

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

William Barbosa dos Santos

Diferença entre templates de autômatos celulares de  
ordem  $k$ -ária

Projeto de Pesquisa apresentado ao Programa de Pós-Graduação em Engenharia Elétrica e Computação da Universidade Presbiteriana Mackenzie como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica e Computação.

Orientador: Pedro Paulo Balbi de Oliveira

São Paulo  
2017

## RESUMO

Autômatos celulares são sistemas dinâmicos/computacionais discretos usados em diversas aplicações na ciência em geral (como em propagação de doenças, dispersão de boatos, espalhamento de fogo em florestas, etc) e na computação (criptografia, geração de padrões, computação distribuída, etc). Encontra-se formalizada na literatura a noção de *template* de regras, que constitui a representação de um determinado conjunto de regras de autômatos celulares; associado ao conceito, incluem-se as operações de expansão dos mesmos, bem como a intersecção e diferença entre eles, esta última apenas para autômatos celulares binários. Essas operações estão atualmente implementadas na biblioteca *CA-Templates* que utiliza-se do *software Mathematica*. Os *templates* são importantes por permitirem uma notação capaz de representar um grupo de regras de autômatos celulares que, por exemplo, representem uma propriedade específica, como a conservabilidade. Essa característica torna possível condensar um espaço de regras para pesquisa, evitando que se tenha de varrer o espaço de regras, seja exaustivamente, seja por meio de algum algoritmo de busca. As operações com *templates* permitem ainda mais versatilidade, tornando possível a intersecção ou mesmo disjunção de *templates*. Para essa última, também chamada de operação de diferença, existe hoje apenas a implementação para autômatos celulares binários. O presente trabalho visa expandir a operação de diferença para funcionamento em autômatos celulares não binários, isto é, aqueles em que suas células componentes possuem mais de dois estados possíveis. Com a expansão do funcionamento da operação de diferença volta-se à ideia original dos *templates* de serem uma representação de um determinado conjunto de regras, mas mantendo comportamento e operações que podem funcionar de modo genérico, isto é, para qualquer quantidade de estados e qualquer tamanho de vizinhança.

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

## ABSTRACT

Cellular automata are discrete dynamical systems used on various applications in the science in general (such as disease propagation, dispersion of rumors, fire dispersion in forests, etc) and in the computing field (encryption, pattern generation, distributed computing, etc). The notion of templates of cellular automata can be found in the literature, they are the representation of a certain set of cellular automata rules; alongside with that concept it's included the expansion operation of templates, and also the intersection and exception operations between templates, the last being just for binary cellular automata. The operators are implemented in the CATemplates framework using the Mathematica software. The templates are important because they are a formal notation capable of representing a set of cellular automata rules, for example the rules that respect a specific property (like a conservative cellular automata). This specific characteristic makes it possible to condense the rules to be analyzed in a research instead of going through the whole set of rules, either by brute force or using some search algorithm. The templates operations bring versatility, allowing intersection or even disjunction of templates. The disjunction operation, also known as exception operation, is currently implemented only for binary cellular automata. Here we intend to implement the exception operator for non-binary automatas, that is to say, those that their cells might have more than two possible states. That will make a return to the original concept of a template, which is to represent a set of rules without being bound to a specific space, that will generalize the operators in order to perform properly regardless of the total possible states or the neighborhood size.

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

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>1</b>
<b>2</b>	<b>METODOLOGIA</b>	<b>3</b>
<b>3</b>	<b>AUTÔMATOS CELULARES</b>	<b>5</b>
3.1	CONSERVABILIDADE DE ESTADOS . . . . .	7
3.2	TOTALIDADE E SEMI-TOTALIDADE . . . . .	8
3.3	BALANCEAMENTO . . . . .	8
<b>4</b>	<b>TEMPLATES</b>	<b>10</b>
4.1	EXPANSÃO . . . . .	12
4.2	INTERSECÇÃO . . . . .	14
4.3	TEMPLATE PARA REGRAS BALANCEADAS . . . . .	15
<b>5</b>	<b>DIFERENÇA ENTRE TEMPLATES BINÁRIOS</b>	<b>17</b>
<b>6</b>	<b>DIFERENÇA ENTRE TEMPLATES <math>K</math>-ÁRIOS</b>	<b>22</b>
<b>7</b>	<b>CONSIDERAÇÕES FINAIS E CRONOGRAMA</b>	<b>28</b>
	<b>REFERÊNCIAS BIBLIOGRÁFICAS</b>	<b>30</b>

# 1 INTRODUÇÃO

Autômatos celulares (ACs) são sistemas computacionais discretos onde a unidade mais básica de processamento (a célula) age apenas localmente, atuando de acordo com as células vizinhas. No entanto, mesmo com uma regra simples de ação de cada célula, o AC pode ser capaz de apresentar características globais arbitrariamente complexas (WOLFRAM, 2002).

Essa dicotomia entre simplicidade e complexidade no que tange o comportamento local e global torna os ACs uma opção na pesquisa de problemas que contenham justamente essa característica, como é o caso, por exemplo, de se estudar a dispersão espacial de doenças.

Todavia, uma das dificuldades em se utilizar ACs na resolução de problemas práticos reside em encontrar uma regra de comportamento local que atenda e gere o comportamento global desejado. Para tanto, sempre que possível tende-se a limitar as regras candidatas a uma classe específica. Filtrando apenas regras que atendam à classe desejada diminui-se o espaço de busca total, possibilitando uma melhor análise. DE OLIVEIRA e VERARDO (2014) estudaram uma forma de realizar esse filtro, implementando um mecanismo computacional para automatizar o processo, o qual foi denominado *template*. Utilizando a notação de *template* é possível definir um grupo de regras de ACs que respeitem as regras nele definidas, tal qual realizar determinadas operações sobre os *templates* em si. Duas das operações possíveis seriam a operação de intersecção, que consiste em criar um terceiro *template* que acate as definições de dois *templates* em particular (assim definindo a intersecção dos grupos de regras desses dois *templates*), e a operação de expansão, que é responsável por transformar o *template* no conjunto de todas as regras que o mesmo representa.

Ainda sim, por vezes se faz necessário uma pesquisa de regras que não atendam a determinado comportamento. A inversão lógica do filtro, ou ainda, o complemento do grupo de regras que atendam a determinada característica, apresenta uma complexidade de resolução diferente do que foi estudado por DE OLIVEIRA e VERARDO (2014). Também chamada de operação de exceção, o resultado desejado é um *template* que represente todas as regras de um *template*  $T_1$ , exceto as que também estão contidas em um *tem-*

*plate*  $T_2$ . Essa operação foi analisada por SOARES (2016), tendo sido apresentado uma solução para os *templates* binários. Todavia uma solução para *templates* de ordem  $k$ -ária encontra-se como um problema em aberto. A generalização da operação se faz necessária para retirar a sua limitação de uso somente a cenários onde possui-se ACs binários.

A utilização do algoritmo de operações com *templates* possui considerável utilidade em pesquisas com autômatos celulares, podendo ser utilizado em epidemiologia, setor financeiro, simulações físicas, simulações de comportamento social, etc, pois nesses casos raramente o espaço elementar dos autômatos celulares é suficiente para pesquisas mais complexas na área, e ACs podem chegar facilmente a bilhões de regras possíveis em um dado espaço. Com o objetivo de se encontrar uma regra (ou grupo de regras) que atendam a um problema específico, a necessidade de cada vez mais sofisticação na busca vem à tona. Com as operações de *templates* pesquisadas por DE OLIVEIRA e VERARDO (2014) e SOARES (2016), e aqui estendidas, os algoritmos de buscas podem realizar um trabalho mais preciso em um espaço sensivelmente menor que o espaço completo.

A Seção 3 desse trabalho apresenta mais detalhes sobre autômatos celulares e algumas de suas propriedades estáticas. A Seção 4 descreve a notação utilizada para representar um conjunto de regras de ACs (denominada *template*) e algumas das operações possíveis de serem realizadas com os *templates*. A operação de exceção sobre os *templates* é demonstrada inicialmente na Seção 5, porém com a limitação de funcionamento apenas para ACs binários ( $k = 2$ ). Na Seção 2 é apresentado os objetivos práticos do trabalho e a metodologia seguida para alcançá-los. A Seção 6 exibe a operação de exceção de *templates* generalizada para ACs de  $k > 1$ . Na Seção 7 discutem-se os resultados obtidos.

## 2 METODOLOGIA

Devido aos recursos já existentes no software de desenvolvimento *Wolfram Mathematica* (WOLFRAM RESEARCH, 2017), como a utilização de símbolos para metaprogramação e inferência de fórmulas, o mesmo foi utilizado para a implementação inicial da biblioteca *CATemplates* (VERARDO; DE OLIVEIRA, 2017). A mesma biblioteca teve funcionalidades expandidas no trabalho de SOARES (2016) sem alteração da tecnologia utilizada, e portanto segue como escolha para prosseguimento dos estudos dos *templates* de autômatos celulares e suas operações.

Neste trabalho estuda-se a implementação de negação de *templates* (de ordem  $k$ -ária) por duas vertentes diferentes. Uma delas utilizando múltiplos *templates* que, quando expandidos, resultam no espaço de regras que seria resultante da operação de negação. Isto é, ao se realizar a negação de um *template* o algoritmo é responsável por gerar internamente os *templates* que, juntos, atendam positivamente ao espaço desejado.

A segunda possibilidade de solução ao problema é a implementação de um obstrutor durante o processo de expansão. Para além das fórmulas matemáticas que hoje já definem um *template* (VERARDO, 2014) pode-se utilizar também uma equação lógica que o *template* deve contemplar para se tornar válido.

Após a implementação dos dois métodos se faz necessário também a implementação de testes unitários para validar a exatidão dos resultados obtidos. Para tal, as regras geradas pela expansão do *template* resultante da operação de negação devem ser comparadas com regras geradas através de força bruta com resultados conhecidos. Essas comparações devem ser restringidas a  $k = 3$  devido ao aumento exponencial no número total de regras no espaço ao se elevar os parâmetros de um AC.

Com os métodos validados com sucesso a performance dos mesmos deve ser comparada utilizando um cronômetro iniciado via algoritmo diretamente antes da expansão, e interrompido logo após a expansão ser finalizada. Tal teste deve ser executado cinco vezes para diferentes valores de  $k$  e  $r$  (mas limitando-os a  $k \leq 3$  e  $r \leq 3$ ).

A possibilidade de paralelização do algoritmo de expansão é o próximo passo do trabalho. Dividindo-se o conjunto de valores das variáveis livres de um *template* pode-

se permitir que diferentes linhas de processamento realizem o trabalho de expansão simultaneamente. Com a paralelização implementada com sucesso o teste de performance pode ser executado novamente para comparação dos resultados e avaliação do ganho obtido (se algum).



### 3 AUTÔMATOS CELULARES

O que é um AC Autômatos celulares são sistemas dinâmicos, discretos quanto ao tempo, espaço e quantidade de variáveis possíveis. Um AC é composto de células, com cada célula podendo estar em um estado. O comportamento do AC ocorre através de uma linha do tempo, onde em cada passo as células podem ou não alterar seu estado. As células interagem entre si, e de acordo com o resultado dessa interação é que seu estado se altera ou não. Um célula é influenciada pelas células na sua vizinhança, e sua interação com elas define seu estado durante as iterações do AC. A computação de um AC típico ocorre de maneira local. Isto é, uma célula interage com suas vizinhas, mas não há a computação global. Em outras palavras, cada célula possui um limite definido de conhecimento dos dados do AC, não sendo influenciada diretamente pelos dados do AC completo.

Para que serve Devido à natureza de computação local os Autômatos celulares são comumente usados em simulações de dispersão, além da área teórica da computação, como objeto de estudo na solução de problemas. À título de exemplo há estudos de utilização de AC's para dispersão de gases, fluxo de trânsito, criptografia, computação universal...

Explicação de Células Computacionalmente cada célula de um AC é um dado. De modo que pode representar virtualmente qualquer coisa, podemos usar os números um e zero para representar os estados vivo e morto por exemplo. Ao se criar um AC se faz necessário definir qual a gama de estados possíveis para a célula, de modo a criar-se as regras de interação posteriormente. Para computacionar o AC é típico a utilização de números para representar os estados da célula, podendo estes números serem reais ou inteiros. A definição clássica utiliza-se de números inteiros apenas, que nesse caso devem variar de 0 à  $k - 1$  ( $k$  sendo o número de estados possíveis para a célula deste AC).

Explicação de vizinhança A vizinhança de uma célula é composta por todas as células que influenciam diretamente em seu estado. Em cada passo da linha do tempo do AC, são os estados das células da vizinhança que definirão o estado da célula. A vizinhança da célula é definida pelo raio ( $r$ ). Imaginando-se as células de um AC dispostas em um reticulado, a vizinhança de uma célula qualquer é composta pelas que estão à  $r$  células de distância. Para além disso, a vizinhança também pode ser irregular. Para um AC unidimensional por exemplo, pode-se ter  $r$  sendo múltiplo de 0.5. Nesse caso a vizinhança

passa a ser irregular pois a vizinhança será maior em uma das direções do reticulados. Por exemplo: para  $r = 2,5$  a vizinhança pode ser 2 células à esquerda e 1 à direita.

**Explicação de periodicidade** Como as células são dispostas em um reticulado para definição de suas vizinhanças, as células na borda do reticulado acabam ficando com a vizinhança incompleta. Para esses casos pode-se definir algumas condições de contorno, como o contorno periódico, onde a primeira célula do reticulado fará vizinhança com a última (como se unissemos as duas pontas do reticulado). Outra condição de contorno utilizada é com valores fixos, onde as células das pontas fazem vizinhança com células fixas (que jamais alteram seus estados).

**Explicação de Regras** O que define a interação das células é a regra do AC. Na definição clássica o AC possui apenas uma regra. A regra do AC é o conjunto de instruções que define como uma célula deve se portar durante as iterações do AC de acordo com sua vizinhança. A regra pode descrever explicitamente que, para determinados valores das células da vizinhança, a célula deve assumir um estado  $X$ . Ou a regra pode definir uma fórmula matemática ou lógica onde os valores das células vizinhas são as entradas da função, e o resultado é o estado que a célula deve assumir.

Representação de Regras Gráfico Binário

Aplicação de Regras na evolução de um AC

Família de Regras

Cálculo do tamanho da família Demonstração do crescimento exponencial da família Computabilidade Universal com AC Jogo da Vida Propriedades Estáticas de AC's

Autômatos celulares (ACs) são sistemas dinâmicos, discretos quanto ao tempo, espaço e quantidade de variáveis possíveis. São tipicamente representados por um reticulado, onde as células alteram seu estado conforme regras aplicadas localmente. O reticulado pode possuir uma ou mais dimensões e conter um número finito ou infinito de células. Para um número finito de células define-se as condições de contorno (condições da vizinhança das células da extremidade do reticulado), tipicamente utiliza-se condições de contorno periódica, onde as extremidades do reticulado se tocam.

Para um AC clássico é necessário definir seu raio ( $r$ ) e a quantidade de estados possíveis ( $k$ ) de uma célula. Para um AC de raio  $r = 1$  a vizinhança de uma célula é composta

pelas células diretamente a seu redor. No caso de ACs unidimensionais isso é uma célula à esquerda e uma à direita (além da central). É possível definir também um AC com raio múltiplo de 0.5, o que indica uma vizinhança assimétrica.

Se  $r = 0.5$  por exemplo, a vizinhança é composta apenas da célula à esquerda, desconsiderando a da direita, ou vice-versa. Se  $r = 1.5$  a vizinhança passa a ser composta de duas células à direita e uma à esquerda, ou vice-versa, de forma que a quantidade de vizinhos (excluindo a própria célula) dividido por 2 seja sempre igual ao raio ( $x = 2r$ , onde  $x$  é o total de vizinhos adjacentes à célula). Já um AC de  $k = 2$  define a quantidade de estados que uma célula pode assumir, no caso apenas dois estados (usualmente 0 ou 1, ou branco e preto numa representação gráfica).

Os valores de cada célula de um AC no próximo passo de uma evolução temporal, na definição clássica, dependem unicamente dos valores das células de sua vizinhança. Assim, mesmo um AC sendo composto por, digamos, mil células, o valor de cada célula pode depender apenas de três células (no caso de raio  $r = 1$ ). É essa característica de dependência local que torna simples a implementação de um AC. Não obstante, ao se observar o comportamento global nota-se padrões complexos e intrincados, o que vem estimulando diversas pesquisas na área (WOLFRAM, 2002).

ACs são comumente representados como uma matriz, que pode possuir  $d$  dimensões,  $d > 0$ . Cada célula dessa matriz possui um estado em um tempo  $t$ , sendo esse estado um valor qualquer definido em uma tabela de estados. A partir de uma matriz inicial, os estados das células em  $t + 1$  são calculados tendo como base as próprias células e suas vizinhas em  $t$ . A alteração de estado de uma célula obedece à regra de transição desse autômato celular, que pode ser expressa por uma operação matemática, ou simplesmente por uma tabela de transições de estado (WOLFRAM, 2002).

















							
							

Figura 1: Tabela de transição da regra 30 do espaço elementar.

A tabela de transição de um autômato define qual será o valor da célula no próximo *timestep* do AC. Como demonstra a Figura 1, caso a célula central esteja branca, e suas vizinhas da esquerda e direita também estejam brancas, então a célula central se mantém

branca. Mas se apenas a célula da direita fosse preta, a célula central passaria a ser preta. Essa regra de transição de estado se aplica a todas as possibilidades de valores das células da vizinhança. Pela ordem lexicográfica de Wolfram, a primeira transição da tabela é dada pelas células com todos os valores em 1, e a última com toda a vizinhança em 0. Tendo isso em mente pode-se resumir a tabela de transição apenas aos seus resultados de transição. Para a regra 30 isso seria (0, 0, 0, 1, 1, 1, 1, 0).

O espaço elementar dos autômatos celulares é o menor espaço de regras (de raio simétrico), tendo raio de vizinhança igual a 1 e quantidade de valores possíveis igual a 2. A quantidade de regras de um espaço pode ser definida através da equação  $k^{k^{[2r]+1}}$ , o que abrange todas as tabelas de transição possíveis em um espaço de regras. O espaço elementar abrange um total de 256 regras. Esse é um dos espaços mais estudados devido a quantia relativamente pequena de regras (outros espaços chegam facilmente aos milhares, e até bilhões de regras), mas ainda com uma rica variedade de comportamentos. Porém, ao se aumentar qualquer das propriedades do autômato o número total de regras possíveis cresce exponencialmente. Tal crescimento no espaço das regras impede uma busca abrangente de um determinado comportamento, forçando a necessidade de utilização de técnicas mais sofisticadas para análise (SOARES, 2016).

A Figura 2 representa um AC unidimensional através de uma matriz bidimensional, onde cada linha da matriz representa um *timestep* desse AC. Isto é, a primeira linha dessa matriz é o AC em seu estado inicial (no exemplo com uma única célula preta e as demais brancas). Após se aplicar a regra de transição 30, os estados das células mudam, e vê-se na linha 2 que mais células estão na cor preta. Repetindo esse processo por trinta vezes temos na última linha o estado atual do AC, que passou de uma única célula preta para diversas ao decorrer da matriz. Observando-se ainda essa matriz com todas as transições realizadas nota-se um triângulo formando-se na parte externa, e alguns padrões de formações internos.

Por mais simples que sejam ACs em sua essência, a complexidade que são capazes de gerar e a forma contra-intuitiva com que se comportam tornam estudos como os apresentados por VERARDO (2014) e SOARES (2016) importantes na criação de um ferramental que auxilie pesquisas na área.

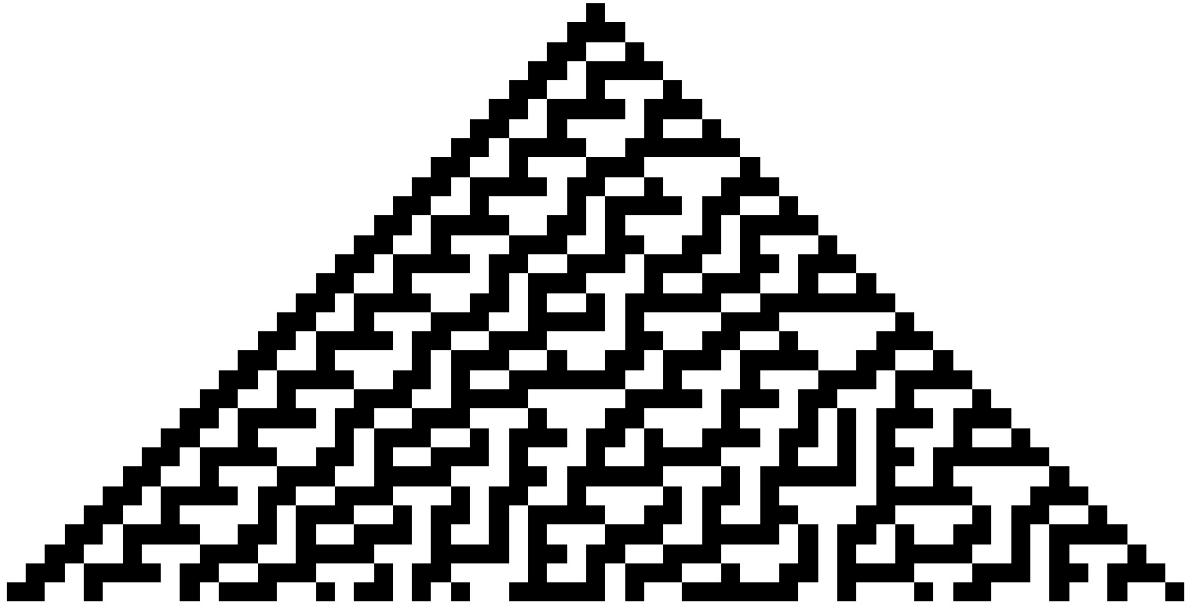


Figura 2: Autômato celular elementar 30, com 30 *timesteps*.

### 3.1 CONSERVABILIDADE DE ESTADOS

“Conservabilidade de estados” é uma das propriedades que uma regra pode possuir. Um autômato celular que utilize uma regra conservativa não tem os valores da soma de suas células alterados durante a evolução do espaço temporal (SOARES, 2016).

Em outras palavras, um AC é tido como conservativo quando a densidade da matriz que o representa não se altera durante suas iterações. Isto é, a soma dos valores em suas células se mantém sempre o mesmo. Para ACs binários isso significa que a quantidade de 1s nunca se altera, enquanto ACs onde  $k > 2$  podem conter um comportamento mais diversificado, mas sempre com a característica de manter a densidade geral do AC (BOCCARA; FUKS, 2002).

Esse tipo de AC vem sendo utilizado nos estudos de fluxo de trânsito, como engarrafamentos, tráfego de massas, cruzamentos de estradas, acidentes de automóveis, etc. Além disso há um interesse da comunidade nessa característica pelo ponto de vista puramente

teórico, tendo MORITA e IMAI (1998) apresentado um AC bidimensional reversível e conservativo capaz de computabilidade universal.

### 3.2 TOTALIDADE E SEMI-TOTALIDADE

Regras consideradas totalísticas são aquelas cujo o próximo valor de determinada célula é definido unicamente pela soma dos valores das células em sua vizinhança. Sendo assim, diferentes valores na vizinhança que somam um mesmo resultado devem sempre convergir a um mesmo valor para célula central (PACKARD; WOLFRAM, 1985).

A semi-totalidade por sua vez define que o próximo valor de uma célula deve depender do seu próprio valor atual e do valor da soma das outras células da vizinhança. Como pode-se observar pela definição, em contraste com os ACs totalísticos, exclui-se a célula central da soma geral onde seria definido sua dependência, e ao invés ela se torna um segundo fator de dependência (PACKARD; WOLFRAM, 1985).

### 3.3 BALANCEAMENTO

Autômatos celulares balanceados são aqueles que possuem em sua matriz de transição quantidades iguais de todos os valores possíveis em  $k$ . Para um AC binário isso significa número igual de 1s e 0s. Devido à essa característica pode-se calcular a quantidade de regras balanceadas dados  $k$  e  $r$  calculando-se as permutações possíveis na tabela de transição. Em especial para o espaço elementar tem-se a equação:  $8!/(4! \times 4!) = 70$  (KRONEMBERGER; DE OLIVEIRA, 2011).

Uma generalização da fórmula para se calcular as regras balanceadas de qualquer espaço pode ser vista na função  $f$  da Equação (1):

$$g(k, r) = k^{\lceil 2r \rceil + 1}$$

$$f(k, r) = \frac{g(k, r)!}{\left(\frac{g(k, r)!}{k}\right)^k} \quad (1)$$

## 4 TEMPLATES

Devido à quantidade de regras de um determinado espaço possivelmente fugir a um trabalho que permita avaliá-las uma a uma comumente cria-se determinadas notações para representar um conjunto de regras específicas. Tais notações geralmente não se tornam objetos de estudo, mas mera ferramenta. DE OLIVEIRA e VERARDO (2014) apresentam uma formalização de tal representação, nomeando-a *Templates* de Autômatos Celulares

Os *Templates* de ACs são uma representação da tabela de transição de estados que podem possuir funções com uma ou mais variáveis e podem representar um conjunto qualquer dentro de um espaço de regras (o que pode inclusive ser o espaço por completo) (DE OLIVEIRA; VERARDO, 2014). Além disso, no *template* é também definido os parâmetros do AC (raio e  $k$ ) e uma lista de funções que devem ser executadas após a operação de expansão (processo de geração das regras representadas pelo *template*). Todas essas definições são contidas no *template* em si, o que o torna uma tupla  $(k, r, c, f)$ , onde  $k$  e  $r$  são o  $k$  e o raio do *template*,  $f$  é a lista de funções a serem executadas após a expansão, e  $c$  é o núcleo do *template* (uma outra tupla, dessa vez representando a tabela de transição das regras). Como o núcleo é a parte central do *template*, é a ele que nos referiremos com mais frequência para compreensão dos operadores e do funcionamento dos *template*.

Como exemplo considere o núcleo do *template*  $(1, 0, 0, 1 - x_1, 1, x_2, x_1, x_0)$  como uma representação de um AC de raio igual a 1 e  $k$  igual a 2. Nesse *template* há as variáveis livres  $x_0, x_1, x_2$  e os valores fixos nas posições 1, 2, 3 e 5. Os campos de variáveis livres podem conter qualquer valor de  $x$  onde  $0 \leq x < k$ . Sendo assim, esse *template* representa todas as regras onde as posições 1, 2, 3 e 5 são iguais às definidas fixamente no *template*, as posições 6, 7 e 8 qualquer valor de  $x$  onde  $0 \leq x < k$ , e a posição 4 respeitando a equação  $1 - x_2$ .

Vale citar que cada uma das posições no *template* representa a transição de uma dada vizinhança. Assim sendo,  $x_0$  que se encontra na posição 8 é a transição da vizinhança  $(0, 0, 0)$ ,  $x_1$  equivale à vizinhança  $(0, 0, 1)$ , até a primeira posição que define a transição da vizinhança  $(1, 1, 1)$  (respeitando a ordem lexicográfica de Wolfram).

Para ilustrar, realizamos a expansão desse *template*, primeiro por considerarmos todas as combinações de transformação das variáveis livres em valores entre 0 e  $k - 1$ :

$$\begin{aligned}
& ((1, 0, 0, 1 - x_1, 1, x_2 = 0, x_1 = 0, x_0 = 0), \\
& (1, 0, 0, 1 - x_1, 1, x_2 = 0, x_1 = 0, x_0 = 1), \\
& (1, 0, 0, 1 - x_1, 1, x_2 = 0, x_1 = 1, x_0 = 0), \\
& (1, 0, 0, 1 - x_1, 1, x_2 = 0, x_1 = 1, x_0 = 1), \\
& (1, 0, 0, 1 - x_1, 1, x_2 = 1, x_1 = 0, x_0 = 0), \\
& (1, 0, 0, 1 - x_1, 1, x_2 = 1, x_1 = 0, x_0 = 1), \\
& (1, 0, 0, 1 - x_1, 1, x_2 = 1, x_1 = 1, x_0 = 0), \\
& (1, 0, 0, 1 - x_1, 1, x_2 = 1, x_1 = 1, x_0 = 1))
\end{aligned} \tag{2}$$

Assim temos todas as regras do espaço representado, faltando apenas calcular as posições não fixas e não de variáveis livres:

$$\begin{aligned}
& ((1, 0, 0, 1, 1, 0, 0, 0), \\
& (1, 0, 0, 1, 1, 0, 0, 1), \\
& (1, 0, 0, 0, 1, 0, 1, 0), \\
& (1, 0, 0, 0, 1, 0, 1, 1), \\
& (1, 0, 0, 1, 1, 1, 0, 0), \\
& (1, 0, 0, 1, 1, 1, 0, 1), \\
& (1, 0, 0, 0, 1, 1, 1, 0), \\
& (1, 0, 0, 0, 1, 1, 1, 1))
\end{aligned} \tag{3}$$

Seguindo essa ideia, um *template* que represente todo o espaço elementar seria composto apenas por variáveis livres,  $(x_8, x_7, x_6, x_5, x_4, x_3, x_2, x_1)$ . Pois realizando todas as substituições possíveis considerando  $k = 2$  teríamos um total de  $2^8$  regras, sendo essas exatamente as do espaço elementar.

Quanto mais restrito um *template*, melhor se torna como ferramenta de pesquisa. Um *template* apenas com variáveis livres pouco contribui no intuito de se realizar testes



e experimentos com ACs. Mas considerando que, por exemplo, queira-se realizar um experimento apenas com as regras conservativas de  $r = 2$  e  $k = 2$ . O total de regras nesse espaço é  $k^{k^{[2r]+1}}$ , sendo então, para os parâmetros definidos, igual a 4.294.967.296. Neste cenário torna-se interessante criar um *template* que restrinja esse espaço apenas às regras conservativas, com a posterior expansão do *template* criado pode-se realizar o experimento pretendido tendo-se evitado o trabalho moroso de identificação das regras conservativas entre as 4.294.967.296 possíveis.

## 4.1 EXPANSÃO

A expansão de um *template* é a transformação do mesmo no conjunto de todas as regras que o compõe. Acaba sendo uma operação custosa por se tratar da transformação da representação de um conjunto no conjunto si (VERARDO, 2014).

Para minimizar o custo da expansão é possível aplicar uma restrição maior ao conjunto que o *template* representa. Um *template* possui três tipos de valores nas posições da tabela de transição que ele representa: variável livre, campo calculado e valor fixo.

Um campo de valor fixo define que para todas as regras geradas após a expansão desse *template*, naquela posição específica, o valor é sempre o definido fixamente no *template*. Para  $k = 3$ , por exemplo, os valores fixos podem ser 0, 1 ou 2.

Um campo de variável livre indica que para aquela posição na tabela de transição a regra pode conter qualquer valor entre 0 e  $k - 1$ . Para o *template* isso significa que aquela posição não é usada para restringir o espaço de regras, já que o campo em verdade não restringe mas, ao invés, pode assumir qualquer valor válido.

O campo calculado é onde as variáveis livres mostram sua utilidade de fato. Pois a ideia principal do *template* é representar um espaço de regras, o que não seria atingido utilizando apenas variáveis livres. Uma posição calculada faz uso dos valores nas demais posições dos *template* para chegar a seu próprio valor. Um exemplo simples seria uma posição de valor  $1 - x_0$ , nesse caso o valor na tabela de transição deve necessariamente ser o resultado da equação, e qualquer regra que não obedeça a esse preceito não se enquadra na representação do *template* que a utilize.

Observa-se pois, para se realizar os cálculos necessários ao campo calculado, é ne-

cessário antes saber-se o valor das variáveis livres. Como as variáveis livres podem conter qualquer valor válido ao AC, torna-se uma questão de realizar os cálculos desses campos para todas as possibilidades de valores das variáveis livres. Para a primeira parte da expansão deve-se por tanto levantar todas as regras possíveis trocando as variáveis livres pelos valores fixos possíveis.

Para tal, assumi-se a base numérica igual ao  $k$  do AC. Considere o *template*  $(2 - x_0, 1 + x_1, x_1, x_0)$  para um AC de raio 0,5 e  $k = 3$ . A  $i$ -ésima regra representada por esse *template* seria  $i$  convertido para a base  $k$ , e cada dígito do valor resultante seria aplicado às variáveis livres.

Em tempo, considere  $i = 7$  (portanto vamos definir a sétima regra representada pelo *template* de exemplo). Nesse caso  $i$  convertido à base 3 seria “21”. Atribuindo esses valores às variáveis livres temos que  $x_1 = 2, x_0 = 1$ . Agora com as variáveis livres com seus devidos valores podemos definir as posições calculadas, tendo a posição quatro e três de nosso exemplo os valores  $2 - 1, 1 + 2$  respectivamente.

A expansão do *template* por completo se baseia em realizar esse processo para todo  $i$ , onde  $0 \leq i < k^m$ ,  $m$  sendo a quantidade de variáveis livres do *template*.

No entanto é importante notar, tal qual no exemplo anteriormente mostrado, que isso por vezes resulta em regras inválidas para o espaço do AC representado. Como pode-se ver, a posição três de nosso *template* é  $1 + x_1 = 1 + 2 = 3$ , porém o AC como limitamos é de  $k = 3$ . A versatilidade da representação por *template* como definida causa o efeito colateral de determinados cálculos resultarem em posições inválidas ao espaço de regras do *template*. Para esses casos as regras inválidas são simplesmente descartadas após a conclusão da expansão.

Aproveitando-se tal mecanismo de exclusão os *templates* também permitem uma regra de exclusão mais explícita e restritiva. Podendo-se conter determinada posição do *template* a valores menores que  $k - 1$ . Aproveitando ainda o exemplo anterior, podemos adicionar a restrição da seguinte forma:  $((2 - x_0) \in \{0, 1\}), 1 + x_1, x_1, x_0)$ . Assim, por mais que a quarta posição do *template* seja calculada, o resultado deve ser 0 ou 1, caso contrário a regra será descartada após a expansão.

## 4.2 INTERSECÇÃO

A operação de intersecção é importante no processo de restringir o espaço estudado, pois através dela é possível combinar diferentes *templates* em um novo que possua todas as propriedades de suas partes. Isso é útil ao se delimitar um conjunto de regras desejado e se torna particularmente importante pois permite que, após uma intersecção, o espaço representado pelo *template* seja mais limitado, o que faz com que a expansão da família seja menos custosa computacionalmente (VERARDO, 2014).

O processo de intersecção é dividido em duas fases, sendo a segunda necessária apenas para *templates* que possuam expressões de limitação de conjunto, como  $x_0 \in \{0, 1, 2\}$ . A primeira fase do processo consiste em igualar as posições de dois *templates* diferentes. Após isso é utilizada a função *Solve* do *Mathematica* (WOLFRAM RESEARCH, 2017).

Tome como exemplo os *templates*  $T_1 = (0 + x_1, 1 - x_0, x_1, x_0)$  e  $T_2 = (x_4, x_3, 1, x_0)$ , ambos representando regras de um AC de raio 0,5 e  $k = 2$ . A primeira fase portanto igualaria os dois *templates* da seguinte maneira:  $(0 + x_1 = x_4, 1 - x_0 = x_3, x_1 = 1, x_0 = x_0)$ . Para esse caso, após a execução da função *Solve* teríamos o *template*  $T_3 = (1, 1 - x_0, 1, x_0)$ . Seguindo esse processo haverão casos sem solução, o que torna os *templates* em questão sem uma intersecção possível.

A segunda fase é responsável por igualar a restrição aplicada aos dois *templates*, para os casos onde isso não é possível os *templates* são considerados disjuntos (isto é, sem uma intersecção possível). Adicionando a restrição aos *templates* anteriores como exemplo poderíamos ter  $T_1 = (0 + x_1, 1 - x_0, x_1, x_0 \in \{0\})$  e  $T_2 = (x_4, x_3, 1, x_0 \in \{1\})$ . Nesse caso  $x_0$  deveria ser 0 de acordo com  $T_1$ , mas seu valor seria na verdade 1 pelo  $T_2$ , formando assim dois *templates* disjuntos.

Agora assumamos  $T_1 = (0 + x_1, 1 - x_0, x_1, x_0 \in \{0, 1\})$  e  $T_2 = (x_4, x_3, 1, x_0 \in \{1\})$ . Desse modo temos que  $x_0$  pode ser 0 ou 1 pelo *template*  $T_1$ , mas apenas 1 pelo *template*  $T_2$ . De forma que a intersecção resulta em  $T_3 = (1, 1 - x_0, 1, x_0 \in \{1\})$ , que é simplificado para  $T_3 = (1, 0, 1, 1)$ .

### 4.3 TEMPLATE PARA REGRAS BALANCEADAS

Sendo uma regra balanceada qualquer regra que possua em sua tabela de transição quantidades iguais de todos os estados presentes em  $k$ , pode-se concluir por dedução que um *template* que represente as regras balanceadas deve possuir todas as suas posições com variáveis livres exceto por uma. Isto porque, não fosse assim e tivéssemos dois campos calculados, o *template* não seria capaz de representar a regra balanceada oriunda da permuta de seus campos calculados.

Assumindo a última posição como a posição a ser calculada, a fórmula para o cálculo do campo foi desenvolvida em (4).

$$b(k, r) = - \left( \sum_{x_i=x_1}^{x_n} x_i - \sum_{i=0}^{k-1} i \frac{n}{k} \right), \text{ onde } n = k^{\lceil 2r \rceil + 1} \quad (4)$$

$$x_0 = - \left( \sum_{x_i=x_1}^{x_n} x_i - \frac{k(k-1)}{2} \frac{n}{k} \right), \text{ onde } n = k^{\lceil 2r \rceil + 1} \quad (5)$$

A fórmula (5) mostra a Equação (4) pouco mais resumida. Assim, tendo-se um *template* de características quaisquer, basta a aplicação desse cálculo na última posição para que o mesmo represente as regras balanceadas. No entanto é necessário que utilize-se uma função pós expansão que retire as regras inválidas, isto é, regras que possuem campos com valores não existentes em  $k$ . Isto porque a fórmula apresentada em (5) força o balanceamento do *template* com base na soma que deveria ser o resultado de todos os campos do mesmo, uma vez que todas as regras balanceadas de um espaço possuem o mesmo valor de soma.

Essa característica do cálculo também ocasiona a precisão apenas para autômatos celulares binários. Portanto a fórmula apresentada não funciona para ACs onde  $k > 2$ .

O *template* (6) demonstra o uso da fórmula (5) para representar regras balanceadas

no espaço  $k = 2, r = 0.5$ , e o *template* (7) para regras onde  $k = 2, r = 1$ .

$$\begin{aligned}
T_0 &= (x_3, x_2, x_1, -(x_3 + x_2 + x_1 - \frac{2(2-1)}{2} \frac{4}{2})) \\
T_0 &= (x_3, x_2, x_1, -(x_3 + x_2 + x_1 - 2)) \\
T_0 &= (x_3, x_2, x_1, 2 - x_3 - x_2 - x_1)
\end{aligned} \tag{6}$$

$$\begin{aligned}
T_1 &= (x_7, x_6, x_5, x_4, x_3, x_2, x_1, \\
&\quad - (x_7 + x_6 + x_5 + x_4 + x_3 + x_2 + x_1 - \frac{2(2-1)}{2} \frac{8}{2})) \\
T_1 &= (x_7, x_6, x_5, x_4, x_3, x_2, x_1, -(x_7 + x_6 + x_5 + x_4 + x_3 + x_2 + x_1 - 4)) \\
T_1 &= (x_7, x_6, x_5, x_4, x_3, x_2, x_1, 4 - x_7 - x_6 - x_5 - x_4 - x_3 - x_2 - x_1)
\end{aligned} \tag{7}$$

A Tabela 1 ilustra o resultado dos cálculos em (6) para todos os valores possíveis em  $x_0, x_1$  e  $x_3$ . Os resultados onde  $x_0 = 2$  e  $x_0 = -1$  são descartados após a expansão, e com isso temos exatamente as regras que possuem números iguais de 1s e 0s. O processo é o mesmo para qualquer raio, alterando-se apenas o tamanho da tabela verdade, dado que a quantidade de variáveis livres ( $x_i$ ) aumenta exponencialmente (como pode ser visto pela função  $g$  em (1)).

Tabela 1: Tabela verdade para  $T_0$ .

$x_3$	$x_2$	$x_1$	$x_0$
0	0	0	2
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	-1

## 5 DIFERENÇA ENTRE TEMPLATES BINÁRIOS

Introduzida por SOARES (2016), trata-se da diferença entre *templates*, que se resume em produzir um conjunto de *templates* que contenham todas as regras representadas por um *template*  $T_m$ , mas não as regras representadas por um *template*  $T_s$ . Em adição à intersecção, a diferença entre *templates* pode ser utilizada como meio de limitar ainda mais o conjunto de regras. Essa operação foi descrita formalmente por SOARES (2016) como:

$$D(T_m, T_s) = C_d \Leftrightarrow E(C_d) = E(T_m) \setminus E(I(T_m, T_s))$$

$$C_d = \{T_1, T_2, \dots, T_n\}$$
(8)

Onde  $C_d$  é o conjunto de *templates* que juntos representam todas regras existentes em  $T_m$ , mas não existentes em  $T_s$ . A função  $E$  é a função de expansão descrita na Subsecção 4.1, e  $I$  a função de intersecção explanada na Subsecção 4.2.

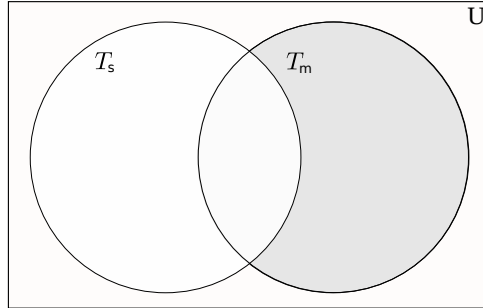


Figura 3: Figura ilustrativa da Equação (8), onde  $C_d$  é o círculo  $T_s$ , exceto pela área de intersecção dos círculos  $T_m$  e  $T_s$  que é excluída de  $C_d$  (SOARES, 2016).

Nota-se que, diferente do processo de intersecção que resulta em um único *template* como resultado, a operação de diferença resulta em um conjunto de *templates*. Isto se deve ao fato de que, para um determinado *template* e determinado conjunto de valores para as variáveis livres, o resultado será sempre a mesma regra. Em outras palavras, para uma dada regra poder ser definida como pertencente a um *template*, ela deve acatar as definições de todas as posições do *template*. Considere o *template*  $T_1$  e a regra  $R_x$  como

definido em (9).

$$\begin{aligned} T_1 &= (x_0 + 1, x_1 - 1, x_1, x_0) \\ R_x &= (1, 0, 1, 0) \end{aligned} \tag{9}$$

Diz-se que a regra  $R_x$  pertence a  $T_1$  caso atenda às equações em  $T_1$ .

$$R_x \in T_1 \iff 1 = x_0 + 1 \wedge 0 = x_1 - 1 \wedge 1 = x_1 \wedge 0 = x_0 \tag{10}$$

Podemos assumir as variáveis livres com os valores da regra, e o exemplo (10) resume-se a:

$$R_x \in T_1 \iff 1 = 0 + 1 \wedge 0 = 1 - 1 \tag{11}$$

Portanto, considerando esse exemplo,  $R_x$  pertence a  $T_1$ . Agora, para se obter todas as regras não definidas em um *template*, se faz necessário negar essa lógica. A negação semântica da definição seria simplesmente dizer que a negação de um *template* é o conjunto de todas regras onde ao menos uma das posições na regra não acate à definição do *template*.

$$\begin{aligned} T_1 &= (x_0 + 1, x_1 - 1, x_1, x_0) \\ R_y &= (0, 0, 1, 0) \\ R_y \notin T_1 &\iff 0 \neq x_0 + 1 \vee 0 \neq x_1 - 1 \vee 1 \neq x_1 \vee 0 \neq x_0 \end{aligned} \tag{12}$$

Como as variáveis livres são na verdade a definição do valor puramente dito, essas posições no *template* sempre serão iguais às mesmas posições na regra, de forma que podemos resumir o exemplo (14) para:

$$R_y \notin T_1 \iff 0 \neq 0 + 1 \vee 0 \neq 1 - 1 \tag{13}$$

A troca do operador lógico é o que força a operação de diferença necessitar de um conjunto de *templates* como resposta à operação. Pois, como por definição um *template*

representa as regras que acatam às operações de todas as posições do *template*, não há meios de construir um *template* que represente as regras em que ao menos uma das posições do *template* não seja respeitada. De modo que, para cada OU lógico da operação de inversão do *template* se acrescenta um novo *template* onde todas as posições são variáveis livres exceto por uma única, que possui a operação invertida do *template* a ser negado. Considere o conjunto  $C_x$  como o conjunto de *templates* que negam  $T_1$ :

$$\begin{aligned} T_1 &= (x_0 + 1, x_1 - 1, x_1, x_0) \\ C_x &= ((1 - (x_0 + 1), x_2, x_1, x_0), (x_3, 1 - (x_1 - 1), x_1, x_0)) \end{aligned} \quad (14)$$

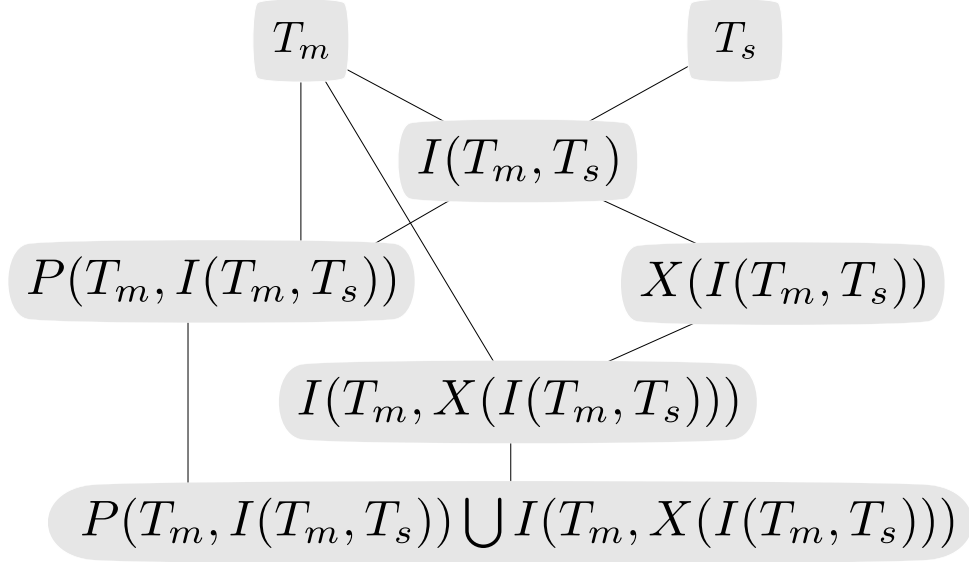


Figura 4: Diagrama representando as principais etapas do algoritmo de diferença entre templates binários (SOARES, 2016).

O fluxograma 4 representa os processos que o algoritmo de diferença realiza para chegar ao conjunto de *templates* desejado. A intersecção entre  $T_m$  e  $T_s$  é realizada e o resultado é utilizado nas demais etapas, pois para o processo de diferença apenas a área de intersecção dos *templates* é o suficiente para realizar a operação. Caso se quisesse apenas as regras  $T_m$  que interseccionam com  $T_s$  poderíamos igualar os *templates* como detalhado na Subseção 4.2. Mas como na verdade o que se deseja são as regras que não estão nessa intersecção, iguala-se  $T_m$  à negação do *template* de intersecção de  $T_m$  com  $T_s$ , e a combinação lógica de equações é então resolvida. O conjunto de equações resultantes é então permutado com as posições equivalentes em  $T_m$ , e cada permutação resulta em um dos



*templates* de resultantes da função  $P$  vista no diagrama 4. A Equação (15) exemplifica esse mecanismo.

$$\begin{aligned}
T_m &= (x_0 + 1, x_2, x_1, x_0) \\
T_s &= (x_3, x_1 - 1, x_1, 1) \\
T_i &= I(T_m, T_s) \\
T_i &= (x_0 + 1 = x_3 \wedge x_2 = x_1 - 1 \wedge x_1 = x_1 \wedge x_0 = 1) \\
T_i &= (x_0 + 1, x_1 - 1, x_1, 1) \\
P(T_m, T_i) &= (x_0 + 1 = 1 - (x_0 + 1) \vee x_2 = 1 - (x_1 - 1) \vee x_1 = 1 - x_1 \vee x_0 = 1 - 1) \\
P(T_m, T_i) &= (x_2 = 1 - (x_1 - 1) \vee x_0 = 0) \\
P(T_m, T_i) &= ((x_0 + 1, 1 - (x_1 - 1), x_1, x_0), (x_0 + 1, x_2, x_1, 0))
\end{aligned} \tag{15}$$

Esse resultado por fim é unificado com os *templates* de exceção da intersecção de  $T_m$  com  $T_s$ . Essa fase é necessária pois os *templates* de exceção nesse caso representam regras que não estão em  $T_s$ , mas podem estar em  $T_m$ .

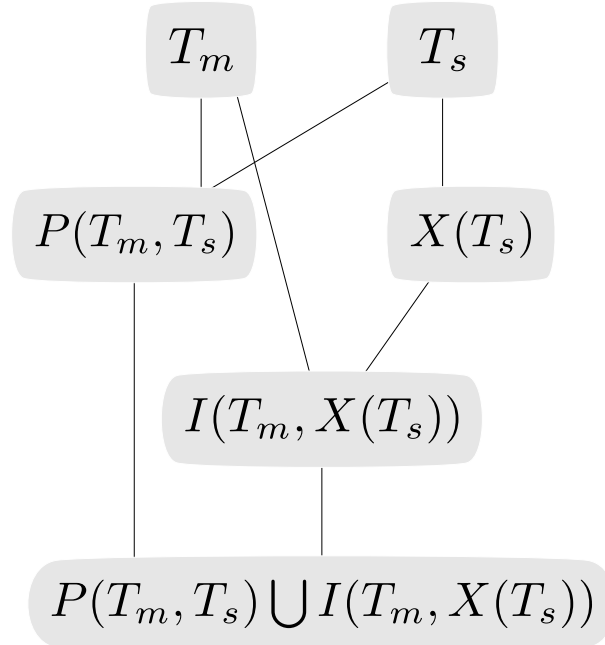


Figura 5: Diagrama representando as principais etapas do algoritmo de diferença entre templates binários, sem a utilização da operação de intersecção (SOARES, 2016).

O digrama 5 demonstra também a operação de diferença, mas dessa vez sem o pro-

cesso de intersecção entre  $T_m$  e  $T_s$ . A intersecção dos *templates* resultaria em informação suficiente para prosseguimento da operação, mas a não intersecção não impacta o processo como um todo, mantendo-se igual exceto pela fase de geração dos *templates* de exceção. Utilizando o mecanismo sem intersecção entre  $T_m$  e  $T_s$  cria-se a necessidade de se gerar os *templates* de exceção de  $T_s$  apenas, porém antes da unificação com o resultado final é mandatório que os *templates* de exceção encontrados sejam interseccionados com  $T_m$ .

## 6 DIFERENÇA ENTRE TEMPLATES $K$ -ÁRIOS

O mecanismo a ser utilizado na operação de diferença entre *templates* de ordem  $k > 2$  aqui sugerido é uma generalização do processo desenvolvido por SOARES (2016), que atende aos *templates* binários.

Pode-se dizer que a fase mais importante do processo existente, e que manteve a operação até então restrita a *templates* onde  $k = 2$ , é a fase de negação de *template* representada pela função  $P$  em 4. A negação de cada posição do *template* especificamente foi realizada utilizando-se a negação binária.

$$n = 1 - (x) \quad (16)$$

A aplicação da Equação (16) a todas as posições do *template* resolve o problema para casos onde  $k = 2$ . Porém para  $k > 2$  se faz necessário que a equação de negação na verdade resulte em todos os resultados possíveis em  $k$  exceto o resultado da equação original. Em outras palavras, assumindo  $k = 4$ , para negar uma determinada posição do *template* onde  $x = 1 - x_2$  é necessário fazer com que  $x$  resulte em todos os resultados possíveis em  $k$  onde  $x \neq 1 - x_2$ .

Como não é possível fazer com que apenas uma equação resulte em mais de um resultado, acaba-se precisando de um conjunto de equações de negação, onde cada equação resulta num valor que respeita a condição  $x \neq 1 - x_2$ . E o conjunto de equações possui todos os resultados possíveis em  $k$ , exceto o resultado onde  $x = 1 - x_2$ .

A Equação (17) mostra um modo genérico de se atender a esse requisito, assumindo  $i$  como um valor entre 1 e  $k - 1$ , deve-se realizar a iteração nesse intervalo. Para cada iteração gera-se uma das equações de negação.

$$n = (k + (x - i)) \bmod k \quad (17)$$

Utilizando-se o mesmo exemplo onde  $k = 4$  mas a equação a ser negada possui  $x = 1$ ,

temos o conjunto  $C_n$  das equações negadas na Equação (18).

$$\begin{aligned}
C_n &= ((k + (x - 1)) \bmod k, \\
&\quad (k + (x - 2)) \bmod k, \\
&\quad (k + (x - 3)) \bmod k) \\
C_n &= ((4 + (1 - 1)) \bmod 4, \\
&\quad (4 + (1 - 2)) \bmod 4, \\
&\quad (4 + (1 - 3)) \bmod 4) \\
C_n &= (4 \bmod 4, 3 \bmod 4, 2 \bmod 4) \\
C_n &= (0, 3, 2)
\end{aligned} \tag{18}$$

Como visto na Equação (18), tendo  $x$  resultando em 1,  $C_n$  resulta em  $(0, 3, 2)$ , que são todos os resultados diferentes de  $x$  mas existentes em  $k$ .

A aplicação dessa equação no mecanismo da operação de diferença resulta somente em uma mudança na fase de negação de *templates*, pois apenas adiciona-se cada uma das equações ao conjunto de equações final que são ligadas pelo operador lógico OU. Como pode ser visto no exemplo (20) (posteriormente deduzido nas equações (21) e (22) respectivamente), que assume  $k = 3$ ,  $r = 0.5$  (com a tabela de transição reduzida para fins didáticos) e os *templates*  $T_m$  e  $T_i$  como definidos em (19).

$$\begin{aligned}
T_m &= (x_0 + 1, x_2, x_1, x_0) \\
T_s &= (x_3, x_1 - 1, x_1, 1) \\
T_i &= I(T_m, T_s) \\
T_i &= (x_0 + 1 = x_3 \wedge x_2 = x_1 - 1 \wedge x_1 = x_1 \wedge x_0 = 1) \\
T_i &= (x_0 + 1, x_1 - 1, x_1, 1)
\end{aligned} \tag{19}$$

$$\begin{aligned}
P(T_m, T_i) &= (x_0 + 1 = (3 + x_0 + 1 + 1) \bmod 3 \\
&\quad \vee x_2 = (3 + x_1 - 1 + 1) \bmod 3 \\
&\quad \vee x_1 = (3 + x_1 + 1) \bmod 3 \\
&\quad \vee x_0 = (3 + 1 + 1) \bmod 3 \\
&\quad \vee x_0 + 1 = (3 + x_0 + 1 + 2) \bmod 3 \\
&\quad \vee x_2 = (3 + x_1 - 1 + 2) \bmod 3 \\
&\quad \vee x_1 = (3 + x_1 + 2) \bmod 3 \\
&\quad \vee x_0 = (3 + 1 + 2) \bmod 3)
\end{aligned} \tag{20}$$

$$\begin{aligned}
P(T_m, T_i) &= (x_0 + 1 = (3 + x_0 + 1 + 1) \bmod 3 \\
&\quad \vee x_2 = (3 + x_1 - 1 + 1) \bmod 3 \\
&\quad \vee x_1 = (3 + x_1 + 1) \bmod 3 \\
&\quad \vee x_0 = (3 + 1 + 1) \bmod 3 \\
&\quad \vee x_2 = (3 + x_1 - 1 + 2) \bmod 3 \\
&\quad \vee x_1 = (3 + x_1 + 2) \bmod 3 \\
&\quad \vee x_0 = (3 + 1 + 2) \bmod 3)
\end{aligned} \tag{21}$$

$$\begin{aligned}
P(T_m, T_i) &= (x_2 = (3 + x_1) \bmod 3) \\
&\quad \vee (x_0 = 2) \\
&\quad \vee (x_2 = (4 + x_1) \bmod 3) \\
&\quad \vee (x_0 = 0))
\end{aligned} \tag{22}$$

Tendo o conjunto de equações solucionadas (22) pode-se seguir realizando a permuta com o *template*  $T_m$ , que no exemplo demonstrado resultaria no conjunto de *templates*

(23).

$$\begin{aligned}
C_x &= P(T_m, T_i) \\
C_x &= ((x_0 + 1, (3 + x_1) \bmod 3, x_1, x_0), \\
&\quad (x_0 + 1, (4 + x_1) \bmod 3, x_1, x_0), \\
&\quad (x_0 + 1, x_2, x_1, 2), \\
&\quad (x_0 + 1, x_2, x_1, 0))
\end{aligned} \tag{23}$$

Assim temos  $C_x$  representando todas as regras existentes em  $T_m$ , mas não em  $T_s$ . Importante notar que a ordem dos processos descritos para a operação de diferença para *templates* binários se mantém. Tal qual permanece os demais mecanismos envolvendo os *templates* de exceção. A única função para o processo de diferença para  $k = 2$  que se altera é a função  $P$  descrita na Subseção 4.2. Assim sendo, mesmo o mecanismo de diferença sem intersecção permanece com seu fluxo inalterado.

Vale observar que na dedução da Equação (20) para (21), e então para (22), que algumas equações foram simplesmente removidas das disjunções lógicas. A retirada das expressões se deve ao fato da expressão sempre resultar em FALSO. Sendo FALSO um valor que nada acrescenta a uma disjunção, já que  $VERDADEIRO \vee FALSO = VERDADEIRO$ , a dita equação pode ser desconsiderada do conjunto.

Assuma a Equação (24) como exemplo.

$$x_0 + 1 = (3 + x_0 + 1 + 2) \bmod 3 \tag{24}$$

Repare que a operação MOD 3 é realizada do lado direito da equidade, de modo que o resultado sempre será sempre 0, 1 ou 2. Para resultar em 0,  $x_0$  precisa ser divisível por 3, para resultar em 1,  $x_0 + 1$  deve ser divisível por 3, e para 2,  $x_0 + 2$  deve divisível por 3. Como  $k = 3$  vamos apenas assumir as demonstrações (25), (26) e (27), onde define-se

o valor de  $x_0$  para 0, 1 e 2 respectivamente.

$$\begin{aligned}
(3 + x_0 + 1 + 2) \bmod 3 &= 0 + 1 \\
(3 + 0 + 1 + 2) \bmod 3 &= 1 \\
6 \bmod 3 &= 1
\end{aligned} \tag{25}$$

$$\begin{aligned}
3 + x_0 + 1 + 2 \bmod 3 &= 1 + 1 \\
3 + 1 + 1 + 2 \bmod 3 &= 2 \\
7 \bmod 3 &= 2
\end{aligned} \tag{26}$$

$$\begin{aligned}
3 + x_0 + 1 + 2 \bmod 3 &= 2 + 1 \\
3 + 2 + 1 + 2 \bmod 3 &= 3 \\
8 \bmod 3 &= 3
\end{aligned} \tag{27}$$

Voltando à Equação (24) que foi originalmente retirada das disjunções, e considerando os resultados das equações ao lado direito da equidade, temos (25), (26) e (27) demonstrando que para todo  $x_0$  possível o resultado da expressão lógica é sempre FALSO, e portanto irrelevante à qualquer disjunção lógica que a contenha.

$$\begin{aligned}
x_0 &= 0 \\
x_0 + 1 &= (3 + x_0 + 1 + 2) \bmod 3 \\
0 + 1 &= 6 \bmod 3 \\
1 &= 0
\end{aligned} \tag{28}$$

$$\begin{aligned}
x_0 &= 1 \\
x_0 + 1 &= (3 + x_0 + 1 + 2) \bmod 3 \\
1 + 1 &= 7 \bmod 3 \\
2 &= 1
\end{aligned} \tag{29}$$

$$\begin{aligned}
x_0 &= 2 \\
x_0 + 1 &= (3 + x_0 + 1 + 2) \bmod 3 \\
2 + 1 &= 8 \bmod 3 \\
3 &= 2
\end{aligned} \tag{30}$$

Embora a não retirada de expressões que caiam nessa situação não impacte na exatidão do resultado da negação dos *templates*, a sua eliminação da disjunção final simplifica os *templates* de negação, gerando um número menor no conjunto resultante.



## 7 CONSIDERAÇÕES FINAIS E CRONOGRAMA

Neste trabalho é apresentado a definição de autômatos celulares clássicos e descrito a notação de *templates* de ACs proposta por VERARDO (2014). Tal notação permite a representação de um conjunto de regras através da generalização da tabela de transição.

A capacidade de representar um grupo de regras é importante pois o espaço de regras de ACs possui um crescimento exponencial ao se alterar seus parâmetros de definição (raio e  $k$ ). A notação de *templates* permite isolar determinada classe de regras com base em definições que essa classe deve seguir. Com esse isolamento as buscas no espaço de regras ficam facilitadas, pois não mais se faz necessário investigar o espaço por completo.

A notação acaba também por permitir operações sobre os *templates* em si. As operações de intersecção e expansão são aqui explanadas, mas implementadas na biblioteca VERARDO e DE OLIVEIRA (2017). A operação de diferença para *templates* binários também é descrita nesse trabalho, e foi inicialmente introduzida por SOARES (2016).

O objetivo desse trabalho é tornar os *templates* de autômatos celulares mais robustos com a generalização da operação de diferença para *templates* de ordem  $k$ -ária. Para que tal fosse possível, a operação apresentada por SOARES (2016) foi analisada e, mantendo-se seu fluxo lógico, um de seus passos foi generalizado para correto funcionamento para ACs de  $k > 2$ .

Com a utilização da biblioteca *CATemplates* (VERARDO; DE OLIVEIRA, 2017) também foi produzido um modelo de *templates* para representação dos ACs balanceados de ordem  $k$ -ária. Tal modelo necessita de testes e simulações para verificação de sua acurácia e correto funcionamento com o método de intersecção de *templates*.

Como trabalhos futuros pretende-se investigar o modelo de representação de ACs balanceados para determinar se é possível definir um grau de balanceamento para um AC (ao invés de simplesmente defini-lo como balanceado ou não balanceado). Ademais pode-se buscar uma melhoria de performance nas operações mais custosas de *templates* (como a operação de expansão).

Tabela 2: Cronograma

	Jan/17	Fev/17	Mar/17	Abr/17	Mai/17	Jun/17	Jul/17
Implementação Template Balanceado		X	X				
Estudo negação de template			X	X			
Pesquisa de viabilidade					X		
Implementação de novo operador de complemento						X	X

Tabela 3: Cronograma

	Ago/17	Set/17	Out/17	Nov/17	Dez/17	Jan/18	Fev/18
Artigo sobre Templates Balanceados	X	X					
Unit de teste para operador de complemento			X				
Análise de performance			X	X			
Expansão de template paralelizada				X	X		
Análise de performance					X	X	
Consolidação dos resultados						X	X

## REFERÊNCIAS BIBLIOGRÁFICAS

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

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](https://doi.org/10.3888/tmj.16-8)>.

KRONEMBERGER, G.; DE OLIVEIRA, P. P. B. A hipótese das regras primitivas e derivadas, na busca construtiva por autômatos celulares reversíveis. In: . São João del Rei, MG: Simpósio Brasileiro de Automação Inteligente, 2011.

MORITA, K.; IMAI, K. Number-conserving reversible cellular automata and their computation universality. Satellite Workshop on Cellular Automata MFCS'98, p. 51–68, 1998.

PACKARD, N. H.; WOLFRAM, S. Two-dimensional cellular automata. *Journal of statistical Physics*, Springer, v. 38, n. 5, p. 901–946, 1985.

SOARES, Z. *Diferença entre Templates de Autômatos Celulares Unidimensionais Binários*. Dissertação (Mestrado) — Universidade Presbiteriana Mackenzie, 2016.

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.], 2017. Disponível em: <<https://github.com/mverardo/CATemplates>>.

WOLFRAM. *A new kind of science*. 1. ed. [S.l.]: Wolfram Media Inc, 2002.

WOLFRAM RESEARCH. *Wolfram Mathematica*. 2017. Disponível em: <<http://www.wolfram.com/mathematica/>>.