande incógnita. Não existem problemas que foram provados

estar nesse subconjunto, dado que geralmente, quando surge um candidato a esta sub-classe, ele é eventualmente provado estar em P ou $NP-Completo$. Esta sub-classe já teve problemas importantes como candidatos, como o Teste da Primalidade. Atualmente, o problema mais importante que ainda não tem sua classe de complexidade completamente definida é o Isomorfismo de Grafos.

Este trabalho também possui interesse especial nas classes P e NP, já que a Seção 3.3 introduz e usa a classe de complexidade Arthur-Merlin (AM) para discutir a complexidade do problema de alcançabilidade de autômatos celulares e o posicionar em relação a NP, um dos focos desta dissertação.