UNIVERSIDADE PRESBITERIANA MACKENZIE
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA ELÉTRICA E COMPUTAÇÃO

Zorandir Soares

Diferença entre Templates de Autômatos Celulares Unidimensionais Binários

Orientador: Prof. Dr. Pedro Paulo Balbi de Oliveira

São Paulo
2016

UNIVERSIDADE PRESBITERIANA MACKENZIE
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA ELÉTRICA E COMPUTAÇÃO

Zorandir Soares

Diferença entre Templates de Autômatos Celulares Unidimensionais Binários

Texto de dissertação apresentado ao Programa de Pós-Graduação em Engenharia Elétrica e Computação da Universidade Presbiteriana Mackenzie como requisito das exigências para obtenção do título de Mestre em Engenharia Elétrica e Computação.

Orientador: Prof. Dr. Pedro Paulo Balbi de Oliveira

São Paulo
2016

S676d Soares, Zorandir
Diferença entre templates de autômatos celulares unidimensionais binários / Zorandir Soares – 2016.
43f.: il., 30 cm

Dissertação (Mestrado em Engenharia Elétrica e Computação) – Universidade Presbiteriana Mackenzie, São Paulo, 2016.
Orientação: Prof. Dr. Pedro Paulo Balbi de Oliveira
Bibliografia: f. 42-43

1. Autômatos celulares. 2. Templates. 3. Diferença entre templates. 4. Templates de exceção. 5. Propriedades estáticas. I. Título.

CDD 621.3

AGRADECIMENTOS

Gostaria de agradecer muito os meus familiares por todo o suporte que foi me dado. Um agradecimento especial dedico à minha mãe e às minhas irmãs, vocês foram a minha base e inspiração. Também agradeço minha amada esposa Nathalia por toda compreensão e todo auxílio durante o período desse projeto.

Agradeço ao Maurício Verardo por todo apoio no desenvolvimento dessa dissertação.

Agradeço aos professores da Universidade Presbiteriana Mackenzie e à todos os membros do Laboratório de Computação Natural por toda troca de conhecimento e experiência. Meus momentos passados com vocês foram valorosos e divertidos.

Meu muito obrigado ao meu orientador, Prof. Dr. Pedro Paulo Balbi de Oliveira, por todas as críticas e sugestões nesse trabalho. Foi um prazer trabalhar com você.

Por fim, agradeço à CAPES pela bolsa de mestrado concedida, ao Mack-Pesquisa - Fundo Mackenzie de Pesquisa, à Universidade Presbiteriana Mackenzie e ao Programa de Pós-Graduação em Engenharia Elétrica e Computação.

RESUMO

Templates são representações formais para conjuntos de autômatos celulares unidimensionais feitas por meio da generalização das tabelas de transição clássicas. Já existem algoritmos na literatura que geram templates para propriedades estáticas de autômatos celulares, assim como há algoritmos que realizam operações como intersecção entre templates e expansão de template. Aqui, é introduzida a operação de templates de exceção, a operação de diferença entre templates, e explica-se o funcionamento do algoritmo dessas operações, que foram implementadas na biblioteca `CATemplates` do software *Mathematica*. Também discutimos a possibilidade de uso de templates no contexto do problema de paridade (a saber, a determinação da paridade de 1s em uma configuração binária cíclica de tamanho ímpar), com o apoio da operação de diferença entre templates, e das operações geradoras de templates de autômatos celulares conservativos de paridade e conservativos de estado.

Palavras-chave: *Autômatos celulares, templates, diferença entre templates, templates de exceção, propriedades estáticas.*

ABSTRACT

Templates are formal representations for sets of one-dimensional cellular automata created by means of a generalisation of the classical state transition tables. Algorithms already do exist in the literature that generate templates for static properties of cellular automata rules, as well as others that perform operations such as intersection between templates and template expansion. Here, we introduce the exception template operation, the operation of difference between templates, and explain the functioning of the algorithm of those operations, which have been implemented in the `CATemplates` package of the *Mathematica* software. We also discuss the possibility of using templates in the context of the parity problem –namely, the determination of the parity of 1s in a cyclic binary configuration of odd length– with the support of the operation of difference between templates, and of the template generation of parity conserving and state conserving cellular automata.

Keywords: *Cellular automata, templates, difference between templates, exception template, static properties.*

Sumário

1 INTRODUÇÃO

Autômatos celulares (ACs) são sistemas dinâmicos discretos em tempo, espaço e variáveis de estado, cuja dinâmica tem sido extensivamente estudada e aplicada em diversas áreas. ACs operam por meio de regras de ação local, e têm a capacidade de gerar comportamentos globais complexos, mesmo com regras locais simples. ACs são também considerados uma idealização discreta de equações diferenciais parciais utilizadas para descrever sistemas naturais (WOLFRAM, 1994).

Existem diversas famílias de autômatos celulares estudadas. Devido ao rápido crescimento das famílias de ACs, de acordo com a variação dos parâmetros de tamanho da vizinhança e quantidade de estados, uma das famílias mais estudadas é a do espaço elementar, por possuir apenas 256 regras.

Há casos em que os estudos de ACs concentram-se em algum comportamento obtido através de restrições aplicadas às tabelas de transição. Os ACs confinados, em que toda transição de estado leva a um valor presente na vizinhança (THEYSSIER, 2004), são um exemplo do uso da técnica. Esses comportamentos e propriedades obtidos através de restrições aplicadas à tabela de transições são denominados como propriedades estáticas (VERARDO, 2014).

Propriedades estáticas permitem prever determinados comportamentos de um AC sem consultar sua evolução espaço-temporal, ou seja, dispensando a simulação do sistema. Propriedades estáticas também podem ser descritas como indicadores de comportamento de uma determinada família de ACs. Um exemplo de propriedade estática, descrita posteriormente em mais detalhes, é a conservabilidade de paridade. A conservabilidade de paridade define um tipo de AC binário que mantêm o número de estados com valor 1 sempre com a mesma paridade.

A representação de propriedades estáticas é crucial pois culmina na eliminação da necessidade de se buscar uma propriedade analisando todo o espaço de um AC. Recentemente foi estabelecida uma representação formal para conjuntos de ACs denominada *Templates* (DE OLIVEIRA; VERARDO, 2014a; DE OLIVEIRA; VERARDO, 2014b). Para se trabalhar com *Templates* foi criada a biblioteca *CATemplates* (VERARDO; DE OLIVEIRA, 2015), desenvolvida na linguagem do software *Wolfram Mathematica* (WOLFRAM RESEARCH, 2015).

Templates têm a capacidade de representar conjuntos de ACs que compartilham de-

terminada propriedade estática; isso evita a necessidade de se operar uma busca em todo espaço original do AC, quando se deseja encontrar representantes dessa classe. Essa capacidade dos templates é o principal motivador deste trabalho, visto que essa habilidade é muito importante para a resolução de diversos problemas que buscam por regras com algum comportamento específico.

1.1 Objetivo

Aqui tem-se como objetivo principal apresentar a operação de diferença entre templates e a operação geradora de templates de exceção. Ademais, esse projeto se orientou por apresentar um exemplo da utilidade de templates em um problema típico de autômatos celulares, o problema da paridade em que, considerando autômatos celulares unidimensionais binários e com condição de contorno periódica, dada uma configuração inicial com um número ímpar de 1s, o reticulado deve convergir para uma configuração de apenas 1s; caso contrário, ele deve convergir para tudo 0.

1.2 Organização do Documento

Este documento está organizado da seguinte forma: no Capítulo 2 são detalhados os ACs, assim como algumas de suas propriedades. O Capítulo 3 apresenta a definição de algumas propriedades estáticas já implementadas no *CATemplates*. O Capítulo 4 apresenta em mais detalhes o funcionamento dos templates, assim como descreve os funcionamentos de duas de suas principais operações. O Capítulo 5 explica o funcionamento dos algoritmos geradores de templates para determinadas propriedades estáticas. O Capítulo 6 apresenta os resultados dessa pesquisa. O Capítulo 7 apresenta os testes feitos para a operação de diferença e mostra exemplos de aplicações dos templates e suas operações no problema de paridade. Por fim, o Capítulo 8 apresenta as considerações finais do presente trabalho.

2 AUTÔMATOS CELULARES

Autômatos celulares (ACs) são idealizações matemáticas simples dos sistemas naturais. Tipicamente, eles consistem de um reticulado de campos discretos idênticos, onde cada campo pode assumir um conjunto finito de valores inteiros. Os valores dos campos evoluem em tempo discreto de acordo com regras (usualmente determinísticas) que especificam o valor de cada campo de acordo com os campos de suas respectivas vizinhanças (WOLFRAM, 1994).

Autômatos celulares podem operar com reticulados em qualquer número de dimensões. Os primeiros ACs eram bidimensionais e foram criados por von Neumann e Burks (1966) para serem usados como um modelo formal de auto-reprodução de sistemas biológicos. Outro conhecido AC bidimensional é o "Jogo da Vida" (ou "Game of Life"), criado por John Conway, que fez sua primeira aparição em uma coluna de jogos matemáticos (GARDNER, 1970). Entre os ACs unidimensionais os mais conhecidos são os do espaço elementar, que foram sistematicamente estudados por Wolfram (1983).

Independente da dimensionalidade do reticulado, é necessário definir como o AC se comportará nas bordas. Um tratamento típico é a aplicação da condição de contorno periódica nas extremidades. Esse tratamento considera reticulados unidimensionais como um anel, como pode ser visualizado na Figura 1, e considera reticulados bidimensionais como um toroide, que pode ser visualizado na Figura 2.

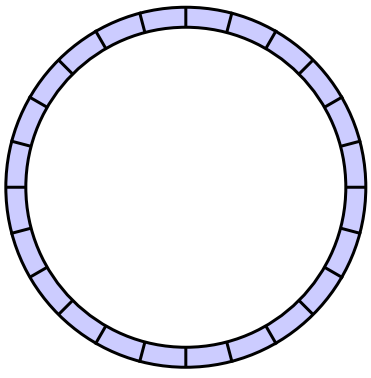


Figura 1: Condição de contorno periódica em um reticulado unidimensional formando um anel.

As células de um autômato celular podem apresentar k estados. O valor desses estados é representado por valores inteiros no intervalo $[0, k-1]$, ou por cores. O estado de uma célula pode ser modificado pelas funções locais, que determinam o novo valor de uma célula baseado em seu estado atual e nos estados das células adjacentes. Para que as funções

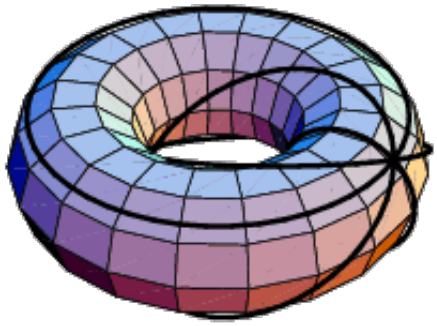


Figura 2: Condição de contorno periódica em um reticulado bidimensional formando um toroide.

locais atualizem os valores de uma célula, é necessário que um raio r seja definido. Esse raio r, por meio da fórmula $2r + 1$, representa o tamanho da vizinhança serão analisadas pelas funções locais. Para $r = 0{,}5$, por exemplo, a vizinhança análisada terá duas células.

Além do raio, é preciso determinar o formato de vizinhança que será utilizada nos parâmetros da função local. Duas vizinhanças bem comuns em ACs bidimensionais são as vizinhanças de von Neumann (WEISSTEIN, 2015c) e Moore (WEISSTEIN, 2015b). Na Figura 3 e na Figura 4 são apresentadas, para raios de 0 a 3, as vizinhanças de von Neumann e Moore, respectivamente. No caso dos ACs unidimensionais, as vizinhanças são definidas apenas pelo raio r e ele descreve quantas células à esquerda e direita da célula atual serão consideradas pela função local.

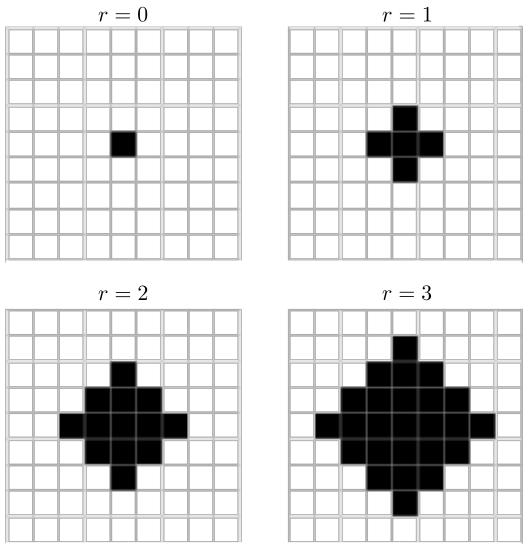


Figura 3: Vizinhança de von Neumann com raio r igual a 0, 1, 2 e 3. Essa foi a vizinhança utilizada nos primeiros trabalhos de von Neumann (WEISSTEIN, 2015c).

Uma família de autômatos celulares é definida pelo raio e pelo número de estados. Autômatos celulares unidimensionais de $r = 1$ e $k = 2$ são conhecidos como a família dos

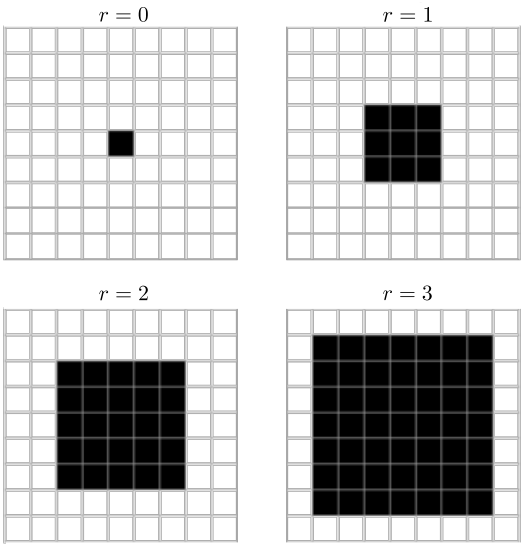


Figura 4: Vizinhança de Moore com raio r igual a 0, 1, 2 e 3. Essa é a vizinhança utilizada no Jogo da Vida (WEISSTEIN, 2015b).

autômatos celulares elementares.

Todo AC é regido por um conjunto de regras locais que determinam como ficarão as configurações no próximo passo de tempo de acordo com as configurações de vizinhança recebidas. Existem diversas maneiras de representar essas regras locais, sendo que a mais comum são as tabelas de transições. A tabela de transições é uma tupla em que os elementos são todas as possíveis configurações de estado das vizinhanças de uma célula, acrescidos de um estado que representa a transição que ocorrerá. O primeiro item da tupla apresenta uma vizinhança formada por todas as células no estado k–1, e o último item apresenta uma vizinhança formada apenas por estados 0, sendo essa a ordenação lexicográfica de Wolfram. Cada regra também apresenta um número de identificação obtido ao se converter o resultado das transições de estados das tabelas para decimal. A Eq. (1) representa a tupla da regra 30.

$$\begin{aligned}(((1,1,1),0),((1,1,0),0),((1,0,1),0),((1,0,0),1),\\ ((0,1,1),1),((0,1,0),1),((0,0,1),1),((0,0,0),0))\end{aligned} \tag{1}$$

Uma outra forma de representar uma tabela de transições é a forma icônica. Na forma icônica o bit 1 é representado por um ícone na cor preta, e o bit 0 por um ícone na cor branca. Cada uma das transições de estados é representada por um conjunto de ícones que simbolizam a vizinhança na parte superior, e o estado resultante após a transição, na parte inferior. A Figura 5 ilustra a representação icônica da regra 30.

Além dessas formas de representação, ainda existe a forma k-ária em que, sabendo-se

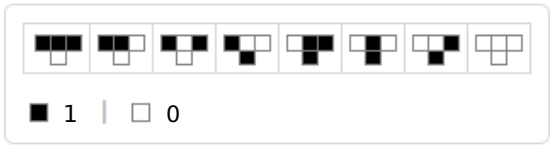


Figura 5: Representação icônica da regra 30.

o valor atribuído para k e r, elimina-se a representação das vizinhanças e deixa-se apenas as representações resultantes. Na Eq. (2) é possível ver a representação k-ária da regra 30.

$$(0, 0, 0, 1, 1, 1, 1, 0) \tag{2}$$

O número de regras de um espaço é dado pela Eq. (3):

$$k^{k^{2r+1}} \tag{3}$$

No espaço elementar há $2^{2^3} = 256$ regras. Aumentando o raio $r = 2$, obtemos uma família de $2^{2^5} = 4.294.967.296$ regras. Para $k = 3$ e $r = 1$, obtém-se um espaço de ACs com $3^{3^3} = 7.625.597.484.987$ regras. Logo, é fácil perceber que qualquer modificação nas variáveis k e r geram famílias com número de regras muito grande. Famílias grandes de ACs representam um desafio na hora de encontrar ACs com propriedades específicas, já que procurar regras através de força bruta em um espaço muito grande se torna uma tarefa extremamente improdutiva.

Para contornar esse problema, é comum utilizar algumas propriedades estáticas para restringir as regras do espaço no qual serão feitas as buscas, no entanto, faz-se necessário uma forma de representar essas propriedades e até mesmo aplicar operações como intersecção e diferença entre elas. Nesse ponto que os *templates* apresentam-se de uma forma interessante e útil para representar conjuntos de regras com determinada propriedade. Templates de ACs são uma generalização das tabelas de transições que permitem representar espaços inteiros de ACs (VERARDO, 2014). No Capítulo 4 os templates serão explicados em mais detalhes e a no Capítulo 3 são explicados algumas propriedades estáticas que podem ser representadas por templates.

3 PROPRIEDADES ESTÁTICAS

Em ACs, propriedades estáticas são propriedades computadas com base nas tabelas de transições. Essas propriedades permitem prever determinados comportamentos de ACs sem consultar sua evolução espaço-temporal.

Esta seção descreve algumas propriedades estáticas que já têm seus algoritmos geradores de templates implementados na biblioteca *CATemplates* (VERARDO; DE OLIVEIRA, 2015).

3.1 Conservabilidade de Estados e Conservabilidade de Paridade

Conservabilidade de estados é uma propriedade estática que determina que a soma dos estados de um determinado autômato celular não deve se alterar durante a evolução espaço-temporal, independente da configuração inicial.

De acordo com Boccara e Fukś (2002), um AC é conservativo quando cada uma de suas regras locais f de vizinhança $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ respeita as condições descritas na Eq. (4).

$$\begin{aligned} f(\alpha_0, \alpha_1, \ldots, \alpha_{n-1}) = \alpha_0 + \sum_{i=0}^{n-2} (& f(0_0, 0_1, \ldots, 0_i, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}) \\ & - f(0_0, 0_1, \ldots, 0_i, \alpha_0, \alpha_1, \ldots, \alpha_{n-i-1})) \end{aligned} \tag{4}$$

Para exemplificar, considere-se a regra 204 do espaço elementar. Por meio da condição mostrada na Eq. (4), será provado que essa regra é conservativa, já que satisfaz a condição. A Eq. (5) representa a tabela de transições da regra 204.

$$\begin{aligned} &(((1,1,1),1),((1,1,0),1),\\ &((1,0,1),0),((1,0,0),0),\\ &((0,1,1),1),((0,1,0),1),\\ &((0,0,1),0),((0,0,0),0)) \end{aligned} \tag{5}$$

Como demonstra Verardo (2014), a aplicação das condições da Eq. (4) nas tabelas de

transições de ACs do espaço elementar, resulta no sistema descrito pela Eq. (6).

$$\begin{cases} f(0,0,0) = 0 + (f(0,0,0) - f(0,0,0)) + (f(0,0,0) - f(0,0,0)) \\ f(0,0,1) = 0 + (f(0,0,1) - f(0,0,0)) + (f(0,0,0) - f(0,0,0)) \\ f(0,1,0) = 0 + (f(0,1,0) - f(0,0,1)) + (f(0,0,1) - f(0,0,0)) \\ f(0,1,1) = 0 + (f(0,1,1) - f(0,0,1)) + (f(0,0,1) - f(0,0,0)) \\ f(1,0,0) = 1 + (f(0,0,0) - f(0,1,0)) + (f(0,0,0) - f(0,0,1)) \\ f(1,0,1) = 1 + (f(0,0,1) - f(0,1,0)) + (f(0,0,0) - f(0,0,1)) \\ f(1,1,0) = 1 + (f(0,1,0) - f(0,1,1)) + (f(0,0,1) - f(0,0,1)) \\ f(1,1,1) = 1 + (f(0,1,1) - f(0,1,1)) + (f(0,0,1) - f(0,0,1)) \end{cases} \tag{6}$$

A Eq. (6) simplificada é representada pela Eq. (7).

$$\begin{cases} f(0,0,0) &= 0 \\ f(0,0,1) &= f(0,0,1) \\ f(0,1,0) &= f(0,1,0) \\ f(0,1,1) &= f(0,1,1) \\ f(1,0,0) &= 1 - f(0,0,1) - f(0,1,0) \\ f(1,0,1) &= 1 - f(0,1,0) \\ f(1,1,0) &= 1 + (f(0,1,0) - f(0,1,1)) \\ f(1,1,1) &= 1 \end{cases} \tag{7}$$

Excluindo-se as condições tautológicas do sistema e atribuindo os valores das funções locais f conforme a tabela de transições da regra 204, é obtido o sistema descrito pela Eq. (8). Esse sistema, ao não apresentar condições contraditórias ou falsas, prova que a regra 204 é conservativa.

$$\begin{cases} 0 &= 0 \\ 0 &= 1 - 0 - 1 \\ 0 &= 1 - 1 \\ 1 &= 1 + (1 - 1) \\ 1 &= 1 \end{cases} \tag{8}$$

O processo de gerar regras conservativas de paridade é bem parecido. Porém, as condições estabelecidas por Boccara e Fukś (2002) são ligeiramente modificadas em relação à Eq. (4), de forma que cada uma das funções locais deve respeitar agora as condições da

Eq. (9).

$$f(\alpha_0, \alpha_1, \ldots, \alpha_{n-1}) \equiv \alpha_0 + \sum_{i=0}^{n-2} (f(0_0, 0_1, \ldots, 0_i, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}) \\ -f(0_0, 0_1, \ldots, 0_i, \alpha_0, \alpha_1, \ldots, \alpha_{n-i-1})) \mod 2 \tag{9}$$

3.2 Simetria Interna

Para entender a propriedade de simetria interna faz-se necessário compreender o funcionamento das transformações de regras e classes de equivalência dinâmica. As explicações a seguir são válidas para regras binárias pois o algoritmo implementado no *CATemplates* é para regras binárias, apesar de ser possível sua generalização para k estados.

Dada uma tabela de transições de um AC, existem três transformações que podem ser empregadas e que resultam em ACs com comportamentos dinâmicos equivalentes: *reflexão*, *conjugação* e *composição reflexão-conjugação*. A reflexão é a transformação obtida ao refletir os bits das vizinhanças da tabela. A conjugação é obtida ao inverter todos os estados das células da tabela de transições. Já a composição é a transformação obtida ao se efetuar a reflexão e a conjugação, independente da ordem.

Essas transformações podem ser aplicadas tanto à vizinhança, como também a toda a tabela de transições. Para aplicar uma transformação a toda uma tabela, basta aplicá-la a cada uma das vizinhanças da tabela. Uma tabela, transição ou vizinhança é chamada de invariante a uma transformação caso essa transformação, quando aplicada a ela, não promova nenhum efeito. Um exemplo de vizinhança invariante por reflexão é a vizinhança $(1, 0, 1)$.

Para exemplificar essas transformações e as equivalências dinâmicas considere-se a tabela de transição da regra 60, ilustrada pela Figura 6. Ao aplicar a transformação por reflexão na regra 60 obtém-se a regra 102 do espaço elementar, ilustrada pela Figura 7.

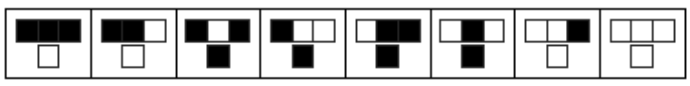


Figura 6: Tabela de transições da regra 60 do espaço elementar.

Ao aplicar a transformação por conjugação na regra 60 obtém-se a regra 195 do espaço elementar, conforme ilustrado na Figura 8. Por fim, ao aplicar a transformação por composição de reflexão e conjugação na regra 60 obtém-se a regra 153 do espaço elementar, conforme ilustrado na Figura 9.

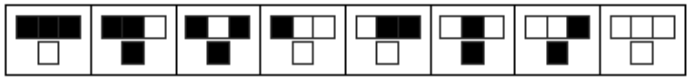


Figura 7: Tabela de transições da regra 102 do espaço elementar, obtida através da transformação de reflexão aplicada na tabela de transições da regra 60.

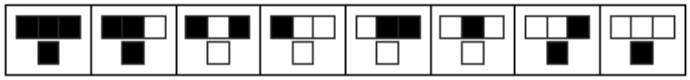


Figura 8: Tabela de transições da regra 195 do espaço elementar, obtida através da transformação de conjugação aplicada na tabela de transições da regra 60.

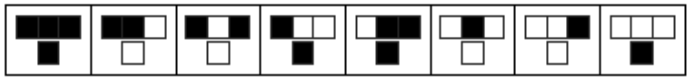


Figura 9: Tabela de transições da regra 153 do espaço elementar, obtida através da transformação de composição aplicada na tabela de transições da regra 60.

Tanto a regra 60, como as regras 102, 153 e 195 pertencem à mesma classe de equivalência dinâmica. Uma forma interessante de entender o porquê é analisando a evolução espaço-temporal dessas regras, ilustrada pela Figura 10.

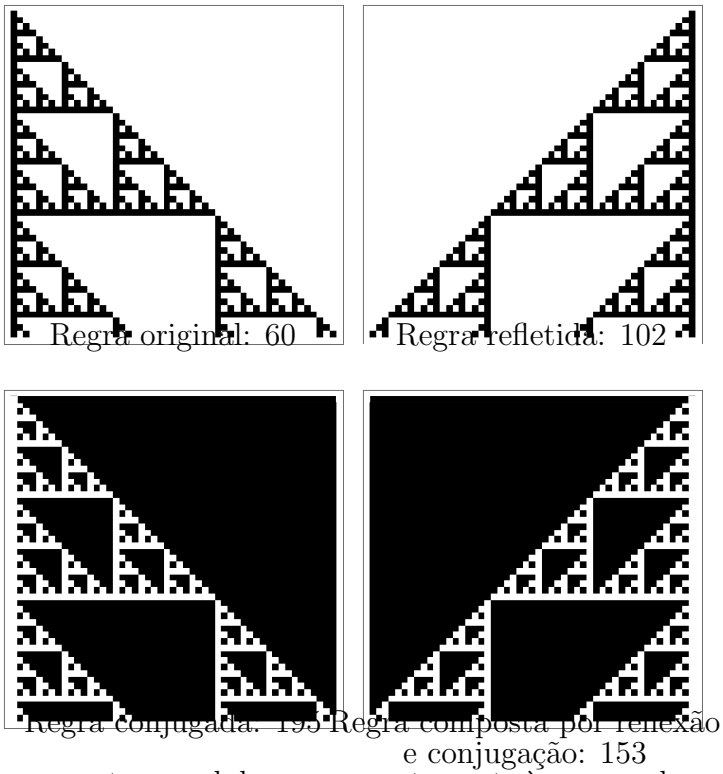


Figura 10: Evolução espaço-temporal das regras pertencente à mesma classe dinâmica da regra 60.

Tendo em vista as tabelas de transições obtidas por meio das transformações, é possível saber quão simétrica é uma regra em relação a uma determinada transformação ou, em outras palavras, qual o grau de simetria interna de uma regra em relação à transformação em questão. A simetria interna pode ser representada pelo número de vizinhanças que permanecem iguais após aplicada uma determinada transformação. Exemplificando, a regra 60 dos ACs elementares tem valor de simetria interna por reflexão igual a 4, pois compartilha as transições de estado $((1,1,1),0)$, $((1,0,1),1)$, $((0,1,0),1)$ e $((0,0,0),0)$ com a regra resultante de sua transformação por reflexão, a regra 102. Vale notar que, como as vizinhanças compartilhadas são todas e apenas as vizinhanças invariantes do espaço elementar, logo pode-se dizer que a regra 60 tem a menor simetria interna possível para o espaço elementar.

Um resultado bem diferente pode ser visto ao se repetir esse mesmo processo para a regra 204, que tem valor 8 de simetria interna para transformação por reflexão. Esse valor é máximo possível para o espaço elementar e evidencia que, ao se aplicar a transformação por reflexão na regra 204, será obtida a mesma regra como resultado. Uma regra que possua máxima simetria para uma transformação é uma regra invariante.

Vale frisar que apenas transformações por reflexão apresentam vizinhanças invariantes. No caso das reflexões por conjugação e por composição isso não ocorre.

3.3 Totalidade e Semi-totalidade

Os ACs totalísticos são autômatos celulares cujo valor de uma célula depende apenas da soma dos valores dos seus vizinhos no passo de tempo anterior (WOLFRAM, 1983).

De forma análoga, pode-se considerar que as transições dependem da média dos valores das células que compõem a vizinhança. A Figura 11 ilustra a tabela de transição de um AC totalístico com $k = 3$ e $r = 1$ (especificamente, o de número 777, entre os totalísticos do espaço em questão) (WOLFRAM, 2002).

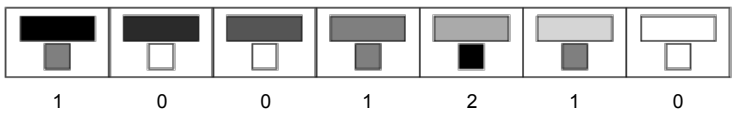


Figura 11: Tabela de transição do AC totalístico 777. As transições dependem da média dos estados das células da vizinhança. Cada média possível é representada por tons de cinza entre o branco (estado 0) e o preto (estado 2).

Já os ACs semi-totalísticos (*outer-totalistic CA*) são generalizações de ACs totalísticos (PHILLIPS; WEISSTEIN, 2015). Um AC é considerado semi-totalístico caso suas transições sejam definidas pela soma (ou média) das células da vizinhança sem levar em consideração a célula central.

3.4 Confinamento

Os autômatos celulares confinados (do inglês, *captive*), são uma classe de ACs que se baseiam em uma caracterização de suas funções locais que não adotem estados definidos em qualquer estrutura externa à vizinhança (THEYSSIER, 2004).

Theyssier (2004) formalmente define que, dada a função local f de um AC para a vizinhança $(\alpha_0, \ldots, \alpha_{2r})$, sendo o r o raio, um AC é considerado confinado se respeitar a condição descrita na Eq. (10).

$$f((\alpha_0, \ldots, \alpha_{2r})) = \beta, \beta \in \{\alpha_0, \ldots, \alpha_{2r}\} \tag{10}$$

Naturalmente, qualquer AC binário que tenha as funções locais $f((0_0, 0_1, \ldots, 0_{2r})) = 0$ e $f((1_0, 1_1 1 \ldots, 1_{2r})) = 1$ é um AC confinado.

4 TEMPLATES

Templates de autômatos celulares são uma generalização de suas tabelas de transição de estado, capazes de representar famílias de regras. Os templates foram criados por de Oliveira e Verardo (2014a) e implementados como um algoritmo na linguagem do software *Wolfram Mathematica* (WOLFRAM RESEARCH, 2015), atualmente disponíveis na biblioteca *open source CATemplates* (VERARDO; DE OLIVEIRA, 2015) no GitHub.

Formalmente, um *template* é uma n-tupla formada por k^{2r+1} itens, e cada item i representa uma função $g_i(x_{k^{2r+1}-1}, \ldots, x_0)$. As variáveis x_i podem assumir qualquer estado entre 0 e $k-1$, o que implica que, no caso binário, x_i pode assumir os valores 0 e 1. Pode-se limitar os valores possíveis de x_i através da notação $x_i \in C$, onde C é um conjunto representando os possíveis valores de x_i; note-se, entretanto, que essa notação só tem sentido prático para ACs com quantidade de estados $k > 2$.

Exemplificando, dado um template $T_1 = (1, 1, 1, 1, 1 - x_1, x_2, x_1, 0)$, ele representará todas as regras que tenham na posição 0 (sempre da direita para a esquerda) o estado 0, nas posições 4, 5, 6 e 7 o estado 1, nas posições 1 e 2 qualquer estado no intervalo $[0, k-1]$, e na posição 3 o estado complementar ao valor da posição 1. Perceba-se que o tamanho da n-tupla é k^{2r+1}, acarretando que no template T_1 os únicos valores inteiros possíveis para k e r são 2 e 1 respectivamente. Portanto T_1 representará um subespaço dos ACs elementares.

Deste modo o template T_1 representa o conjunto de autômatos celulares elementares $\{(1,1,1,1,1,0,0,0), (1,1,1,1,0,0,1,0), (1,1,1,1,1,1,0,0), (1,1,1,1,0,1,1,0)\}$, ou em suas formas decimais, o conjunto de regras $\{248, 242, 252, 246\}$.

Cada template tem um número de substituições máximo igual a k^m, sendo m o número de variáveis livres. O maior template possível de uma família de ACs é o *template base*, em que todas as variáveis são livres, e que representa todas as regras do espaço em questão. O menor é o *template constante*, em que não há variáveis livres, e representa, portanto, apenas uma regra. A 8-tupla mostrada na Eq. (11) representa um template constante que corresponde à regra elementar 30.

$$(0, 0, 0, 1, 1, 1, 1, 0) \tag{11}$$

Já a 8-tupla, apresentada na Eq. (12), representa um template base que está associado

a todas as 256 regras do espaço elementar já que, para $m = 8$, temos $2^m = 256$.

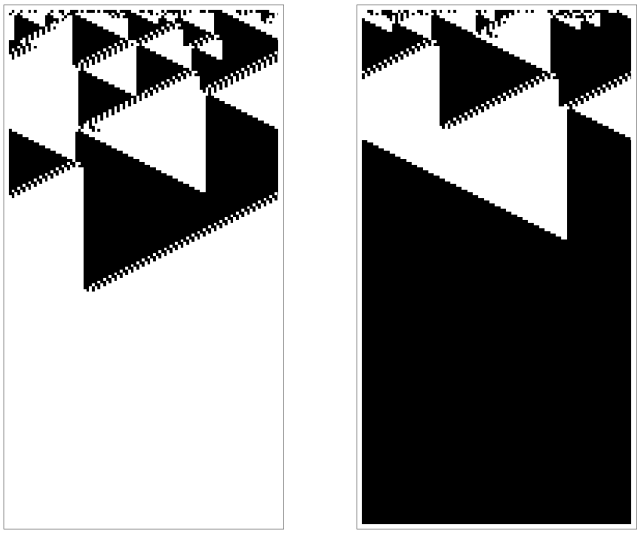
$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \tag{12}$$

É importante enfatizar que nem sempre o número de substituições é igual a k^m. Isto ocorre pois algumas substituições podem originar tabelas de transições inválidas. O template binário $(1, 1, 1, 1, 1, x_0+x_1, x_1, x_0)$, por exemplo, não pode apresentar as substituições $x_0 = 1$ e $x_1 = 1$ ao mesmo tempo, pois isso faz com que $x_0 + x_1 \notin [0, k-1]$, invalidando assim essa substituição para $k = 2$.

A representação de famílias de autômatos celulares através de templates possibilita a utilização de templates para problemas já bem conhecidos da área de ACs. Um exemplo de problema que pode se beneficiar dos templates é o problema da paridade.

Para o problema de paridade, nesse projeto, considera-se um AC binário, unidimensional, com reticulado de tamanho ímpar, em condição de contorno periódica. Se uma configuração inicial contiver um número ímpar de estados com valor 1, o AC deve convergir para uma configuração global onde toda as células estejam preenchidas com 1 (i.e., com o ponto fixo 1^+), caso contrário, ele deve convergir para todos os estados com o valor 0. Há um problema nessa definição para reticulado de tamanho par, pois uma configuração inicial com todas as células apresentando o estado 1 teriam que convergir para uma configuração com todos os estados apresentando o valor 0. Devido a isso, pode-se dizer que as regras que solucionam o problema de paridade em ACs são *perfeitas* se eles resolverem o problema de paridade em qualquer configuração inicial arbitrária para ACs de tamanho ímpar (BETEL; DE OLIVEIRA; FLOCCHINI, 2013). Conforme Betel, de Oliveira e Flocchini (2013) nos lembram, duas condições devem ser satisfeitas em relação ao problema de paridade. A primeira é que, se f é a regra local que resolve o problema de paridade, então $f(0, \ldots, 0) = 0$ e $f(1, \ldots, 1) = 1$. A segunda é que, para uma regra preservar a configuração de paridade, ela deve apresentar número par de transições ativas (quais, sejam, as que imponham uma troca do bit de saída); em outras palavras, toda aplicação da regra deve levar a uma nova configuração com a mesma paridade.

A Figura 12 ilustra o desenvolvimento espaço-temporal da regra BFO (BETEL; DE OLIVEIRA; FLOCCHINI, 2013) que resolve o problema de paridade para o raio 4. Nessa imagem o desenvolvimento temporal à esquerda contém, em sua configuração inicial, um número par de estados igual a 1 e, na evolução temporal ilustrada à direita, um número ímpar de estados igual a 1.



(a) Configuração inicial com número ímpar de 1s

(b) Configuração inicial com número par de 1s

Figura 12: Evolução espaço-temporal da regra BFO, que resolve o problema de paridade para raio 4 (BETEL; DE OLIVEIRA; FLOCCHINI, 2013).

O estudo do problema da paridade é interessante pois ajuda a compreender o impacto das interações locais nas soluções globais. Especificamente, entender a influência que o tamanho da vizinhança em autômatos celulares apresenta na computabilidade pode apresentar consequências úteis tanto para ACs, como para a compreensão de sistemas emergentes complexos em geral.

Estudos feitos sobre o problema de paridade já levaram ao conhecimento de que o problema não tem solução perfeita para ACs elementares e de raio 2. Todavia foi construída uma regra perfeita que soluciona o problema de paridade para raio 4. Em relação aos ACs de raio 3, ainda não foi encontrada solução perfeita e há evidências empíricas desfavoráveis a uma solução para esse raio (BETEL; DE OLIVEIRA; FLOCCHINI, 2013).

Betel, de Oliveira e Flocchini (2013), buscando verificar se o problema da paridade em ACs de raio 2 apresentam alguma regra que solucione o problema para qualquer configuração inicial, encontraram de forma analítica como as transições de estado de supostas regras que resolvessem o problema de paridade deveriam ser.

Para representar como as transições de estado deveriam ser, foram utilizados grafos de De Bruijn. Grafos de De Bruijn são grafos, propostos de forma independentes por De Bruijn (1946) e Good (1946), cujos nós são sequências de símbolos de algum alfabeto e

cujas arestas indicam as sequências em que se cruzarão (WEISSTEIN, 2015a). Em AC, esse tipo de grafo direcionado representa em cada uma de suas arestas uma vizinhança possível, e o valor associado às arestas será o resultado da transição, dada à vizinhança representada pela aresta. A Figura 13 ilustra dois grafos de De Bruijn, sendo que o da esquerda pode ser utilizado para ilustrar todas as vizinhanças possíveis dos autômatos celulares elementares, e o da direita pode ser usado para ilustrar todas as vizinhanças de autômatos celulares com raio $1,5$ e 2 estados.

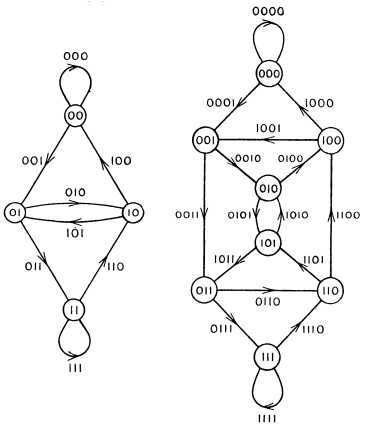


Figura 13: Exemplo de grafo de De Bruijn (GOOD, 1946)

Com isso, utilizando grafos de De Bruijn, foram definidas quais variáveis deveriam ser valores constantes, quais deveriam ser livres, e quais deveriam apresentar interdependência. Com essas definições foram encontrados dois conjuntos de ACs. Os grafos ilustrados na Figura 14 e na Figura 15 são os grafos desenvolvidos por Betel, de Oliveira e Flocchini (2013).

Ambos os grafos apresentam arestas contendo as variáveis livres a, b, c, d e x, e uma interdependência em que uma transição de estado deve ter o valor oposto à variável x. A única diferença entre os dois grafos está nas arestas contendo variáveis estáticas.

Ao utilizar os grafos de De Bruijn, fixando algumas transições de estado, a família de ACs em que se procurava as regras que solucionariam o problema de paridade foi restringida para apenas 64 regras. Ao se restringir o espaço de busca, antes composto por 2^{32} regras, as regras puderam ser estudadas em mais detalhes até que falhassem. Entretanto, conforme mostrado em (VERARDO, 2014), a representação desse espaço de 64 regras pode ser equivalentemente representado por meio de *templates*. O template da Eq. (13) representa o mesmo espaço que a Figura 14, e o template da Eq. (14) é

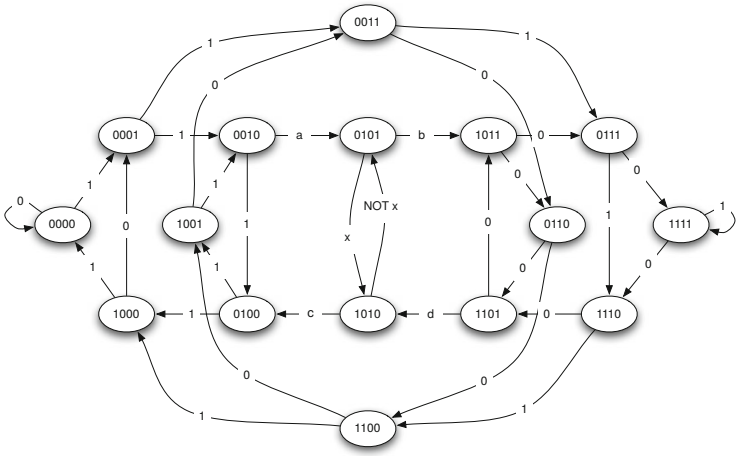


Figura 14: Grafo de De Bruijn representando regras que possivelmente solucionassem o problema de paridade (BETEL; DE OLIVEIRA; FLOCCHINI, 2013).

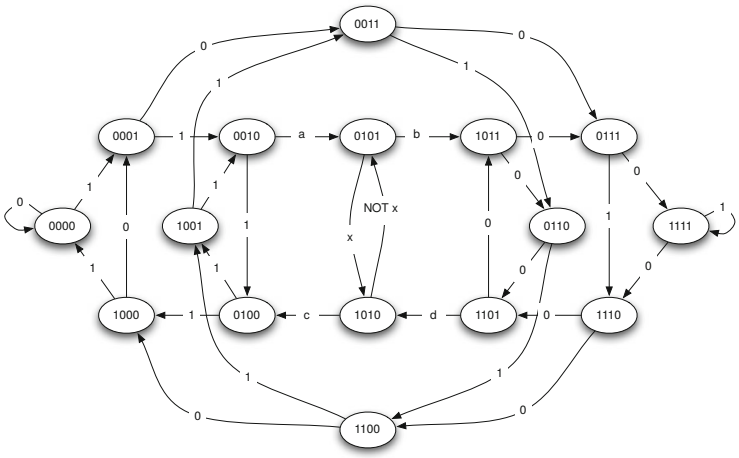


Figura 15: Outro grafo de De Bruijn representando regras adicionais que possivelmente solucionassem o problema de paridade (BETEL; DE OLIVEIRA; FLOCCHINI, 2013).

equivalente à Figura 15. Ambos os templates referem-se a raio $r = 2$ e 2 estados.

$$(0, 1, 1, 1, 1, x_{26}, 0, 1, 1, 1, 1 - x_{10}, x_{20}, 0, 0, 1, 0, 1, 0, 1, 0, x_{11}, x_{10}, 0, 0, 1, 0, x_5, 0, 1, 0, 0, 1) \tag{13}$$

$$(0, 1, 1, 0, 1, x_{26}, 1, 0, 1, 1, 1 - x_{10}, x_{20}, 0, 1, 1, 0, 1, 0, 1, 1, x_{11}, x_{10}, 0, 0, 1, 0, x_5, 0, 0, 0, 0, 1) \tag{14}$$

4.1 Expansão de Templates

Expansão é o processo pelo qual se obtêm todas as tabelas de transição R_k associadas a um template T. A operação de expansão foi apresentada por Verardo (2014) e foi descrita em mais detalhes da seguinte maneira:

$$E(T) = R_k \tag{15}$$

A operação de expansão pode ser dividida em dois passos: processamento e pós-processamento. O primeiro consiste em efetuar todas as k^m substituições de variáveis possíveis. Considere como exemplo o template $T_1 = (1, 1, 1, 1, 1 - x_1, x_2, x_1, 0)$, o primeiro passo do processo de expansão consiste em encontrar as tabelas k-árias resultantes das combinações possíveis das substituições de x_1 e x_2, conforme pode ser melhor visualizado na Tabela 1.

Tabela 1: Processo de expansão.

i	x_2	x_1	Tabela k-ária resultante
0	0	0	(1,1,1,1,1,0,0,0)
1	0	1	(1,1,1,1,1,1,0,0)
2	1	0	(1,1,1,1,0,0,1,0)
3	1	1	(1,1,1,1,0,1,1,0)

Na maioria dos casos o pós-processamento não é necessário, ou ele consiste apenas em eliminar as tabelas k-árias inválidas. No caso do template T_1 acima todas as tabelas resultantes são válidas e nenhum pós-processamento se faz necessário; mas nem sempre isso ocorre. No template $T_2 = (1, 1, 1, 1, 1, x_0 + x_1, x_1, x_0)$, por exemplo, a substituição $x_0 = 1$ e $x_1 = 1$ resulta numa tabela k-ária inválida, pois a posição 2 resultaria em um estado com valor fora do intervalo $[0, k - 1]$, para $k = 2$. A Tabela 2 evidencia melhor essa substituição inválida.

Conforme mostrado na Tabela 2, para o template T_2 listar apenas regras válidas é necessário aplicar um pós-processamento que filtre as tabelas k-árias inválidas, no caso, a gerada pela expansão de $i = 3$.

Um outro exemplo de template que resulta em substituições inválidas são os templates que utilizam a notação de restrição por conjuntos. Considere-se o template $T_3 =$

Tabela 2: Processo de expansão

i	x_1	x_0	tabela k-ária resultante
0	0	0	(1,1,1,1,1,0,0,0)
1	0	1	(1,1,1,1,1,1,0,1)
2	1	0	(1,1,1,1,1,1,1,0)
3	1	1	(1,1,1,1,1,2,1,1)

$(2,2,2,2,2,2,2,2,x_0 \in \{0,1\})$ da família de $k = 3$ e $r = 0{,}5$, sendo que raio 0,5 define uma vizinhança com apenas dois campos. A substituição obtida para $i = 2$ (conforme a Tabela 3) seria inválida pois nesse caso $x_0 = 2 \notin \{0,1\}$, sendo assim necessário que o pós-processamento desconsidere essa expansão gerada.

Tabela 3: Processo de expansão

i	x_0	tabela k-ária resultante
0	0	(2,2,2,2,2,2,2,2,0)
1	1	(2,2,2,2,2,2,2,2,1)
2	2	(2,2,2,2,2,2,2,2,2)

O pós-processamento da operação de expansão pode variar de acordo com o template. No template que representa as regras conservativas de paridade, por exemplo, o pós-processamento consiste em aplicar a operação *mod* 2 em cada um dos campos das tabelas k-ária. Em templates com restrições de variáveis, é no pós-processamento que se eliminam as tabelas k-árias incompatíveis com as restrições.

A existência de regras inválidas possibilita templates que representem um conjunto de regras diferentes de k^m. Essa possibilidade é bastante útil para os templates de regras conservativas, regras conservativas de paridade e regras confinadas, que representam um número de regras diferente de k^m. Deste modo, toda vez que se gera um template, também é necessário informar que tipo de pós-processamento será necessário utilizar na hora da expansão. A Tabela 4 mostra os pós-processamentos necessários para as principais funções geradoras de templates do *CATemplates*.

Note-se que a i-ésima substituição, $i \in [0, k^m - 1]$, sempre representa apenas uma substituição possível para as variáveis livres de um template. Isso ocorre pois i é a repre-

Tabela 4: Pós-processamento necessário por templates

Template	Pós-processamento
Template de regras confinadas	Filtrar regras com restrições inválidas
Template de conservabilidade de estados	Filtrar regras inválidas
Template de conservabilidade de paridade	Aplicar *mod* 2 e filtrar regras inválidas
Template de totalidade e semi-totalidade	Nenhum pós-processamento necessário
Templates de simetria	Nenhum pós-processamento necessário

sentação decimal da conversão k-ária das concatenações dos valores das variáveis livres em ordem decrescente. Exemplificando, considere no templates $T_4 = (2, 2, 2, 2, 2, 2, 2, x_1 \in \{0, 1\}, x_0)$ para $k = 3$, o valor de $i = 5$ será convertido pelo processo de expansão obtendo-se assim o seu equivalente na base ternária $(1, 2)$ e então cada um dos dígitos é atribuído a uma variável, resultando assim no conjunto de substituições $x_1 = 1, x_0 = 2$.

A forma com que o valor de i representa apenas uma expansão possibilita se obter a i-ésima expansão de um template. Essa propriedade é relevante devido ao fato de a expansão ser uma operação potencialmente custosa, e a possibilidade de ser realizar a i-ésima expansão de um template facilita e permite o paralelismo.

4.2 Intersecção de Templates

Intersecção é o processo pelo qual, de dois templates T_1 e T_2, obtém-se o template T_3 que representa o conjunto R_k. O conjunto R_k representa todas as regras pertencentes aos dois templates recebidos como parâmetro. É necessário que os dois templates recebidos como parâmetro pertençam ao mesmo espaço. A operação de intersecção foi descrita por Verardo (2014) e mostrada em mais detalhes da seguinte maneira:

$$I(T_1, T_2) = T_3 \Leftrightarrow E(T_3) = E(T_1) \cap E(T_2) \tag{16}$$

A operação de intersecção, assim como a de expansão, também é efetuada em duas etapas. Na primeira etapa igualam-se os dois templates e assim se obtém um sistema de equações. Esse sistema de equações é então passado como argumento para a função *Solve*, nativa da *Wolfram Language* (WOLFRAM RESEARCH, 2015), para ser resolvido. A função *Solve* retorna então os relacionamentos entre as variáveis, que ao serem aplicados aos templates recebidos, retorna dois templates equivalentes, bastando escolher um que será o template de intersecção. No caso dos templates não apresentarem intersecção, a função *Solve* nada retornará.

Para melhor compreensão, considerem-se os templates $T_1 = (x_7, x_3, 1-x_4, x_4, x_3, x_2, 2, x_0)$ e $T_2 = (x_7, 1, x_5, 0, x_3, x_2, 2, 2)$, ambos com $r = 0{,}5$ e $k = 3$. Esse templates serão transformados em um sistema de equações como demonstrado na Eq. (17).

$$\begin{cases} x_7 &= x_7 \\ x_3 &= 1 \\ 1 - x_4 &= x_5 \\ x_4 &= 0 \\ x_3 &= x_3 \\ x_2 &= x_2 \\ 2 &= 2 \\ x_0 &= 2 \end{cases} \tag{17}$$

Esse sistema de equações é passado então como argumento para a função *Solve* que, por sua vez, retorna um conjunto solução S, neste exemplo, $S = \{x_0 = 2, x_3 = 1, x_4 = 0, x_5 = 1 - x_4, x_6 = 0\}$. O conjunto S é aplicado como um conjunto de substituições sobre os dois templates recebidos como parâmetro que, em caso de templates sem restrição de

variáveis, sempre retorna o mesmo template. No exemplo, após aplicadas as substituições do conjunto de soluções S, obtém-se como resultado o template $T_3 = (x_7, 1, 1, 0, 1, x_2, 2, 2)$.

A segunda etapa do algoritmo é aplicada apenas para templates com alguma restrição de variável. Essa etapa consiste em extrair as expressões que estabelecem as restrições, e através delas obter um segundo sistema de equações. A solução desse sistema pode ser vazia, expressando assim que os templates não tem intersecção, ou pode indicar os valores que as variáveis com restrição podem assumir.

Para exemplificar essa segunda etapa, considerem-se os templates $T_{r1} = (x_7 \in \{0, 1, 2\}, x_3, 1 - x_4, x_4, x_3, x_2 \in \{1, 2\}, 2, x_0)$ e $T_{r2} = (x_7 \in \{0, 1\}, 1, x_5, 0, x_3, x_2 \in \{1\}, 2, 2)$, derivados dos templates T_1 e T_2 do exemplo anterior, aos quais foram acrescidas as restrições de variáveis. A primeira etapa ocorre normalmente; entretanto, quando as substituições do conjunto S forem aplicadas nos templates recebidos, não serão mais obtidos templates iguais, quais sejam, $\{(x_7 \in \{0, 1, 2\}, 1, 1, 0, 1, x_2 \in \{1, 2\}, 2, 2), (x_7 \in \{0, 1\}, 1, 1, 0, 1, x_2 \in \{1\}, 2, 2)\}$. Na sequência o algoritmo faz a extração das expressões de restrição de variáveis e obtêm o conjunto $\{x_7 \in \{0, 1\}, x_2 \in \{1, 2\}, x_2 \in \{1\}\}$. Esse conjunto é então convertido para o sistema de equações representadas pela Eq. (18).

$$\begin{cases} x_7 = 0 \quad \vee \quad x_7 = 1 \quad \vee \quad x_7 = 2 \\ x_7 = 0 \quad \vee \quad x_7 = 1 \\ x_2 = 1 \quad \vee \quad x_2 = 2 \\ x_2 = 1 \end{cases} \tag{18}$$

Esse sistema de equações é então passado como argumento para a função *Solve*, que retorna seu conjunto solução. Por fim o algoritmo usa o conjunto solução obtido para remover as restrições da variável x_2, transformando-a no valor 1, e restringir a variável x_7 apenas ao conjunto $\{0, 1\}$. O template de intersecção gerado por todo esse processo é representado pela Eq. (19).

$$T_{r3} = (x_7 \in \{0, 1\}, x_3, 1 - x_4, x_4, x_3, 1, 2, x_0) \tag{19}$$

Observe-se no exemplo que o template T_{r2} também poderia ser representado substituindo a variável x_2 e seu conjunto de restrição $\{1\}$ apenas pelo valor constante 1, como mostrado a seguir: $T_{r2} = (x_7 \in \{0, 1\}, 1, x_5, 0, x_3, 1, 2, 2)$. Esse tipo de mudança é recomendado pois variáveis a mais acarretam em mais processamento.

Note-se ainda que, no caso binário ($k = 2$), a notação de restrição nunca é necessária, visto que ela terá apenas um valor factível, tornando preferível que essa variável e sua restrição sejam substituídas por um valor constante, ou a variável poderá assumir qualquer estado, sendo assim, por definição, uma variável livre.

5 REPRESENTAÇÃO DE PROPRIEDADES ESTÁTICAS POR MEIO DE TEMPLATES

Uma propriedade obtida por meio de restrições aplicadas à tabela de transições de um AC é denominada como uma propriedade estática. Propriedades estáticas ajudam prever a evolução de um AC e, por isso, elas podem ser usadas para restringir as regras do espaço no qual serão feitas as buscas para solucionar determinado problema de ACs.

Nesse Capítulo são mostrados templates capazes de representar propriedades estáticas e seus algoritmos geradores.

5.1 Templates Para Regras Com Conservabilidade de Estados e Com Conservabilidade de Paridade

Conservabilidade de estados é uma propriedade estática que determina que a soma dos estados de um determinado autômato celular não deve se alterar durante a evolução espaço-temporal, independente da configuração inicial.

O algoritmo que gera templates que representam regras conservativas, criado por de Oliveira e Verardo (2014a), primeiramente recebe as variáveis k e r definindo assim a família das regras que serão geradas. Em seguida, cria todas as vizinhanças do espaço, com exceção das vizinhanças que geram tautologias, e após esses passos, aplica as condições de Boccara e Fukś (2002). Para se excluir as vizinhanças que geram tautologias, basta excluir as vizinhanças que começam com 0 mas não sejam compostas apenas por 0 (SCHRANKO; DE OLIVEIRA, 2010).

Para exemplificar o funcionamento do algoritmo, considere-se o espaço com $k = 2$ e $r = 1$. Primeiramente, o algoritmo obterá o conjunto das vizinhanças que não geram regras tautológicas, qual seja, o conjunto $\{(1,1,1),(1,1,0),(1,0,1),(1,0,0),(0,0,0)\}$. Então é

criado o sistema de equações (20), baseado nas condições de Boccara e Fukś (2002).

$$\begin{cases} x_0 &= 0 \\ x_4 &= 1 + 2x_0 - x_1 - x_2 \\ x_5 &= 1 + x_0 - x_2 \\ x_6 &= 1 + x_2 - x_3 \\ x_7 &= 1 \end{cases} \tag{20}$$

Por fim, o algoritmo utiliza a função *Solve* do *Wolfram Mathematica* para simplificar o sistema, e utiliza o conjunto solução retornado pela função *Solve* como regras de substituições. Essas regras de substituições são aplicadas no template base, gerando assim o template das regras conservativas. O template gerado está mostrado na Eq. (21), e sua expansão gera as cinco regras conservativas do espaço elementar, após eliminadas as regras inválidas.

$$(1, x_2 - x_3 + 1, 1 - x_2, -x_1 - x_2 + 1, x_3, x_2, x_1, 0) \tag{21}$$

O processo de gerar regras conservativas de paridade é bem parecido. Respeitando-se as condições da Eq. (9), mostrada na Subseção 3.1, chega-se no mesmo template que representa as regras conservativas, representado pela Eq. (21). Todavia, antes de filtrar as regras inválidas no pós-processamento da expansão desse template, é aplicado *mod* 2 a todas as tabelas k-árias encontradas.

A biblioteca *CATemplates* já tem implementado o algoritmo gerador de templates de regras conservativas de paridade e, conforme será mostrado posteriormente, esse algoritmo pode apresentar utilidade na busca de uma solução para o problema de paridade.

5.2 Templates Para Regras Com Valores Arbitrários de Simetria Interna

Já há implementado no *CATemplates* um algoritmo gerador de templates que representam regras do espaço binário com um determinado valor de simetria para uma transformação.

O algoritmo recebe como parâmetro um raio r, para definir a família de ACs que será representada, o valor de simetria interna desejado, e uma das transformações de simetria.

Para melhor representar o funcionamento do algoritmo, assuma-se que foi passado como parâmetro o raio $r = 1$, a transformação de reflexão e a simetria interna desejada igual a 6. O primeiro passo do algoritmo será gerar todas as vizinhanças do espaço, que no nosso exemplo representa o conjunto mostrado na Eq. (22)

$$\{(1,1,1),(1,1,0),(1,0,1),(1,0,0),(0,1,1),(0,1,0),(0,0,1),(0,0,0)\} \tag{22}$$

Em seguida, a transformação escolhida (no caso reflexão) é aplicada a cada uma das vizinhanças, gerando assim o conjunto mostrado na Eq. (23).

$$\begin{aligned}\{&((1,1,1),(1,1,1)),((1,1,0),(0,1,1)),((1,0,1),(1,0,1)),((1,0,0),(0,0,1)),\\&((0,1,1),(1,1,0)),((0,1,0),(0,1,0)),((0,0,1),(1,0,0)),((0,0,0),(0,0,0))\}\end{aligned} \tag{23}$$

Dos pares de vizinhanças encontrados, guarda-se então na variável v o número de vizinhanças invariantes à transformação escolhida e removem-se os pares de vizinhança idênticos. Com isso, o conjunto resultante estará de acordo com a Eq. (24) e v será igual a 4, para a transformação por reflexão no raio 1.

$$\{((1,1,0),(0,1,1)),((1,0,0),(0,0,1)),((0,1,1),(1,1,0)),((0,0,1),(1,0,0))\} \tag{24}$$

Nesse momento, caso o valor de simetria interna passado como parâmetro seja menor que a quantidade de vizinhanças invariantes, o algoritmo retorna como resultado um conjunto vazio, tendo em vista que o valor mínimo de simetria interna é igual ao total de vizinhanças invariantes. Caso contrário o algoritmo agora removerá todas as repetições de pares de vizinhanças, considerando que pares de vizinhanças em ordem diferentes são repetições. A Eq. (25) representa como ficará o conjunto após as remoções.

$$\{(((1,1,0),(0,1,1)),((1,0,0),(0,0,1))\} \tag{25}$$

Cada vizinhança é então substituída pela variável de template correspondente à sua posição, gerando assim um conjunto que representa as equivalências entre variáveis de um template. Esse conjunto de equivalências podem ser melhor visualizados pela Eq. (26) e pelo sistema de equação equivalente representado pela Eq. (27).

$$\{(x_6, x_3),(x_4, x_1)\} \tag{26}$$

$$\begin{cases} x_4 &= x_1 \\ x_6 &= x_3 \end{cases} \tag{27}$$

Igualar todas as variáveis correspondentes à vizinhança gerando o sistema de equações é útil para encontrar a máxima simetria interna de uma determinada transformação. Mas para encontrar valores de simetria arbitrários em uma família de ACs é necessário que o sistema de equações apresente algumas inequações, como mostrado nas Eq. (28) e Eq. (29).

$$\begin{cases} x_4 & \neq & x_1 \\ x_6 & = & x_3 \end{cases} \tag{28}$$

$$\begin{cases} x_4 & = & x_1 \\ x_6 & \neq & x_3 \end{cases} \tag{29}$$

A relação de desigualdade é representada nos templates por meio de uma função que sempre dê um resultado diferente. No caso binário essa função é $1 - x_i$. Logo, os sistemas representados pela Eq. (28) e Eq. (29) são, respectivamente, equivalentes a Eq. (30) e Eq. (31)

$$\begin{cases} x_4 & = & 1 - x_1 \\ x_6 & = & x_3 \end{cases} \tag{30}$$

$$\begin{cases} x_4 & = & x_1 \\ x_6 & = 1- & x_3 \end{cases} \tag{31}$$

Para montar essas equivalências, o algoritmo particiona o conjunto de equivalência em $(s-v)/2$ partes, sendo que s é o valor de simetria interna passado como argumento e v é o número de vizinhanças invariantes à transformação determinada. Após o particionamento, o algoritmo une as partições com os complementos do conjunto, adicionando assim as desigualdades nos pares do complemento. Para os exemplos dados, $s = 6$ e $v = 4$, o que resulta no conjunto representado pela Eq. (32).

$$C_{reflex6} = \{((x_4, x_1), (x_6, 1 - x_3)), ((x_4, 1 - x_1), (x_6, x_3))\} \tag{32}$$

O conjunto $C_{reflex6}$ representa as equivalências entre variáveis que, quando aplicadas ao template base, originam os templates que representam as regras com simetria interna 6. Ao realizar essas substituições obtemos o conjunto de templates representado pela Eq. (33).

$$\{(x_7, 1 - x_3, x_5, x_1, x_3, x_2, x_1, x_0), (x_7, x_3, x_5, 1 - x_1, x_3, x_2, x_1, x_0)\} \tag{33}$$

Esse conjunto de templates representam todas as regras com valor de simetria por reflexão igual a 6 do espaço elementar.

Esse algoritmo teve sua primeira implementação apresentada por de Oliveira e Verardo (2014a; 2014b). Posteriormente uma nova versão foi apresentada em (VERARDO, 2014).

5.3 Templates para Regras Totalísticas e Semi-Totalísticas

A partir das definições de como devem ser as transições de estados dos ACs totalísticos e dos ACs semi-totalísticos, demonstradas na Subseção 3.3, é possível representá-los por meio de templates. A biblioteca *CATemplates* já apresenta os algoritmos que geram essas propriedades e seu funcionamento é bem simples.

O algoritmo que gera regras totalísticas recebe como argumento os valores de r e k, definindo assim uma família de ACs. Em seguida enumera as vizinhanças do espaço e calcula a soma dos estados de cada uma delas. O resultado da soma é o que define quais transições devem ser iguais para que o template represente apenas regras totalísticas.

Para determinar qual será a variável associada, dada uma vizinhança qualquer, o algoritmo verifica se a vizinhança foi a única até então que obteve um determinado valor de soma. Em caso positivo o algoritmo criará e associará uma variável x_i para esta transição, sendo i o valor decimal da vizinhança. Caso contrário, a transição será associada a uma variável x_i em que i é o valor decimal da primeira vizinhança encontrada com o mesmo valor de soma das vizinhanças; observe-se que esta vizinhança será a de menor valor, uma vez que a construção do template se dá a partir de $i = 0$.

A Eq. (34) representa o template para o espaço elementar gerado pelo processo descrito acima. Esse template, quando expandido, gera $k^m = 2^4 = 16$ regras.

$$(x_7, x_3, x_3, x_1, x_3, x_1, x_1, x_0) \tag{34}$$

Já para a família de ACs de $k = 3$ e $r = 1$, o algoritmo gera o template mostrado na Eq. (35), sendo que esse template representa as $k^m = 3^7 = 2.187$ regras totalísticas do espaço.

$$\begin{aligned}(&x_{26}, x_{17}, x_8, x_{17}, x_8, x_5, x_8, x_5, x_2, x_{17}, x_8, x_5, x_8,\\ &x_5, x_2, x_5, x_2, x_1, x_8, x_5, x_2, x_5, x_2, x_1, x_2, x_1, x_0)\end{aligned} \tag{35}$$

Para as regras semi-totalísticas, o algoritmo segue o mesmo processo básico, porém

com uma pequena modificação: as vizinhanças são somadas, mas a célula central é desconsiderada na soma e posteriormente concatenada ao final do resultado. Neste processo, as vizinhanças que contenham o mesmo estado na célula central e a mesma soma de valores dos estados das células externas (como as vizinhanças $(2, 1, 0)$ e $(1, 1, 1)$) geram o mesmo valor após a transição de estado. A partir desse ponto o algoritmo segue o mesmo processo aplicado às regras totalísticas: para determinar qual será a variável associada a uma vizinhança o algoritmo verifica se a vizinhança foi a única até então que obteve um determinado valor. Em caso positivo o algoritmo criará e associará uma variável x_i para esta transição, sendo i o valor decimal da vizinhança. Caso contrário, a transição será associada uma variável x_i em que i é o valor decimal da primeira (e de menor valor) vizinhança encontrada com o mesmo valor de soma das vizinhanças.

Aplicando esse algoritmo gerador de templates de regras semi-totalísticas com os argumentos $k = 2$ e $r = 1$ é obtido o template mostrado na Eq. (36).

$$(x_7, x_3, x_5, x_1, x_3, x_2, x_1, x_0) \tag{36}$$

Quando expandido, o template da Eq. (36) gera as $k^m = 2^6 = 64$ regras semi-totalísticas do espaço elementar.

Para a família de ACs com $k = 3$ e $r = 1$, o algoritmo gera o template da Eq. (37), que quando expandida gera as $k^m = 3^{15} = 14.348.907$ regras semi-totalísticas do espaço.

$$\begin{aligned}(&x_{26}, x_{17}, x_8, x_{23}, x_{14}, x_5, x_{20}, x_{11}, x_2, x_{17}, x_8, x_7, x_{14},\\ &x_5, x_4, x_{11}, x_2, x_1, x_8, x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0)\end{aligned} \tag{37}$$

5.4 Templates Para Regras Confinadas

A propriedade estática de confinamento pode ser facilmente representada através de templates. Para isto, basta restringir as variáveis a um conjunto de valores presentes na vizinhança correspondente. A biblioteca aberta *CATemplates* também já apresenta um algoritmo que gera as regras confinadas. Esse algoritmo recebe como parâmetro os argumentos k e r, gera as vizinhanças do espaço, e verifica em cada uma das vizinhanças os estados que elas têm. Caso a vizinhança tenha todos os estados do intervalo $[0, k-1]$ essa posição terá uma variável livre no template. Caso a vizinhança tenha apenas um estado, a posição correspondente no template recebe uma constante. Por fim, caso a vizinhança apresente mais de um estado, mas não todos, a posição correspondente do template apresentará uma variável restrita pela expressão $x_i \in C$.

A Eq. (38) representa o template de todas as regras confinadas do espaço elementar e a Eq. (39) representa a família de $k = 2$ e $r = 0, 5$.

$$(1, x_6, x_5, x_4, x_3, x_2, x_1, 0) \tag{38}$$

$$(1, x_2, x_1, 0) \tag{39}$$

Por fim, a Eq. (40) representa a família dos autômatos celulares confinados de $r = 1$ e três estados.

$$\begin{aligned}(2, x_{25} \in \{1,2\}, x_{24} \in \{0,2\}, x_{23} \in \{1,2\}, x_{22} \in \{1,2\}, x_{21}, x_{20} \in \{0,2\}, x_{19}, x_{18} \in \{0,2\},\\ x_{17} \in \{1,2\}, x_{16} \in \{1,2\}, x_{15}, x_{14} \in \{1,2\}, 1, x_{12} \in \{0,1\}, x_{11}, x_{10} \in \{0,1\}, x_9 \in \{0,1\},\\ x_8 \in \{0,2\}, x_7, x_6 \in \{0,2\}, x_5, x_4 \in \{0,1\}, x_3 \in \{0,1\}, x_2 \in \{0,2\}, x_1 \in \{0,1\}, 0)\end{aligned} \tag{40}$$

6 DIFERENÇA ENTRE TEMPLATES

Neste capítulo apresenta-se o desenvolvimento da operação de diferença entre templates com $k = 2$, introduz-se a operação de geração de templates de exceção, e demonstra-se uma série de passos que, utilizando templates, pode auxiliar na busca de ACs de raio 3 que tem a possibilidade de solucionar o problema de paridade.

6.1 Templates de Exceção

Alguns templates apresentam em seus campos valores que, quando expandidos, gerarão substituições inválidas e, por consequência, regras inválidas. O template $T_o = (x_7, x_6, x_5, 1 - x_1 - x_5, 2 - x_1 - x_2, x_2, x_1, 0)$ com $k = 2$ é um exemplo disso. É trivial perceber que qualquer expansão do template T_o que tenha o conjunto de substituições $\{x_1 = 1, x_5 = 1\}$ fará com que a posição 4 do template apresente o valor -1, que não pertence ao intervalo $[0, k-1]$; da mesma maneira, qualquer expansão do template T_o que tenha o conjunto de substituições $\{x_1 = 0, x_2 = 0\}$ fará com que a posição 3 do template apresente o valor 2 que também não pertence ao intervalo $[0, k-1]$. Um *template de exceção* é um template que apresenta esses conjuntos substituições que levam um template passado com parâmetro a apresentar substituições fora do intervalo inteiro $[0, k-1]$. Já o conjunto de *templates de exceção* de um template representa todos os templates de exceção encontrados a partir do template passado como parâmetro. O conjunto de templates de exceção do template T_o, utilizado como exemplo, pode ser representado pela Eq. 41.

$$C_e = \{(x_7, x_6, 1, x_4, x_3, 1, x_2, x_0), (x_7, x_6, x_5, x_4, x_3, 0, 0, x_0)\} \tag{41}$$

Formalmente, a operação geradora de templates de exceção X gera um conjunto C_e com todos os templates de exceção do template T_o passado como parâmetro. Ou seja, a operação que gera o conjunto templates de exceção de um template pode ser descrita da seguinte maneira:

$$\begin{aligned} X(T_o) &= C_e \\ C_e &= \{T_1, T_2, \ldots, T_n\} \end{aligned} \tag{42}$$

O algoritmo dessa operação primeiro encontra todas as posições que não apresentem apenas variáveis livres ou constantes. O algoritmo gera então para cada uma das posições

encontradas todas as substituições possíveis, dentro do intervalo $[0, k-1]$, para cada uma das variáveis presentes na posição. Por fim, verifica se a solução com as substituições possíveis levam valores fora dos intervalos $[0, k-1]$.

Para exemplificar, considere o template $T_o = (1, x_6, x_5, 1-x_1-x_2, 2-x_1-x_2, x_2, x_1, 0)$. O primeiro passo do algoritmo é encontrar um conjunto com as posições que não apresentem apenas constantes ou variáveis livres, obtendo-se assim $\{\{1-x_1-x_2\}, \{2-x_1-x_2\}\}$. Então, aplica-se a cada um dos itens do conjunto obtido todas as combinações possíveis das substituições de x_1 e x_2, conforme mostrado na Tabela 5 e 6 para $\{\{1-x_1-x_2\}, \{2-x_1-x_2\}\}$, respectivamente.

Tabela 5: Expansão do campo 1 - $\mathrm{x}_1 - x_2$.

x_2	x_1	Expansão do campo
0	0	1 - $\mathrm{x}_1 - x_2 = 1$
0	1	1 - $\mathrm{x}_1 - x_2 = 0$
1	0	1 - $\mathrm{x}_1 - x_2 = 0$
1	1	1 - $\mathrm{x}_1 - x_2 = -1$

Tabela 6: Expansão do campo 2 - $\mathrm{x}_1 - x_2$.

x_2	x_1	Expansão do campo
0	0	2 - $\mathrm{x}_1 - x_2 = 2$
0	1	2 - $\mathrm{x}_1 - x_2 = 1$
1	0	2 - $\mathrm{x}_1 - x_2 = 1$
1	1	2 - $\mathrm{x}_1 - x_2 = 0$

Para finalizar a operação, o algoritmo seleciona as substituições que geram valores inválidos, e as aplica ao template base, gerando assim o conjunto de templates C_e, mostrado na Eq. 43.

$$C_e = \{(x_7, x_6, x_5, x_4, x_3, 1, 1, x_0), (x_7, x_6, x_5, x_4, x_3, 0, 0, x_0)\} \tag{43}$$

A operação que gera os templates de exceção foi apresentada por Soares, Verardo e de Oliveira (2016) e é importante por ser essencial para a operação de diferença entre templates.

É importante frisar que essa é uma operação com complexidade $O(c^n)$, onde n é o número de variáveis nas posições que não apresentem apenas variáveis livres ou constantes. Por conta disso, a operação de diferença pode se mostrar demasiadamente custosa quando aplicadas a templates com muitas dependências entre variáveis, tais como os templates de conservabilidade e conservabilidade de paridade.

6.2 Diferença entre Templates binários

A operação D de diferença entre os templates binários T_m e T_s é responsável por obter um conjunto C_d com diversos templates, os quais, quando expandidos (via a operação de expansão E), apresentam apenas as regras geradas pela expansão do template T_m que não estejam também presentes na expansão do template de intersecção entre T_s e T_m, chamado aqui de $I(T_m, T_s)$. Ou seja, a operação de diferença pode ser descrita da seguinte maneira:

$$\begin{aligned} D(T_m, T_s) = C_d \Leftrightarrow E(C_d) &= E(T_m) \setminus E(I(T_m, T_s)) \\ C_d &= \{T_1, T_2, \ldots, T_n\} \end{aligned} \tag{44}$$

A Figura 16 ilustra essa operação, onde o template minuendo T_m e o template subtraendo T_s passados como parâmetros para a operação de diferença estão representado pelos dois círculos, e o conjunto C_d retornado pela função está representado pela área em cinza na imagem.

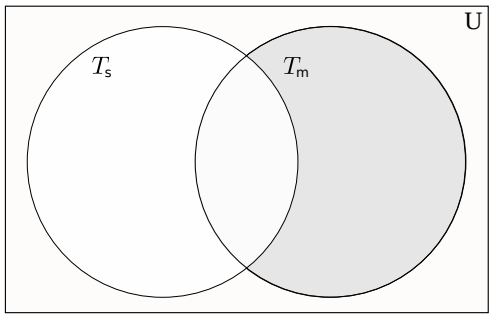


Figura 16: Os círculos T_m e T_s são os templates que representam dois conjuntos originais de regras. Em cinza, C_d é o conjunto de templates que representam o conjunto de regras retornado pela operação de diferença entre T_m e T_s.

O diagrama 17 mostra de forma mais detalhada cada uma das etapas desse algoritmo.

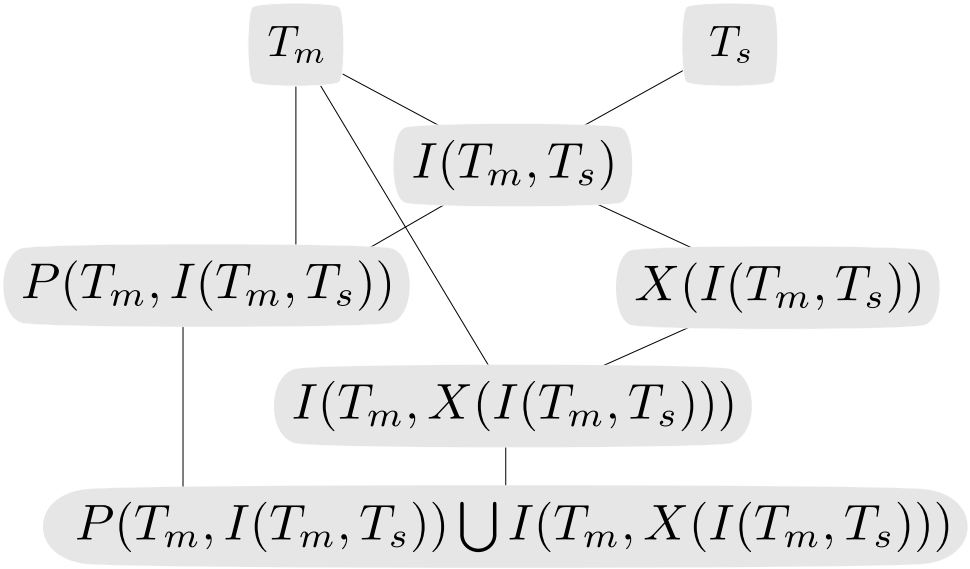


Figura 17: Diagrama representando as principais etapas do algoritmo de diferença entre templates binários.

O processo que o algoritmo usa para encontrar a diferença entre dois templates é efetuado através de uma sequência de etapas. Na primeira etapa a operação $P(T_m, I(T_m, T_s))$ é executada. Essa operação consiste em igualar o template T_m com o template obtido pela intersecção de T_m com T_s, obtendo-se assim combinações lógicas de equações; remover eventuais equações tautológicas; aplicar a operação de negação nas equações e trocar o operador lógico $\wedge$ por $\vee$, que, no caso binário, consiste apenas em efetuar as permutações $\rho = (0 \to 1, 1 \to 0, \wedge \to \vee)$ ao resultado final das equações; por fim, solucionar a equação, resultando num conjunto com diversos conjuntos de regras de substituições, as quais são aplicadas ao template $I(T_m, T_s)$, gerando um conjunto de templates que é parte do resultado dessa operação. Complementar a esta etapa, é necessário encontrar o conjunto dos templates de exceção por meio da operação $X(I(T_m, T_s))$ e efetuar a intersecção de cada um dos templates de exceção do template $I(T_m, T_s)$ com o template T_m. O resultado de ambas etapas é então unido, gerando o resultado da operação de diferença.

De forma mais genérica, $I(T_m, X(I(T_m, T_s)))$ é responsável por encontrar os templates com substituições de variáveis que geram regras inválidas para $I(T_m, T_s)$, mas apresentam regras válidas em T_m. Já a operação P se responsabiliza por encontrar os templates que representem regras de T_m que não pertençam a $I(T_m, T_s)$, contanto que as substiuições de variáveis gerem regras válidas nos templates T_m e $I(T_m, T_s)$.

Para melhor compreender a necessidades dessas etapas, considerem-se os templates $T_m = (x_3, x_2, x_1, x_0)$ e $T_s = (x_3, 1, 1 + x_0, x_0)$, ambos com $k = 2$ e $r = 0,5$. No primeiro

passo da operação de diferença, executa-se a operação $P(T_m, I(T_m, T_s))$ que iguala os templates T_m com $I(T_m, T_s)$, o que gera o sistema de equações mostrado na Eq. (45).

$$\begin{cases} x_3 &= & x_3 \\ x_2 &= & 1 \\ x_1 &= & 1 + x_0 \\ x_0 &= & x_0 \end{cases} \tag{45}$$

Esse sistema, deve então ser representado por meio de combinações lógicas de equações, conforme a Eq. (46).

$$x_3 = x_3 \wedge x_2 = 1 \wedge x_1 = 1 + x_0 \wedge x_0 = x_0 \tag{46}$$

Antes de solucionar a Eq. (46), o algoritmo eliminam todas as equações tautológicas e troca-se cada operador lógico $\wedge$ por $\vee$; como resultado, obtém-se a Eq. (47).

$$x_2 = 1 \vee x_1 = 1 + x_0 \tag{47}$$

Ainda antes de solucionar a equação, aplica-se a operação de negação binária nas equações por meio da função $f(x) = 1 - x$. A Eq. (48) representa a combinação lógica de equações que resulta das operações precedentes.

$$x_2 = 1 - 1 \vee x_1 = 1 - (1 + x_0) \tag{48}$$

A etapa de troca do operador lógico e de permutação ρ são a base desta operação, pois essas etapas determinam que se qualquer campo de um template resultar numa regra com um valor diferente do esperado no template original, esse campo deve ser considerado para a criação dos templates de diferença.

Por fim, a combinação lógica de equações da Eq. (48) é solucionada, resultando no conjunto solução $S = \{\{x_1 = -x_0\}, \{x_2 = 0\}\}$. Perceba-se que S apresenta mais de um conjunto de substituições e, portanto, cada um deles deve ser utilizado para realizar as substituições no template T_m. Essas substituições fazem com que como resultado da operação $P(T_m, I(T_m, T_s))$ se obtenha o conjunto de templates representados pela Eq. (49).

$$\begin{gathered} \{(x_3, 0, x_1, x_0), \\ (x_3, x_2, -x_0, x_0)\} \end{gathered} \tag{49}$$

Sempre que $I(T_m, T_s)$ não apresentar templates de exceção, apenas etapa P já seria suficiente para obter os templates que representam todas as regras que pertençam a T_m, mas não pertençam a T_s. Neste exemplo isso não ocorre, e é fácil perceber que a regra $(1,1,1,1) \in E(T_m)$ e $(1,1,1,1) \notin E(T_s)$; logo, essa é uma regra que deveria aparecer na operação D que realiza a diferença entre esses dois templates, mas isso não ocorreria utilizando apenas a operação P. O motivo é que a operação P, ao realizar a igualdade entre os templates T_m e T_s, pode trazer para os templates resultantes as características do template T_s que geram expansões inválidas. Deste modo, cada template gerado por $P(T_m, I(T_m, T_s))$ representa um template T_m com uma variável em posição correspondente em $I(T_m, T_s)$ negada. E como a negação só pode ser feita em valores binários, essa operação representa todas as regras de T_m com alguma variável binária em posição correspondente $I(T_m, T_s)$ negada. Para as variáveis não binárias, se faz necessário gerar os template de exceção com a operação $X(I(T_m, T_s))$, que resulta no conjunto de templates $\{(x_3, x_2, x_1, 1)\}$ (no caso, só com 1 elemento). Os templates desse conjunto são então interseccionados com o template T_m, resultando no conjunto representados pela Eq. (50).

$$\{(x_3, x_2, x_1, 1)\} \tag{50}$$

Por fim, faz-se a unificação dos conjuntos das equações Eq. (49) e Eq. (50), obtendo-se o conjunto de templates C_d, representado pela Eq. 51.

$$\begin{aligned} C_d = \{&(x_3, 0, x_1, x_0), \\ &(x_3, x_2, -x_0, x_0), \\ &(x_3, x_2, x_1, 1)\} \end{aligned} \tag{51}$$

A operação de diferença D também pode ser realizada sem se efetuar a intersecção. Nesse caso, a operação sem intersecção, chamada aqui apenas por D, encontrará um conjunto de templates C_d. A expansão dos templates de C_d apresentará apenas as regras geradas pela expansão do template T_m que não estejam também presentes na expansão do template T_s. Essa operação pode ser apresentada da seguinte maneira:

$$\begin{aligned} D(T_m, T_s) = C_d \Leftrightarrow E(C_d) = E(T_m) \setminus E(T_s) \\ C_d = \{T_1, T_2, \ldots, T_n\} \end{aligned} \tag{52}$$

O processo que o algoritmo da operação D usa para encontrar a diferença entre dois templates é bem parecido com o do algoritmo da operação D. Primeiramente, igualam-se

os templates T_m e T_s, obtendo-se assim combinações lógicas de equações. Após remover eventuais equações tautológicas, o algoritmo aplica a operação de negação nas equações e troca o operador lógico $\wedge$ por $\vee$. Esse sistema é então solucionado, resultando num conjunto com diversos conjuntos de regras de substituições. E após eliminar desse conjunto eventuais substituições envolvendo variáveis que não pertençam ao T_m, esses conjuntos de regras de substituição são aplicados ao template T_m, gerando o conjunto de templates que é parte do resultado dessa operação. Um segundo conjunto de templates, composto pela intersecção de cada um dos templates de exceção do template T_s com o template T_m, será a outra parte do resultado.

Similar a operação de diferença com intersecção, $I(T_m, X(T_s))$ é responsável por encontrar os templates com substituições de variáveis que geram regras inválidas para T_s, mas apresentam regras válidas em T_m. Já a operação P se responsabiliza por encontrar os templates que representem regras válidas de T_m que não pertençam a T_s, contanto que as substiuições de variávei gerem regras válidas também em T_s.

O diagrama 18 detalha cada uma das etapas da operação de diferença sem intersecção.

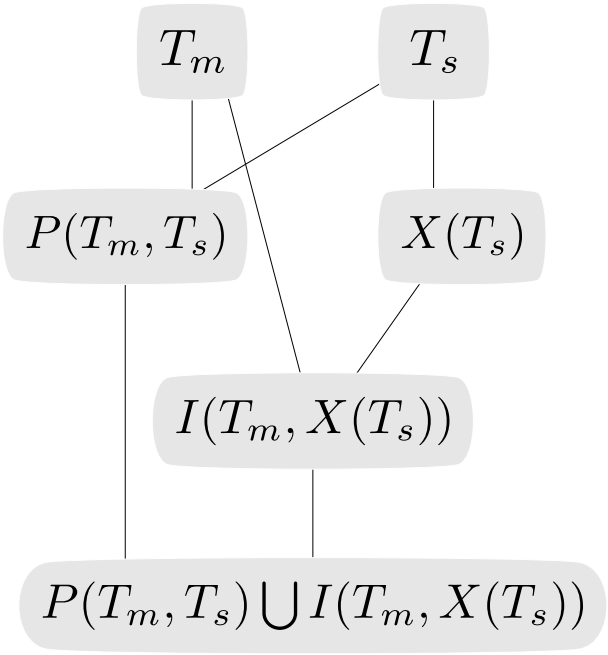


Figura 18: Diagrama representando as principais etapas do algoritmo de diferença entre templates binários, sem a utilização da operação de intersecção.

Os algoritmos D, com ou sem intersecção, apresentam como resultado conjuntos que podem ser diferentes mesmo quando ambos recebem os mesmos templates como entrada.

Mas ambos contêm exatamente as mesmas regras quando se expandem os templates dos conjuntos.

7 DISCUSSÃO E TESTES

A implementação das operações de diferença foi desenvolvida na linguagem do software *Wolfram Mathematica* (WOLFRAM RESEARCH, 2015) e na mesma linguagem foram criados os testes unitários que verificam se a saída da operação está correta de acordo com os possíveis dados de entrada. Todos os testes receberam dois templates, T_m e T_s, expandiram esses dois templates individualmente, e depois subtraíram das regras encontradas na expansão de T_m todas as regras representadas por T_s. Se o resultado dessa diferença for idêntico às regras geradas pela expansão dos templates resultantes da operação de diferença entre os templates T_m e T_s, o teste foi bem sucedido. A Eq. 53 ilustra a igualdade que deve ser satisfeita para o teste retornar ser bem sucedido.

$$E(T_m) \setminus E(T_s) = E(D(T_m, T_s)) \tag{53}$$

Por conta desta custosa forma de teste que realiza operações de força bruta, enumerando todas as regras dos templates recebidos como parâmetro, os testes dessa operação se concentraram nos raios 1, 2 e 3.

Uma importante questão sobre a expansão do conjunto de regras C_{exp}, encontrado pela operação de diferença, é definir qual será o pós-processamento da operação de expansão que deve ser usado com esses templates. A operação de pós-processamento a ser utilizada na operação de diferença sempre será a mesma utilizada no template T_m passado como parâmetro, acrescida da operação de filtro das regras inválidas caso essa operação já não tenha sido feita. Isto porque a operação de diferença deve representar todas as regras válidas da expansão de T_m que não sejam representadas por T_s. Assim, caso T_m e T_s não tenham intersecção, o resultado deve ser apenas o template T_m com o mesmo pós-processamento.

Nos testes para todos os raios foi gerado um conjunto de templates e, a partir dele, geraram-se todos os pares de combinações possíveis. Por fim, foi realizado o teste da operação de diferença com cada um dos pares de template gerados.

Para os testes com $r = 1$, cada um dos templates do conjunto usado para os testes representava uma das seguintes propriedades estáticas: conservabilidade de estados; confinamento; totalidade; semi-totalidade; e máxima simetria interna por conjugação, propriedade também conhecida como invariância a troca de cor. Além desses templates,

também foi utilizado o template base, que representa todas as regras do espaço.

Para os testes com $r = 2$, os templates usados representavam as seguintes propriedades estáticas: conservabilidade de estados; totalidade; semi-totalidade; e invariância à troca de cor. Já para os testes $r = 3$, os templates de teste representavam apenas as propriedades estáticas de totalidade e semi-totalidade.

Foram feitos testes com menos propriedades de acordo com o raio pois esses testes envolvem força bruta, e para algumas dessas propriedades consistia em enumerar um número muito grande de regras. Assim os testes foram feitos apenas com propriedades em que a expansão dos templates encontrados consistia em enumerar menos que 2^{16} regras. A Tabela 7 mostra em mais detalhes os templates de quais propriedades estáticas foram utilizados para os testes da operação de diferença e em quais tamanhos de raio.

Tabela 7: Propriedades estáticas utilizadas para o teste da operação de diferença em cada raio

	Template minuendo (T_m)				
Template subtraendo (T_s)	Totalidade	Semi-totalidade	Conservabilidade de estados	Invariância à troca de cor	Confinamento
Totalidade	Raio 1, 2 e 3	Raio 1, 2 e 3	Raio 1 e 2	Raio 1 e 2	Raio 1
Semi-totalidade	Raio 1, 2 e 3	Raio 1, 2 e 3	Raio 1 e 2	Raio 1 e 2	Raio 1
Conservabilidade de estados	Raio 1 e 2	Raio 1 e 2	Raio 1 e 2	Raio 1 e 2	Raio 1
Invariância à troca de cor	Raio 1 e 2	Raio 1 e 2	Raio 1 e 2	Raio 1 e 2	Raio 1
Confinamento	Raio 1	Raio 1	Raio 1	Raio 1	Raio 1

Ademais, esses testes mostraram que a operação de diferença está funcionando para diversos templates, sem a necessidade de operações específicas para cada um dos templates. É importante lembrar que a implementação do algoritmo que executa a operação de diferença entre templates ainda não permite trabalhar com $k \neq 2$, pois a negação das equações são feitas por meio da função $f(x) = 1 - x$.

7.1 Templates aplicados no problema de paridade

O desenvolvimento da operação de diferença entre template permite diversas possibilidades de aplicação. Uma das possibilidades é no problema de paridade. Ainda não se sabe se existem regras de raio 3 que solucionem o problema de paridade (BETEL; DE OLIVEIRA; FLOCCHINI, 2013). Mas o uso de templates e suas operações podem ser uma forma interessante de restringir o conjunto de regras na busca dessa solução.

As regras dos ACs que solucionam o problema de paridade têm algumas propriedades

estáticas que podem ser trivialmente percebidas. Um AC que resolva o problema de paridade sempre será confinado, tendo em vista que as vizinhanças homogêneas não devem levar a transições de estado ativas, a qual é a única restrição de variável nos templates confinados para ACs binários. O espaço das regras confinadas de raio 3 ainda é muito grande; entretanto, essa não é a única propriedade estática que um AC que resolva o problema de paridade apresenta. Nesse sentido, também é necessário que ele não seja conservativo, visto que, se a soma dos estados do AC não mudar, ele nunca convergirá como propõe o problema. Por fim, espera-se que um AC que resolva o problema de paridade seja conservativo de paridade.

Dada a possibilidade de se obter os templates para todas essas propriedades, é também possível utilizar esses templates e as operações de diferença e intersecção para expressar esse espaço de busca para a solução do problema de paridade. Para expressar esse espaço, basta efetuar a intersecção dos templates de confinamento com o template de conservabilidade de paridade, visto que ambas as propriedades são necessárias. Entretanto, vale frisar que essa intersecção gera o próprio template de paridade, não fazendo diferença para o resultado final. Posteriormente, deve-se efetuar a operação de diferença com o template obtido pela primeira intersecção com o template das regras conservativas de estado, já que a conservabilidade de estado impede a resolução do problema de paridade.

Formalmente, essas operações de conjuntos entre os templates podem ser representadas através da Eq. (54), sendo que $T_{confinado}$ representa o template das regras confinadas, $T_{conservaparidade}$ representa o templates das regras que conservam a paridade, e $T_{conservaestados}$ indica o template de regras conservativas. O resultado dessa operação é $T_{paridade}$, o qual representa um conjunto de templates que restringem um pouco mais as regras com possibilidades de solucionar o problema de paridade.

$$T_{paridade} = (T_{conservaparidade} \cap T_{confinado}) - T_{conservaestados} \tag{54}$$

Entretanto, apesar de formalmente possível, a realização dessas operações não se mostrou produtiva. Isto ocorre pois a operação de exceção, utilizada pela operação de diferença, tem complexidade $O(c^n)$, onde n é o número das variáveis c com posições que não apresentem apenas variáveis livres ou constantes nos templates. Deste modo, gerar os templates de exceção do template $T_{conservaestados}$ é tão custoso quanto expandir todas as regras desse template.

Dito isto, é mais simples utilizar o template $T_{conservaestados}$ com uma função de pós-processamento alternativa que elimine todas as tabelas k-árias que não apresentem campos

inválidos, e aplique *mod* 2 nas restantes. Assim esse mesmo template vai representar apenas as regras que mantenham a paridade mas não apresentem a propriedade de conservabilidade de estados. Também poderá ser aplicado nesse template operações de intersecção ou diferença com outros templates de modo a se restringir ainda mais o espaço de busca. Por fim, a busca por melhorias na implementação da operação de diferença de modo que a faça gerar um espaço de busca menor também se mostra necessário.

8 CONSIDERAÇÕES FINAIS

No presente trabalho são descritos os templates de autômatos celulares unidimensionais, proposta introduzida por de Oliveira e Verardo (2014a), que, por meio de uma generalização de suas tabelas de transição k-árias, pode representar conjuntos de regras que compartilham determinadas propriedades.

Essa capacidade faz com que não seja necessário buscar ACs com essas propriedades por todo um espaço, o que impossibilitaria a busca através de força bruta, devido ao rápido crescimento das famílias dos ACs conforme se mudam seus parâmetros. Adicionalmente, mesmo buscas heurísticas também ficam facilitadas no sub-espaço definido pelos templates.

Por conta dessa capacidade de representação de ACs com determinadas propriedades, foram descritos aqui algumas propriedades estáticas e os algoritmos geradores dos templates que as representam. Esses algoritmos já estavam implementados na biblioteca *open source CATemplates* (VERARDO; DE OLIVEIRA, 2015). Além disso, foram explicadas as operações de expansão e intersecção do *CATemplates*. Essas operações, desenvolvidas por Verardo (2014), foram mostradas novamente aqui para que fosse possível explicitar sua relevância para a solução de problemas típicos de ACs, como o problema de paridade.

Também apresentamos a operação de diferença entre templates e a operação geradora de templates de exceção, ambas introduzidas por Soares, Verardo e de Oliveira (2016). A operação de diferença entre templates permite que se encontre um conjunto de templates que represente todas as regras que não pertençam ao template passado como argumento. Esta operação – que no momento só aceita templates binários – já está disponível na biblioteca *CATemplates*, e é mais um exemplo de operação que pode ser utilizada para restringir o conjunto de regras a serem avaliadas na busca pela solução de algum problema.

Para que fosse possível a implementação da operação de diferença, foi desenvolvida a operação que, dado um template, gera os *templates de exceção* correspondentes. *Templates de exceção* são gerados a partir de templates base, substituindo-se algumas variáveis pelas substituições que geram regras inválidas no template original. O algoritmo que gera templates de exceção é essencial para a operação de diferença.

Também foi exposto nesse trabalho um conjunto de processos utilizando templates que podem auxiliar na restrição do espaço de busca para a solução do problema de paridade.

Como possíveis trabalhos futuros, pretende-se generalizar a operação de diferença para qualquer valor de k, bem como pretende-se implementar novos algoritmos geradores de templates que representem outras propriedades estáticas. Ademais, pode-se buscar também a implementação de novas operações de templates baseadas nas operações de conjuntos, como a operação de união de templates.

REFERÊNCIAS BIBLIOGRÁFICAS

BETEL, H.; DE OLIVEIRA, P. P. B.; FLOCCHINI, P. Solving the parity problem in one-dimensional cellular automata. *Natural Computing*, v. 12, n. 3, p. 323–337, 2013.

BOCCARA, N.; FUKŚ, H. Number-conserving cellular automaton rules. *Fundamenta Informaticae*, IOS Press, v. 52, n. 1-3, p. 1–13, 2002.

DE BRUIJN, N. A combinatorial problem. *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen. Series A*, v. 49, n. 7, p. 758, 1946.

DE OLIVEIRA, P. P. B.; VERARDO, M. Representing families of cellular automata rules. *The Mathematica Journal*, v. 16, n. 8, 2014. Disponível em: <dx.doi.org/doi:10-.3888/tmj.16-8>.

DE OLIVEIRA, P. P. B.; VERARDO, M. Template based representation of cellular automata rules. In: ISOKAWA, T.; IMAI, K.; MATSIU, N.; PEPER, F.; UMEO, H. (Ed.). *20th International Workshop on Cellular Automata and Discrete Complex Systems*. Himeji, Japão, Julho 7-9: [s.n.], 2014. p. 199–204.

GARDNER, M. Mathematical games: The fantastic combinations of john conway's new solitaire game "life". *Scientific American*, vol, v. 223, n. 4, p. 120–123, 1970.

GOOD, I. J. Normal recurring decimals. *Journal of the London Mathematical Society*, Oxford University Press, v. 1, n. 3, p. 167–169, 1946.

PHILLIPS, R.; WEISSTEIN, E. W. Outer-totalistic cellular automaton. From MathWorld–A Wolfram Web Resource, 2015. Disponível em: <http://mathworld-.wolfram.com/Outer-TotalisticCellularAutomaton.html>. Acesso em: 26 jun. 2015.

SCHRANKO, A.; DE OLIVEIRA, P. P. B. Towards the definition of conservation degree for one-dimensional cellular automata rules. *J. Cellular Automata*, v. 5, p. 383–401, 2010.

SOARES, Z.; VERARDO, M.; DE OLIVEIRA, P. P. B. The difference operation between templates of binary cellular automata. In: ROCHA, Á.; CORREIA, M. A.; ADELI, H.; REIS, P. L.; TEIXEIRA, M. M. (Ed.). *New Advances in Information Systems and Technologies*. [S.l.]: Springer, 2016. p. 707–715.

THEYSSIER, G. Captive cellular automata. In: *Mathematical Foundations of Computer Science 2004*. [S.l.]: Springer, 2004. p. 427–438.

VERARDO, M. *Representando famílias de autômatos celulares por meio de templates*. Dissertação (Mestrado) — Universidade Presbiteriana Mackenzie, 2014.

VERARDO, M.; DE OLIVEIRA, P. P. B. *CATemplates*. [S.l.], 2015. Disponível em: <https://github.com/mverardo/CATemplates>.

VON NEUMANN, J.; BURKS, A. W. Theory of self-reproducing automata. University of Illinois Press, 1966.

WEISSTEIN, E. W. De Bruijn graph. From MathWorld–A Wolfram Web Resource, 2015. Disponível em: <http://mathworld.wolfram.com/deBruijnGraph.html>. Acesso em: 01 set. 2015.

WEISSTEIN, E. W. Moore neighborhood. From MathWorld–A Wolfram Web Resource, 2015. Disponível em: <http://mathworld.wolfram.com/MooreNeighborhood.html>. Acesso em: 19 mai. 2015.

WEISSTEIN, E. W. von Neumann neighborhood. From MathWorld–A Wolfram Web Resource, 2015. Disponível em: <http://mathworld.wolfram.com/vonNeumannNeighborhood.html>. Acesso em: 19 mai. 2015.

WOLFRAM RESEARCH. *Wolfram Mathematica*. 2015. Disponível em: <http://www.wolfram.com/mathematica/>.

WOLFRAM, S. Statistical mechanics of cellular automata. *Reviews of modern physics*, APS, v. 55, n. 3, p. 601, 1983.

WOLFRAM, S. *Cellular automata and complexity: collected papers*. [S.l.]: Addison-Wesley Reading, 1994.

WOLFRAM, S. *A new kind of science*. [S.l.]: Wolfram Media, Champaign, 2002.