UNIVERSIDADE PRESBITERIANA MACKENZIE
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA ELÉTRICA

Zorandir Soares Junior

Aperfeiçoamento Da Representação De Autômatos Celulares Por Meio Templates

Orientador: Prof. Dr. Pedro Paulo Balbi de Oliveira

São Paulo
2015

UNIVERSIDADE PRESBITERIANA MACKENZIE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Zorandir Soares Junior

Aperfeiçoamento Da Representação De Autômatos Celulares Por Meio Templates

Projeto de pesquisa apresentado ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Presbiteriana Mackenzie como parte dos requisitos para a concessão de bolsa.

Orientador: Prof. Dr. Pedro Paulo Balbi de Oliveira

São Paulo
2015

RESUMO

Templates são representação formais para conjuntos de autômatos celulares (ACs) feitas por meio da generalização das tabelas de transição clássicas. Já existem algoritmos que geram templates para propriedades estáticas e algoritmos que realizam operações como intersecção entre templates e expansão de template. Neste projeto é introduzido a operação de complemento de templates, assim como explicado o funcionamento do algoritmo dessa operação implementada na biblioteca *CATemplates*. Também são apresentados exemplos das possibilidades de uso do templates no problema de paridade, com o apoio das operações de complemento e das operações geradoras de templates de ACs conservativos de paridade e ACs conservativos de estado.

Palavras-chave: *Autômatos celulares, templates, conservabilidade, problema de paridade.*

ABSTRACT

Templates are formal representation for cellular automaton sets (CAs) made through the generalization of the classical transition tables. Already there are algorithms that generate templates for static properties and algorithms that perform operations such as intersection between templates and template expansion. In this project it is introduced the templates complement operating, and explained the functioning of the algorithm of this operation implemented in the *CATemplates* package. Also show examples of possibilities of using templates in the parity problem, with the support of complement operations and generating operations of CAs conservative parity and ACs state conservative.

Keywords: *Cellular automaton, templates, conservation, parity problem.*

Sumário

1 INTRODUÇÃO

Autômatos Celulares (ACs) são sistemas dinâmicos discretos em tempo e espaço cuja dinâmica tem sido extensivamente estudada e aplicada em diversas áreas. ACs tem a capacidade de através de regras de comportamentos locais simples, gerar comportamentos globais complexos. De acordo com Wolfram (1994), ACs podem também ser considerados uma idealização discreta de equações parciais diferenciais, muitas vezes utilizadas para descrever sistemas naturais. A Seção 2 apresenta mais detalhadamente os ACs, assim como alguma de suas propriedades.

Existem diversos famílias de autômatos celulares que podem ser estudadas. Devido ao rápido crescimento das famílias dos ACs conforme se mudam seu parâmetros de raio e estado, uma das famílias mais estudadas foi 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, criados por Theyssier (2004), é um dos caso que se pode usar como exemplo. Esses comportamentos e propriedades obtidos através de restrições aplicadas à tabela de transição podem ser denominados como propriedade estática.

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.

Existem algumas formas de representar propriedades estáticas, e essa representação é crucial pois culmina na eliminação da necessidade de se buscar uma propriedade analisando todo o espaço de um AC. Em Li e Packard (1990) já foi introduzido variáveis na representação de conjuntos de tabelas de transição. Por meio de grafos de De Bruijn, BETEL, de OLIVEIRA e FLOCCHINI (2013) representa um conjunto de ACs na busca pela solução do problema de paridade. Mais recentemente de OLIVEIRA e VERARDO (2014) estabeleceu uma representação formal para conjuntos de ACs, denominada *Templates*, assim como esse estudo exemplificou o uso da biblioteca *CATemplates* (VERARDO; de

OLIVEIRA, 2015) desenvolvida na linguagem do software *Wolfram Mathematica* (WOLFRAM RESEARCH, 2015). A Seção 3 apresenta em mais detalhes o funcionamento dos templates, assim como descreve o funcionamentos das operações que os utilizam.

Este estudo visa desenvolver novos algoritmos geradores de templates baseado em propriedades estáticas, assim como apresentar o funcionamento da operação de complemento de templates. Ademais esse trabalho apresenta como a teoria de templates pode ajudar na busca pela solução do problema de paridade.

A Seção 4 apresenta algumas propriedade estáticas e o funcionamento de suas respectivas operações de geração de template. A Seção 5 apresenta alguns exemplos de aplicações para os templates, focado principalmente no problema de paridade. Por fim, a Seção 6 apresenta ideias para trabalhos futuros e a conclusão.

2 AUTÔMATOS CELULARES

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

Uma definição parecida é dada por Weisstein (2015a), para quem um AC é um conjunto de células "coloridas", em um reticulado com forma previamente especificada que evolui ao longo de uma série de passos de tempo discretos, de acordo com um conjunto de regras baseadas nos estados de células vizinhas. As regras são aplicadas de forma iterativa para o número de passos de tempo desejado.

Os autômatos celulares são compostos por células distribuídas em um reticulado, e essas células mudam de estado ao longo de passos de tempo e de acordo com regras locais. O conjunto de células em determinado passo de tempo é denominado configuração ou estado global, e são as configurações que descrevem a evolução espaço temporal de um AC.

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

Independente da dimensionalidade do reticulado, é necessário definir como o AC se comportará nas bordas do reticulado. Um tratamento típico, é aplicar condição de contorno periódica nas extremidades do reticulado. Ou seja, tratar reticulados unidimensionais como um anel, e bidimensionais como toro. Esses tratamentos podem ser visualizados na Figura 1 e na Figura 2, respectivamente.

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

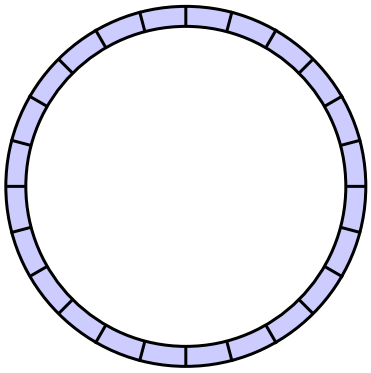

Figura 1: Condição de contorno periódica em um reticulado unidimensional formando um anel.

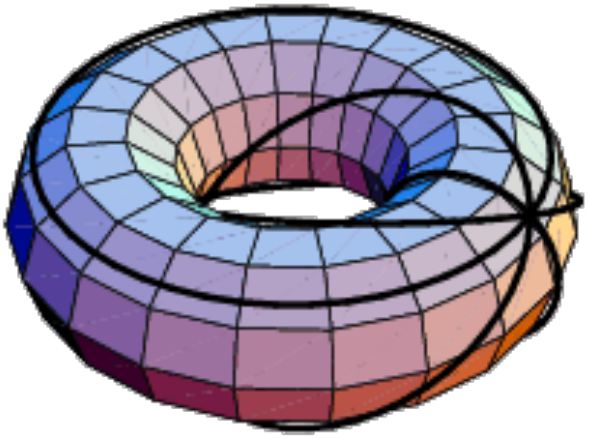

Figura 2: Condição de contorno periódica em um reticulado bidimensional formando um toroide.

das células adjacentes. Para que as funções locais atualizem os valores de uma célula, é necessário que um raio r seja definido. Esse raio r representa o número de células adjacentes que serão analisadas em cada direção pelas funções locais.

Além do raio, é preciso determinar o formato de vizinhança que serão utilizados nos parâmetros da função local. Duas vizinhanças bem comuns em ACs bidimensionais, são as vizinhanças de von Neumann e Moore. Na Figura 3 e na Figura 4 são apresentadas as vizinhanças de von Neumann e Moore, respectivamente, para raios de 1 a 4. No caso dos ACs unidimensionais, as vizinhanças são definidas apenas pelo raio r, e ele descreve quantas células a esquerda e direita da célula atual serão consideras pela função local.

Definindo qual será o raio, o número de estados, o tipo de vizinhança e a dimensionalidade das configurações, é definido uma família de autômatos celulares. Autômatos celulares unidimensionais, de raio $r = 1$ e dois estados, ou seja $k = 2$, são conhecidos

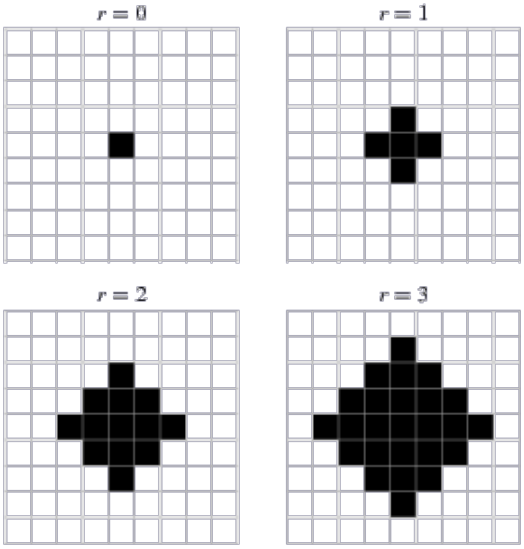

Figura 3: Vizinhança de von VonNeumann com raio r igual a 1, 2, 3 e 4. Essa foi a vizinhança utilizada nos primeiros trabalhos de von Neumann (WEISSTEIN, 2015c).

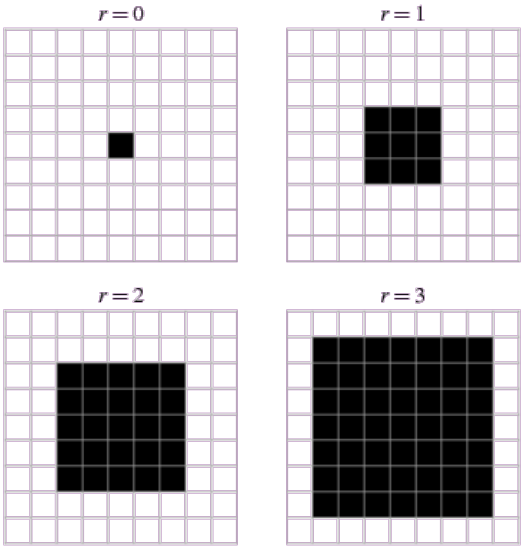

Figura 4: Vizinhança de Moore com raio r igual a 1, 2, 3 e 4. Essa foi a vizinhança utilizada no jogo da vida (WEISSTEIN, 2015b).

como a família dos 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, a mais comum são as tabelas de transições. Tabela de transição é uma $n-$upla em que os elementos são todos os possíveis estados de vizinhanças de uma célula acrescentados de um estado que

representa a transição que ocorrerá. A Equação 1 representa a n-upla 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 os bits 1 são representados por ícones na cor preta, e os bits 0 por ícones na cor branca. Cada uma das transições de estados é representada por um conjunto de ícones representando a vizinhança na parte de cima, e um ícone representando o estado resultante após a transição abaixo. A Figura 5 mostra a representação icônica da regra 30.

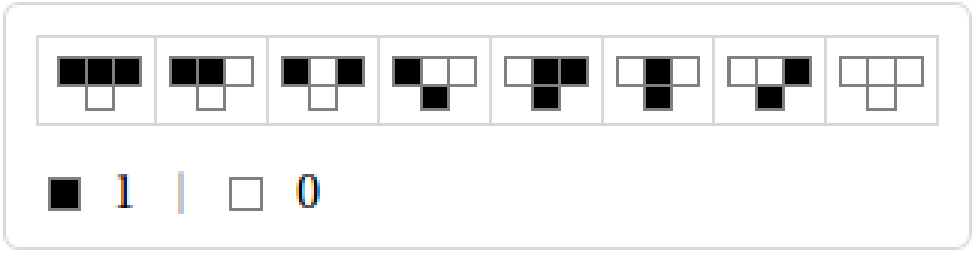

Figura 5: Representação icônica da regra 30.

Além dessas formas de representação, ainda existe a forma k-aria, em que sabendo-se o valor atribuído para k e r, elimina-se a a representação das vizinhanças e deixa-se apenas as representações resultantes. Na Equação 2 é possível ver a representação k-aria da regra 30. O número da regra é obtido ao se converter a representação k-aria para decimal. Esse número é um identificador único em família de autômatos celulares, ou seja, sempre representa apenas uma tabela de transições de estado.

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

O número de regras de um espaço é definido pela Equação 3:

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

No espaço elementar há $2^{2^3} = 256$ regras, aumentando apenas para 2 o valor do raio r, obtemos uma família de $2^{2^5} = 4.294.967.296$ regras. Caso se aumente o número de estados para 3, obtém-se um espaço de ACs com $3^{3^3} = 7.625.597.484.987$ regras. Logo, é fácil perceber como qualquer modificação nas variáveis k e r geram famílias com número de regras muito grande. Famílias de ACs representam um desafio grande 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 muito comum utilizar algumas propriedades estáticas para restringir as regras do espaço em que se fará as buscas pelo regra com o comportamento desejado. Alguns exemplos de propriedades estáticas que podem auxiliar nesse "filtro" são o confinamento, conservabilidade de estados e conservabilidade de paridade. Todas as propriedades estáticas citadas anteriormente são detalhadas na Seção 4.

O problema da paridade é um dos problemas que envolvem fazer a ligação entre comportamento local e comportamento global. Nesse projeto, considerando que um AC binário, unidimensional e com condição de contorno periódica, se uma configuração inicial contiver um número ímpar de estados com valor 1, o AC deve convergir para que todas as células estejam preenchidas com 1. Caso contrário, ele deve convergir para todos os estados com o valor 0. Por conta da própria definição do problema, fica simples perceber que essa condições não podem ser satisfeitas em uma grelha de tamanho par, afinal uma configuração de com todos os estados apresentando o mesmo valor não poderia ser um estado quiescente. Devido essa questão, 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.

Ainda em relação ao problema de paridade, BETEL, de OLIVEIRA e FLOCCHINI (2013) descrevem 2 propriedades básicas: 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 propriedade define que para uma regra preservar a configuração de paridade, a regra deve conter um número par de transições ativas. Em outras palavras, toda aplicação da regra deve levar a uma nova configuração com a mesma paridade.

Procurar uma regra que resolva o problema de paridade em autômatos celulares unidimensionais de raio 3, por exemplo, através de buscas por força bruta acarretaria em testar mais de 340 undecilhões de regras. Uma maneira de facilitar essa busca seria restringir as regras através de propriedades estáticas de conservação de paridade e conservação de estados, mas é necessário uma forma de representar essas propriedades e até mesmo aplicar operações como intersecção e complemento entre elas.

Nesse ponto que os *templates* apresentam-se de uma forma interessantes e útil para representar conjuntos de regras com determinada propriedade. Templates de ACs são uma generalizações das tabelas de transições que permitem representar espaços inteiros

de ACs (VERARDO, 2014).

3 TEMPLATES

Templates de autômatos celulares são uma generalização de tabelas de transições que faz com que um templates seja capaz de representar famílias de autômatos celulares. Os templates foram criados por de OLIVEIRA e VERARDO (2014) e implementada 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-upla formada por k^{2r+1} itens, e cada item i representa uma função $g_i(x_0, x_1, \ldots, x_{k^{2r+1}-1})$. As variáveis x_i podem assumir qualquer estado entre 0 e $k-1$, logo no caso binário x_i pode assumir os valores 0 e 1. É possível 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. Entretanto vale frisar que no caso binário não tem lógica implementar uma notação como $x_i \in 1$ ou $x_i \in 0$ pois os templates também aceitam constante, sendo essas notações equivalente à apenas as constante 1 e 0 respectivamente.

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 em sua primeira posição (sempre da direita pra esquerda) o estado 0, nas posições 5, 6, 7 e 8 o estado 1, nas posições 2 e 3 qualquer estado no intervalo $[0, k-1]$ e na posição 4 o estado complementar ao valor de x_1. Perceba que o tamanho da n-tupla é determinado pela função k^{2r+1}, logo 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 representam 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 sua forma decimal $\{248, 242, 252, 246\}$.

Cada template tem um número de substituições máximo igual a k^m, sendo o 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. O menor é o *template constante*, em que não há variáveis livres, logo representa apenas uma regra. As 8-uplas representada pela Equação 4 representa um template constante que pode ser associado apenas a regra 30.

$$(0, 0, 0, 1, 1, 1, 1, 0) \tag{4}$$

Já a 8-uplas representada pela Equação 5 representa um template base que está as-

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

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \tag{5}$$

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 $(1, 1, 1, 1, 1, x_0 + x_1, x_1, x_0)$, 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.

3.1 EXPANSÃO DE TEMPLATES

Expansão é o processo no 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 maior detalhes da seguinte maneira:

$$E(T) = R_k \tag{6}$$

A operação de expansão pode ser dividida em dois passo, em que o primeiro consiste em efetuar todas as i substituições de variáveis, sendo que i pertence ao intervalo discreto $[0, k^m - 1]$. 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-arias 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-aria 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)

O segundo passo da operação de expansão é eliminar as tabelas k-aria inválidas. No caso do template T_1 todas as tabelas resultantes eram válidas, 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-aria inválida pois a terceira posição apresenta um estado com um valor fora do interval $[0, k - 1]$, para $k = 2$. A Tabela 2 evidência melhor essa substituição inválida.

Tabela 2: Processo de expansão

i	x_1	x_0	tabela k-aria 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)

Ainda há outra maneira em que templates resultam em substituições inválidas, sendo uma delas através da utilização da notação restrição por conjuntos. Considere o templates $T_3 = (2, 2, 2, 2, 2, 2, 2, 2, x_0 \in \{0, 1\})$ da família de $k = 3$ e $r = 0, 5$. A substituição obtida para $i = 2$ seria inválida pois nesse caso $x_0 = 2$ e $2 \notin \{0, 1\}$. A Tabela 3 evidencia melhor esse processo.

Tabela 3: Processo de expansão

i	x_0	tabela k-aria 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)

A existência de regras inválidas são os responsáveis por templates que representem um conjunto de regras menores que k^m. Essa possibilidade é bastante útil para os templates de regras conservativas e regras confinadas.

O valor de i sempre representa apenas uma substituição possível para as variáveis livres de um templates. Isso ocorre pois i é a representação decimal da conversão k-aria das concatenações dos valores das variáveis livres em ordem decrescente. Exemplificando, considere o templates $T_3 = (2, 2, 2, 2, 2, 2, 2, x_1, x_0 \in \{0, 1\})$ 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 $x_0 = 2, x_1 = 1$.

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

3.2 INTERSECÇÃO DE TEMPLATES

Intersecção é o processo no qual se obtêm um template que represente o conjunto R_k de todas as regras pertencentes aos dois templates definidos para o mesmo espaço recebidos como parâmetro. A operação de intersecção foi descrita por Verardo (2014) e mostrada em maiores detalhes da seguinte maneira:

$$I(T_1, T_2) = T_3 \Leftrightarrow E(T_3) = E(T_1) \cap E(T_2) \tag{7}$$

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

Para melhor compreensão, considere 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 transformado em um sistema de equações como demonstrado na Equação 8.

$$\begin{cases} x_7 = x_7 \\ x_3 = 1 \\ 1 - x_4 = x_5 \\ x_4 = 0 \\ x_3 = x_3 \\ x_2 = x_2 \\ 2 = 2 \\ x_0 = 2 \end{cases} \tag{8}$$

Esse sistema de equações é passado então como argumento para a função Solve, que por sua vez retorna um conjunto solução S, que nesse exemplo é $S = \{x_0 = 2, x_3 = 1, x_4 = 0, x_5 = 1 - x_4, x_6 = 0\}$. O cojunto 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. Neste exemplo, após aplicada as substituições

do conjunto de soluções S obtêm-se como resultado o templates $T_3 = (x_7, 1, 1, 0, 1, x_2, 2, 2)$.

A segunda etapa do algoritmo apenas é aplicada 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 ou 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, considere 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 $T_{r2} = (x_7 \in \{0, 1\}, 1, x_5, 0, x_3, x_2 \in \{1\}, 2, 2)$. A primeira etapa ocorre normalmente, entretanto, quando as substituições do conjunto S forem aplicadas nos templates recebidos, não será mais obtido templates iguais. Nesse caso o conjunto de templates obtidos será $\{(x_7 \in \{0, 1, 2\}, 1, 1, 0, 1, x_2 \in \{1, 2\}, 2, 2), (x_7 \in \{0, 1\}, 1, 1, 0, 1, x_2 \in \{1\}, 2, 2)\}$. Na sequencia o algoritmos 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 Equação 9.

$$\begin{cases} x_7 = 0 \quad \vee \quad x_7 = 1 \quad \vee \quad x_7 = 2 \\ x_7 = 0 \quad \vee \quad x_7 = 1 \\ x_2 = 1 \quad \vee \quad x_2 = 2 \\ x_2 = 1 \end{cases} \tag{9}$$

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 retornado para remover as restrições da variável x_2, transformando-a no valor 1, e restringi a variável x_7 apenas ao conjunto $\{0, 1\}$. O template de intersecção gerado por todo esse processo é representado pela Equação 10.

$$T_{r3} = (x_7 \in \{0, 1\}, x_3, 1 - x_4, x_4, x_3, 1, 2, x_0) \tag{10}$$

Vale frisar 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 sempre preferível pois variáveis a mais acarretam em mais processamento.

No caso de binário, ou seja $k = 2$, a notação de restrição nunca se faz necessária visto que ou a restrição terá apenas um valor factível, sendo preferível que essa variável e sua

restrição sejam substituídas por valor constante, ou a variável poderá assumir qualquer estado, sendo assim por definição uma variável livre.

3.3 COMPLEMENTO DE TEMPLATES

A operação de complemento é responsável por, dado um template, obter um conjunto de templates que represente todas as regras que não pertençam ao template passado como argumento. Está operação pode ser melhor visualizada abaixo:

$$C(T_1) = \bar{T_1} \tag{11}$$

A Figura 6 ilustra essa operação, que consiste em passar um templates T_1 para a função, e receber um conjunto de templates complementares a T_1, aqui representados como $\bar{T_1}$.

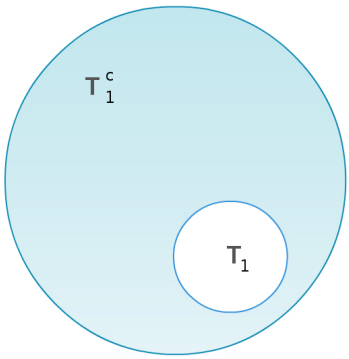

Figura 6: Em branco, T_1 representa uma família de ACs. Em azul, $\bar{T_1}$ representa o complemento dessa família.

O processo que o algoritmo usa para encontrar o conjunto complementar de um template é efetuado através de uma sequência de etapas. A primeira etapa consiste em igualar o template recebido com o template base do mesmo espaço. Com esse processo obtêm-se uma combinações lógicas de equações, remove-se então todas as equações tautológicas, aplica-se a operação de conjugação nas equações, e troca-se o operador lógico $\wedge$ por $\vee$. Por fim esse sistema é então passado como argumento para a função Solve. O resultado da função Solve é um conjuntos com diversos conjuntos de substituições que são aplicados ao template base.

Para melhor visualizar essa primeira etapa, considere o template $T_1 = (x_7, x_6, x_5, 1 - x_1, x_3, x_2, x_1, 0)$ de $k = 2$ e $r = 1$. Esse template será igualado com o template base

gerando o sistema de equações 12, representado abaixo:

$$\begin{cases} x_7 &= & x_7 \\ x_6 &= & x_6 \\ x_5 &= & x_5 \\ x_4 &= & 1 - x_1 \\ x_3 &= & x_3 \\ x_2 &= & x_2 \\ x_1 &= & x_1 \\ x_0 &= & 0 \end{cases} \tag{12}$$

Esse sistema deve ser representado através de combinações lógicas de equações. A Equação 13 é equivalente a Equação 12 e também pode ser resolvido pela função Solve do *Wolfram Language*.

$$x_7 = x_7 \wedge x_6 = x_6 \wedge x_5 = x_5 \wedge x_4 = 1 - x_1 \wedge x_3 = x_3 \wedge x_2 = x_2 \wedge x_1 = x_1 \wedge x_0 = 0 \tag{13}$$

Antes de passar a Equação 13 para a função Solve, o algoritmo elimina todas as equações tautológicas e troca todo operadores lógico $\wedge$ por $\vee$. Por fim é aplicado a operação de conjugação, que no caso binário é efetuar a permutação $\rho = (0 \to 1, 1 \to 0)$. Essa permutação também pode ser feita no caso binário aplicando a função $f(x) = 1-(x)$. A Equação 14 representa a combinação lógica de equações resultante dessas operações.

$$x_4 = 1 - (1 - x_1) \vee x_0 = 1 - 0 \tag{14}$$

A Equação 14 é então passada como argumento para a função Solve, que retorna o conjunto solução S tal que $S = \{\{x4 \to x_1\}, \{x0 \to 1\}\}$. Perceba que S apresenta mais de um conjunto de substituições, então utilizaremos ambos para gerar realizar as substituições no template base, obtendo assim o seguinte conjunto de templates $\{(x_7, x_6, x_5, x_1, x_3, x_2, x_1, x0), (x_7, x_6, x_5, x4, x_3, x_2, x_1, 1)\}$. A união desses dois templates representa todas as regras não representadas pelo template T_1.

No exemplo dado com o template T_1 apenas a primeira etapa da operação de complemento é necessária, mas nem sempre é assim. Alguns templates apresentam *templates de exceção*. Os templates que no processo de expansão apresentam alguma substituição inválida são os templates que possuem *templates de exceção*. Na segunda etapa da

operação de complemento se verifica se o template possui combinações de substituições que o levem a gerar regras inválidas. Caso exista, gera-se os templates de exceção do template passado como argumento e adiciona-o ao conjunto de templates complementares.

Exemplificando, considere o template $T_2 = (x_7, x_6, x_5, 1 - x_1 - x_2, x_3, x_2, x_1, 0)$ para $k = 2$. A primeira etapa da operação de complemento ocorre normalmente e encontra os templates complementares $\{(x_7, x_6, x_5, x_4, x_3, x_2, x_1, 1), (x_7, x_6, x_5, x_1 + x_2, x_3, x_2, x_1, x_0)\}$. Todavia é trivial perceber que qualquer expansão do template T_2 que tenha o conjunto de substituições $\{x_1 = 1, x_2 = 1\}$ fará com que a quinta posição do template apresente o valor 2 que não pertence ao intervalo $[0, k-1]$. Logo, todos os templates que apresentem $\{x_1 = 1, x_2 = 1\}$ são complementares ao template T_2. O processo de gerar o template de exceção consiste partindo de templates base aplicar cada um dos conjuntos de substituições que levam à regras inválidas. No caso de T_2, será obtido o template de exceção $T_{excecao1} = (x_7, x_6, x_5, x_4, x_3, 1, 1, x_0)$. A operação de complemento então adiciona na lista de templates complementares os templates de exceção, finalizando assim o processo e obtendo o seguinte conjunto de templates complementares:

$$\{(x_7, x_6, x_5, x_4, x_3, x_2, x_1, 1), (x_7, x_6, x_5, x_1 + x_2, x_3, x_2, x_1, x_0), (x_7, x_6, x_5, x_4, x_3, 1, 1, x_0)\} \tag{15}$$

É importante dizer que que a implementação do algoritmo que executa a operação de complemento ainda não permite trabalhar com $k \neq 2$. Outra questão relevante que deve ser devidamente enfatizada é que apesar de nos exemplos com T_1 e T_2 não se ter apresentado mais de um templates de exceção, isso é possível e comum.

4 PROPRIEDADES ESTÁTICAS DOS AUTÔMATOS CELULARES

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

Existem diversas propriedade estáticas conhecidas, esta seção descreverá algumas propriedades estáticas já implementadas na biblioteca *CATemplates* (VERARDO; de OLIVEIRA, 2015).

4.1 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 devem se alterar durante a evolução espaço temporal, independente da configuração inicial passada.

De acordo com (BOCCARA; FUKś, 2002), para um AC ser conservativo cada uma de suas regras locais f de vizinhança $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ deve respeitar as condições descritas na Equação 16.

$$\begin{aligned} f(\alpha_0, \alpha_1, \ldots, \alpha_{n-1}) = \alpha_0 + (\sum_{i=0}^{n-2} & f(0_0, 0_1, \ldots, 0_i, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}) \\ & - f(0_0, 0_1, \ldots, 0_i, \alpha_0, \alpha_1, \ldots, \alpha_{n-i-1})) \end{aligned} \tag{16}$$

Para exemplificar, considere a regra 204 do espaço elementar. Por meio da condição mostrada na Equação 16, será provado que essa regra é conservativa, já que satisfaz a condição. A Equação 17 representa a tabela de transição da regra 204.

$$\begin{aligned} &(((1,1,1),1),((1,1,0),1),((1,0,1),0),((1,0,0),0), \\ &((0,1,1),1),((0,1,0),1),((0,0,1),0),((0,0,0),0)) \end{aligned} \tag{17}$$

Como demonstra (VERARDO, 2014), a aplicação das condições descritas na Equação 16 nas tabelas de transições de ACs do espaço elementar, sempre resultará no sistema

descrito pela Equação 18, sendo que essa Equação simplificada pode ser representada pela Equação 19.

$$\begin{cases} f(0,0,0) = 0 + (f(0,0,0) - f(0,0,0)) + (f(0,0,0) - f(0,0,0)) \\ f(0,0,1) = 0 + (f(0,0,1) - f(0,0,0)) + (f(0,0,0) - f(0,0,0)) \\ f(0,1,0) = 0 + (f(0,1,0) - f(0,0,1)) + (f(0,0,1) - f(0,0,0)) \\ f(0,1,1) = 0 + (f(0,1,1) - f(0,0,1)) + (f(0,0,1) - f(0,0,0)) \\ f(1,0,0) = 1 + (f(0,0,0) - f(0,1,0)) + (f(0,0,0) - f(0,0,1)) \\ f(1,0,1) = 1 + (f(0,0,1) - f(0,1,0)) + (f(0,0,0) - f(0,0,1)) \\ f(1,1,0) = 1 + (f(0,1,0) - f(0,1,1)) + (f(0,0,1) - f(0,0,1)) \\ f(1,1,1) = 1 + (f(0,1,1) - f(0,1,1)) + (f(0,0,1) - f(0,0,1)) \end{cases} \tag{18}$$

$$\begin{cases} f(0,0,0) &= 0 \\ f(0,0,1) &= f(0,0,1) \\ f(0,1,0) &= f(0,1,0) \\ f(0,1,1) &= f(0,1,1) \\ f(1,0,0) &= 1 - f(0,0,1) - f(0,1,0) \\ f(1,0,1) &= 1 - f(0,1,0) \\ f(1,1,0) &= 1 + (f(0,1,0) - f(0,1,1)) \\ f(1,1,1) &= 1 \end{cases} \tag{19}$$

Excluindo-se as condições tautológicas do sistema e atribuindo os valores das funções f conforme a tabela de transição da regra 204, é obtido o sistema descrito pela Equação 20, que claramente respeita todas as condições do sistema, provando assim que a regra 204 é conservativa.

$$\begin{cases} 0 &= 0 \\ 0 &= 1 - 0 - 1 \\ 0 &= 1 - 1 \\ 1 &= 1 + (1 - 1) \\ 1 &= 1 \end{cases} \tag{20}$$

A implementação desse algoritmo, apresentada em (de OLIVEIRA; VERARDO, 2014), funciona na seguinte sequência de passos: o algoritmo recebe as variáveis k e r definindo assim a família das regras que serão geradas, depois são criadas todas as vizinhanças do espaço, excluídas as vizinhanças geram tautologias e então aplicadas as condições de Boc-

cara e Fukś (2002). Basicamente, vizinhanças que geram regras tautológicas, de acordo com (NADA, a), são as que começam com 0 mas não sejam compostas apenas por 0.

Exemplificando, considere um espaço com $k = 2$ e $r = 1$. Primeiramente o algoritmo obterá o conjunto das vizinhanças que não geram regras tautológicas, portanto será obtido o conjunto $\{(1,1,1),(1,1,0),(1,0,1),(1,0,0),(0,0,0)\}$. Então é criado o sistema de equações 21, baseado nas condições de Boccara e Fukś (2002).

$$\begin{cases} x_0 &= 0 \\ x_4 &= 1 + 2x_0 - x_1 - x_2 \\ x_5 &= 1 + x_0 - x_2 \\ x_6 &= 1 + x_2 - x_3 \\ x_7 &= 1 \end{cases} \tag{21}$$

Por fim o algoritmo utiliza a função Solve do *Wofram Mathematica* para simplificar o sistema, e utiliza o conjunto solução retornado pela função Solve como regras de substituições, e as aplica no template base, gerando assim o template das regras conservativas. O template gerado pode ser observado na Equação 22, e sua expansão nos gera as cinco regras conservativas do espaço elementar após eliminadas as regras inválidas.

$$(1, x_2 - x_3 + 1, 1 - x_2, -x_1 - x_2 + 1, x_3, x_2, x_1, 0) \tag{22}$$

O processo de gerar regras conservativas de paridade é bem parecido, porém as condições de Boccara e Fukś (2002) são ligeiramente modificadas em relação a Equação 16, de forma que cada uma das funções locais deve respeitar agora as condições da Equação 23.

$$\begin{aligned} f(\alpha_0, \alpha_1, \ldots, \alpha_{n-1}) \equiv \alpha_0 + (\sum_{i=0}^{n-2} & f(0_0, 0_1, \ldots, 0_i, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}) \\ & - f(0_0, 0_1, \ldots, 0_i, \alpha_0, \alpha_1, \ldots, \alpha_{n-i-1})) \; (mod 2) \end{aligned} \tag{23}$$

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

4.2 CONFINAMENTO

Autômatos celulares confinados, ou *captive* em Inglês, são uma classe de AC, que se baseia em uma caracterização de sua funções locais sem dotar o estado definido com qualquer estrutura externa à vizinhança (THEYSSIER, 2004).

Essa propriedade estática foi proposta por Theyssier (2004), que formalmente define que dado 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 Equação 24.

$$f((\alpha_0, \ldots, \alpha_{2r})) = \beta, \beta \in \{\alpha_0, \ldots, \alpha_{2r}\} \tag{24}$$

Essa propriedade pode ser facilmente representada através de templates, bastando restringir as variáveis à um conjunto de valores presentes na vizinhança correspondente. Já há um algoritmo que gera as regras confinadas implementado no biblioteca *open source CATemplates*. Esse algoritmo tem um funcionamento simples: ele recebe como parâmetro os argumentos k e r. Gera então as vizinhanças do espaço e verifica em cada uma das vizinhanças os estados que elas tem. Então, caso a vizinhança tenha todos os estados do intervalo $[0, k-1]$, no template essa posição terá uma variável livre. Caso a vizinhança tenha apenas um estado, a posição correspondente no template assume um valor fixo. Por fim, caso a vizinhança apresente mais de um estado, mas não todos, a posição correspondente do template apresentará uma variável restritas mediante expressões $x_i \in C$.

No caso binário é trivial perceber que qualquer AC binário que tenha as funções locais $f((0_0, 0_1, \ldots, 0_{2r})) = 0$ e $f((1_0, 1{,}1 \ldots, 1_{2r})) = 1$ é caracterizado como um AC confinado. A Equação 25 representa o template de todas as regras confinadas do espaço elementar. Já as equações 26 representam a família de $k = 2$ e $r = 0, 5$.

$$(1, x_6, x_5, x_4, x_3, x_2, x_1, 0) \tag{25}$$

$$(1, x_2, x_1, 0) \tag{26}$$

Já a Equação 27 representa a família dos autômatos celulares confinados de $r = 1$ e

três estados.

$$(2, x_{25} \in \{1,2\}, x_{24} \in \{0,2\}, x_{23} \in \{1,2\}, x_{22} \in \{1,2\}, x_{21}, x_{20} \in \{0,2\}, x_{19}, x_{18} \in \{0,2\},$$
$$x_{17} \in \{1,2\}, x_{16} \in \{1,2\}, x_{15}, x_{14} \in \{1,2\}, 1, x_{12} \in \{0,1\}, x_{11}, x_{10} \in \{0,1\}, x_9 \in \{0,1\},$$
$$x_8 \in \{0,2\}, x_7, x_6 \in \{0,2\}, x_5, x_4 \in \{0,1\}, x_3 \in \{0,1\}, x_2 \in \{0,2\}, x_1 \in \{0,1\}, 0) \tag{27}$$

5 APLICAÇÃO

A representação de famílias de autômatos celulares através de templates, assim como a possibilidade de efetuar operações de intersecção e complemento de templates gera diversas possibilidades de utilização para problemas já bem conhecidos da área. Um exemplo de problema que pode se beneficiar dos templates é o problema da paridade.

O problema da paridade refere-se à capacidade de um AC binário, unidimensional, com condição de contorno periódica, apresentar uma evolução temporal da seguinte forma: se uma configuração inicial contiver um número ímpar de células com o valor 1, o AC deve convergir para que todas as células estejam no estado 1; caso contrário, ele deve convergir para todas as células no estado 0. Pode-se dizer que as regras que solucionam o problema de paridade em ACs são *perfeitas* se elas resolverem o problema em qualquer configuração inicial arbitrária, em reticulados de tamanho ímpar.

A Figura 7 ilustra o desenvolvimento espaço temporal de uma regra que usualmente resolve o problema de paridade. Nessa imagem o desenvolvimento temporal à esquerda contém em sua configuração inicial um número par de estados igual a 1, já na evolução temporal ilustrada à direita há um número ímpar de estados igual a 1.

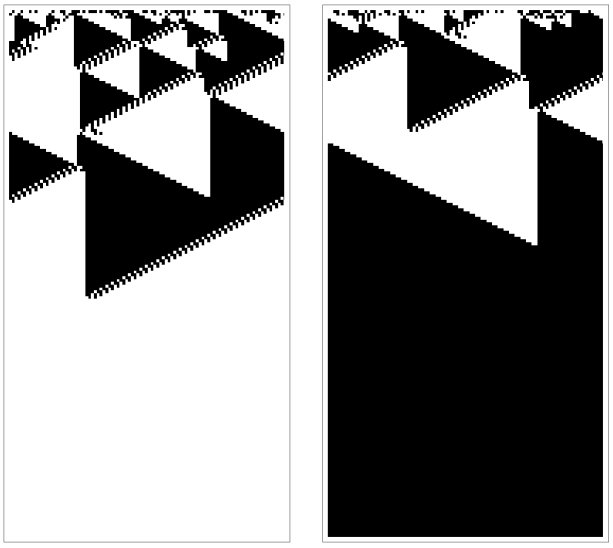

Figura 7: Desenvolvimento espaço temporal de um AC que usualmente resolve o problema de paridade.

O estudo do problema de paridade é interessante pois ajuda a compreender outro problemas ainda sem solução em ACs, que é que a despeito da habilidade de realizar computações, pouco se sabe sobre como se deve projetar transições de estado locais que

levem a um comportamento global pré determinado. Entretanto, entender o impacto das interações locais nas soluções globais influi em diversos sistemas de computação emergente. Entender a influência que o tamanho da vizinhança dos autômatos celulares apresenta na computabilidade pode apresentar consequências úteis para o projeto de ACs e 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 perfeitas para ACs elementares e de raio 2, todavia já foi descoberta uma regra perfeita que soluciona o problema de paridade para raio 4. Em relação aos ACs de raio 3, ainda não foi encontradas solução perfeita e há evidências empíricas desfavoráveis a uma solução para esse raio (BETEL; de OLIVEIRA; FLOCCHINI, 2013).

Esse estudos de BETEL, de OLIVEIRA e FLOCCHINI (2013), buscando definir se o problema de paridade em ACs de raio 2 apresenta alguma soluções perfeita, encontraram de forma analítica como as transações de estado de supostas regras que resolvessem o problema de paridade deveriam ser. Com isso, utilizando grafos de *De Bruijn* foram definidas quais variáveis deveriam ser estáticas, quais deveriam ser livres e quais deveriam apresentar interdependência. Com essa definições foram definidas duas famílias de ACs. Os grafos ilustrados na Figura 8 e na Figura 9 são os grafos desenvolvidos por BETEL, de OLIVEIRA e FLOCCHINI (2013).

Ambos os grafos apresentam 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. No segundo grafo a única diferença são nas transições de estados estáticas. Como se pode ver na Figura 9.

Ao utilizar os grafos de *De Bruijn* fixando algumas transições de estado, a família de ACs em que se procurava as regras que solucionavam o problema de paridade, antes composta por 2^{32} regras, foi restringida para apenas 64 regras, que puderam ser estudadas em mais detalhes até que se falhassem. Entretanto a representação desse espaço de 64 regras poderiam ser perfeitamente representados através de *templates*. O template 28 representa o mesmo espaço que a Figura 8, já o template 29 é equivale a Figura 9.

$$(0, 1, 1, 1, 1, x_{26}, 0, 1, 1, 1, 1 - x_{10}, x_{20,}0, 0, 1, 0, 1, 0, 1, 0, x_{11}, x_{10}, 0, 0, 1, 0, x_5, 0, 1, 0, 0, 1) \tag{28}$$

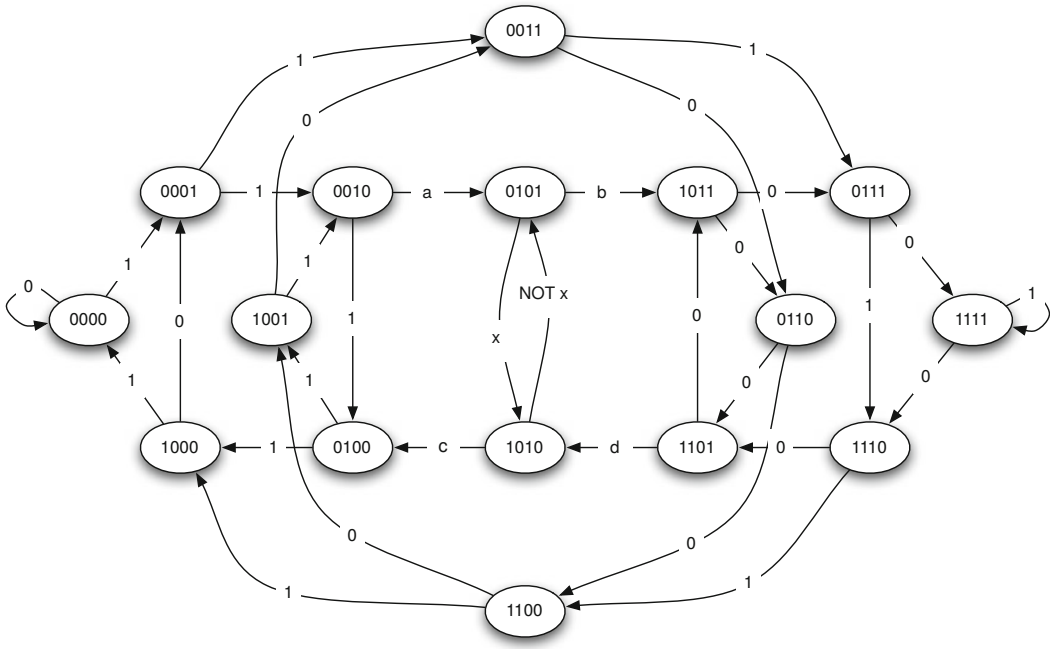

Figura 8: Grafo de De Bruijn representando regras que possivelmente solucionem o problema de paridade.

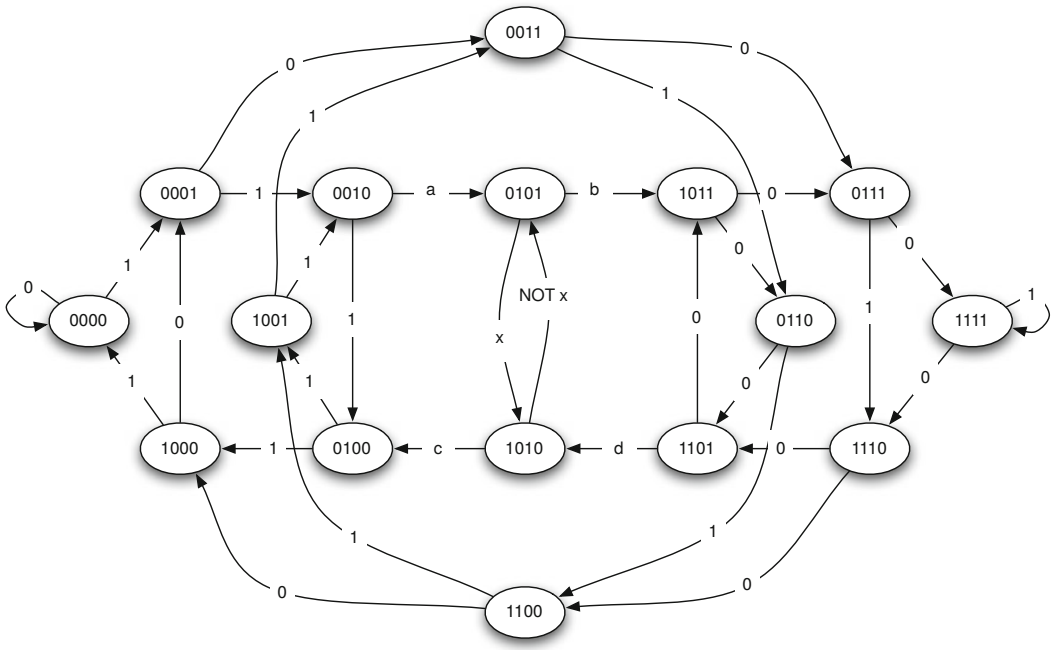

Figura 9: Outro grafo de De Bruijn representando regras que possivelmente solucionem o problema de paridade.

$$(0,1,1,0,1,x_{26},1,0,1,1,1-x_{10},x_{20,}0,1,1,0,1,0,1,1,x_{11},x_{10},0,0,1,0,x_5,0,0,0,0,1) \tag{29}$$

Ainda no contexto do problema de paridade, o conceito de templates pode ser bastante útil. Conforme dito anteriormente, ainda não se sabe se existem regras de raio 3 que solucionem o problema de paridade. Mas os templates podem ser uma forma interessante de restringir o conjunto de regras na busca dessa solução.

Os regras dos ACs que solucionam o problema de paridade tem algumas propriedades estáticas podem ser trivialmente percebidas. Um AC que resolva o problema de paridade sempre será contido, tendo em vista no problema de paridade as vizinhanças homogêneas não devem levar à transições de estado ativas, e está é a única restrição de variável dos templates contidos para AC binários. Vale frisar que o espaço das regras contidas de raio 3 ainda é um espaço muito grande, entretanto essa não é a única propriedade estática que um AC que resolva o problema de paridade deve conter.

Para que um AC resolva problema de paridade também é necessário que ele não seja conservativo, visto que se a soma dos estados do AC não mudar, ele nunca convergirá como propõem o problema. Por fim, é espera-se que um AC que resolva o problema de paridade seja conservativo de paridade, afinal o caso o AC troque de paridade ele tenderá a convergir para a solução errada.

Como conseguimos obter os templates para todas essa propriedades, é possível utilizar as operações de complemento e intersecção para restringir o espaço de busca para a solução desse problema. Basta que se passe como argumento os templates das propriedades estáticas que espera-se que o AC tenha para a operação de intersecção, ou seja, efetua-se a intersecção dos templates de confinamento e conservabilidade de paridade. E, posteriormente, efetue-se a intersecção do template obtido com cada um dos templates resultantes da operação de complemento do template das regras não consideradas uteis para a resolução do problema, ou seja, o complemento dos templates das regras conservativas de estado.

Formalmente essas operações de conjuntos entre os templates pode ser representado através da Equação 30, sendo que $T_{confinado}$ representa o templates das regras confinadas, $T_{conservaparidade}$ representa o templates das regras que conservam a paridade, $\bar{T}_{conservaestados}$ representa o conjunto de templates complementares ao templates das regras conservativas e $T_{paridade}$ representa um conjunto templates que restringem um pouco mais as regras com

possibilidades de solucionar o problema de paridade.

$$T_{paridade} = (T_{conservaparidade} \cap T_{confinado}) \cap \bar{T}_{conservaestados} \tag{30}$$

6 CONSIDERAÇÕES FINAIS

No presente trabalho é descrito os templates de autômatos celulares, proposta introduzida por de OLIVEIRA e VERARDO (2014), que por meio de uma generalização de tabelas de transição k-árias pode representar conjuntos de ACs.

O conceito de templates é importante devido sua capacidade de representar conjuntos de ACs com determinada propriedade dinâmica. Essa propriedade faz com que não seja necessário buscar por todo um espaço de ACs, que devido ao rápido crescimento das famílias dos ACs conforme se mudam seu parâmetros, impossibilitaria a busca através de força bruta.

Por conta dessa dessas capacidade de representação de ACs com determinada propriedades, foi exposto aqui também algumas propriedades estáticas e os algoritmos geradores dos templates que as representam. Esses algoritmos esses já estavam implementados na biblioteca *open source CATemplates* VERARDO e de OLIVEIRA (2015). Além disso foi explicada algumas operações do *CATemplates* que podem ser aplicadas aos templates: a expansão e a intersecção. Essas operações, desenvolvidas por Verardo (2014), foram mostradas novamente aqui para que fosse possível explicitar sua importância e relevância para a solução de problemas típicas do ACs, como o problema de paridade.

Também foi introduzido nesse trabalho a operação de complemento de templates. Essa operação permite que dado um templates, se encontre um conjunto de templates que represente todas as regras que não pertençam ao template passado como argumento. Essa operação ainda não aceita templates não binários, e um possível trabalho futuro é a implementação desse algoritmo generalizado para qualquer valor de k. A operação de complemento, já disponível na biblioteca *CATemplates*, é outro exemplo de operação que pode ser utilizada para restringir o conjunto de regras a serem avaliadas na busca pela solução do problema de paridade, por exemplo.

Para que fosse possível a implementação da operação de complemento, foi introduzido aqui a operação que, dado um template, gera *templates de exceção*. *Templates de exceção* são gerados apartir de templates bases substituindo-se algumas variáveis pelas substituições que geram regras inválidas no template original. Os algoritmo que geram templates de exceção são essenciais para a operação de complemento.

Também é apresentado aqui alguns processos, utilizando templates, que podem auxiliar na restrição do espaço de busca para a solução do problema de paridade

Como trabalhos futuros se estuda a implementação de novos algoritmos geradores de templates outras propriedades estáticas, tais como a propriedade *left permutive*, assim como a implementação de novas operações de templates baseadas na operações de conjuntos, como a operação de união.

7 PLANO DE TRABALHO

Tabela 4: Cronograma de desenvolvimento do projeto

	2014	2015												2016
Atividades	Dez	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez	Jan
Pesquisa bibliográfica	•													
Revisão de literatura	•	•												
Coleta e seleção de dados			•											
Análise de dados			•	•	•									
Programação e testes				•	•	•								
Primeira entrega						•	•							
Revisão e correção							•	•	•					
Submissão de artigos								•		•				
Apresentação da qualificação									•					
Ajustes e melhorias										•	•	•	•	
Apresentação do trabalho final														•

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 - Special issue on cellular automata*, v. 25, p. 1–13, 2002.

de OLIVEIRA, P. P. B.; VERARDO, M. Representing families of cellular automata rules. *Journal of Cellular Automata*, v. 16, 2014.

GARDNER, M. Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, p. 120–123, out. 1970.

LI, W.; PACKARD, N. The structure of the elementary cellular automata rule space. *Complex Systems*, v. 4, n. 3, p. 281–297, 1990.

NEUMANN, J. V. *Theory of Self-Reproducing Automata*. Champaign, IL, USA: University of Illinois Press, 1966.

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. Talvez trocar por artigo.

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

WEISSTEIN, E. W. Cellular automaton. From MathWorld–A Wolfram Web Resource, 2015. Disponível em: <http://mathworld.wolfram.com/CellularAutomaton.html>. Acesso em: 19 mai. 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*, American Physeica Soc, One Physics Ellipse, College PK, MD 20740-3844 USA, 55, n. 3, p. 601–644, 1983. ISSN 0034-6861.

WOLFRAM, S. *Cellular Automata and Complexity: Collected Papers*. Addison-Wesley Publishing Company, 1994. (1-2150-A; Louisiana Barrier Island). ISBN 9780201626643. Disponível em: <http://books.google.co.in/books?id=8u1EDgvtVhEC>.