UNIVERSIDADE PRESBITERIANA MACKENZIE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Maurício Verardo da Costa

Representando famílias de autômatos celulares por meio de *templates*

Texto de dissertação apresentado ao Programa de Pós-Graduação em Engenharia Elétrica como requisito das exigências do exame para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Pedro Paulo Balbi de Oliveira

São Paulo
2014

C837r Costa, Maurício Verardo da
Representando famílias de Autômatos Celulares por meio de Templates. / Maurício Verardo da Costa. São Paulo, 2014.
52 f.; 30 cm

Dissertação (Programa de Pós-Graduação em Engenharia Elétrica) - Coordenadoria de Pós-Graduação, Universidade Presbiteriana Mackenzie, 2014.
Orientador: Pedro Paulo Balbi de Oliveira
Bibliografia : f. 51-52.

1. Autômatos celulares. 2. Templates.
3. representação k-ária. 4. propriedades estáticas de autômatos celulares.

CDD 629.8

AGRADECIMENTOS

Agradeço à CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior pela bolsa de mestrado concedida, à Universidade Presbiteriana Mackenzie, ao Programa de Pós-Graduação em Engenharia Elétrica e, em especial, ao meu orientador, Pedro Paulo Balbi de Oliveira, por todo o conhecimento que já me passou nesses anos de graduação e mestrado. Agradecimentos especiais também vão para todos os familiares e amigos que me acompanharam durante essa jornada.

RESUMO

A noção de representação de autômatos celulares (ACs) por meio de *templates* é aqui introduzida. Ela consiste em uma generalização da tabela de transições de estado clássica, permitindo a representação de subespaços de autômatos celulares, ao invés de apenas indivíduos isolados. São definidas operações aplicáveis aos *templates*, e seu uso é exemplificado por meio da obtenção de algoritmos que encontram subespaços de regras que apresentam propriedades em comum. Para o desenvolvimento destes algoritmos, a utilização do software Wolfram Mathematica é central, dada sua capacidade de resolução automática de sistemas de equações, além da natureza funcional e simbólica da Wolfram Language, linguagem de programação a ele associada. Também são discutidas as vantagens e desvantagens da utilização deste tipo de representação em outros contextos, e possiblidades de extensão para o trabalho. Como apoio ao conceito dos templates, é apresentada a biblioteca para o Wolfram Mathematica chamada *CATemplates*, disponibilizada em um repositório público.

Palavras-chave: *Autômatos celulares, templates, representação k-ária, propriedades de autômatos celulares, conservabilidade, simetria interna, invariância a trocas de cor, confinamento, totalidade, semi-totalidade.*

ABSTRACT

The notion of a template for representing cellular automata (CA) rules is introduced. This enhances the standard representation based on a rule table, in that it refers to families of cellular automata, instead of a rule alone. Operations applicable to the templates are defined herein, and their use is exemplified in the context of finding representations for rule sets that share properties. Wolfram Mathematica's functional nature and built-in equation-solving capabilities are central to develop these algorithms. The perspectives for using templates in further contexts are also discussed, along with possible extensions to the present work. As a support to the template concept, a Wolfram Mathematica package called *CATemplates* is presented, shared with the community using a public repository.

Keywords: *Cellular automata, templates, k-ary representation, cellular automaton properties, number-conservability, internal simmetry, color blindness, captivity, totalistic CA, outer-totalistic CA.*

Sumário

1 INTRODUÇÃO

Autômatos celulares (ACs) são sistemas dinâmicos que, mesmo governados por regras locais até triviais, de maneira contra-intuitiva podem apresentam comportamento arbitrariamente complexo (WOLFRAM, 2002). As regras são normalmente expressas por meio de tabelas de transições, que determinam os estados das células de um AC no próximo passo de tempo de acordo com sua vizinhança local no passo de tempo atual. A Seção 2 descreve em detalhes o funcionamento dos ACs, e apresenta alguns exemplos de comportamento complexo encontrado neste tipo de sistema dinâmico.

Apesar da existência do espaço elementar, um dos espaços mais explorados devido ao seu número baixo de candidatos (256), existem espaços muito maiores que podem ser considerados na busca de ACs que apresentem um determinado comportamento. Como será visto na Seção 2, as famílias de ACs crescem rapidamente conforme seus parâmetros de construção mudam. Neste momento, propriedades estáticas, derivadas das tabelas de transição, surgem como uma maneira de prever a evolução de um AC sem que a simulação do sistema seja necessária, facilitando a busca.

Propriedades estáticas, calculadas a partir da tabela de transições, servem como indicadores do comportamento de um AC. A simetria interna de uma regra (uma medida que estabelece, de certa maneira, quanto uma regra é similar a outras regras que tenham comportamento dinâmico equivalente), por exemplo, já foi usada como indicador da capacidade de resolução da tarefa de classificação de densidade (descrita na Seção 2) de uma regra (WOLZ; DE OLIVEIRA, 2008). A conservabilidade de estados determina que um AC conservativo nunca muda a quantidade de células em cada estado durante sua evolução. Estas e outras propriedades são descritas em maiores detalhes na Seção 3.

Em diversos casos, a metodologia de um estudo se baseia apenas em um tipo especial de AC, tipicamente obtido através de restrições aplicadas à tabela de transições, como é o caso dos ACs totalísticos (WOLFRAM, 2002), e dos ACs confinados (THEYSSIER, 2004), por exemplo. Estas restrições também podem ser formuladas por meio de propriedades estáticas, como será visto na Seção 3.

Nota-se que a existência de uma representação de conjuntos de ACs é de extrema valia para qualquer estudo que use as propriedades estáticas para realizar suas análises.

Este tipo de representação elimina a necessidade de enumerar um espaço inteiro em busca dos ACs que apresentem o comportamento esperado.

Em meio ao trabalho feito em (LI; PACKARD, 1990), os autores representam um conjunto específico de tabelas de transição em formato k-ário introduzindo variáveis na representação. Em (BETEL; DE OLIVEIRA; FLOCCHINI, 2013), o grafo de De Bruijn é usado de maneira similar. As arestas do grafo recebem valores (variáveis ou estáticos) que representam a saída de uma tabela de transições, e um grafo representa um conjunto de ACs. Construções similares podem ser encontradas em outros trabalhos, mas nunca como o foco principal, e sim como uma ferramenta notacional.

O presente trabalho visa estabelecer uma representação formal para conjuntos de ACs, batizada de *Template*, apoiada por uma biblioteca Open Source, que desenvolvemos, denominada *CATemplate* que estende a Wolfram Language (linguagem de programação utiliza pelo software Wolfram Mathematica (WOLFRAM RESEARCH, 2013)) com funções que permitem realizar operações sobre estes conjuntos.

Além disso, exemplifica a utilização do framework estabelecido por meio da criação de algoritmos geradores de templates representantes de conjuntos cujos ACs compartilham de uma determinada propriedade estática, que também foram disponibilizados no pacote *CATemplate*.

A Seção 4 estabelece a representação formalmente, e apresenta os algoritmos que implementam as operações aplicáveis a templates. A Seção 5 mostra os algoritmos geradores de templates com base em propriedades estáticas. A Seção 6 apresenta conclusões e ideias para trabalhos futuros.

2 AUTÔMATOS CELULARES

Autômatos celulares constituem uma classe de sistemas dinâmicos distribuídos, normalmente discretos quanto ao espaço, tempo e quantidade de estados em que podem se encontrar (WOLFRAM, 2002). Como sistemas governados por regras relativamente simples, autômatos celulares representam um modelo significativo para atacar a questão de como a interação entre componentes extremamente simples podem dar origem a comportamento complexo em um sistema dinâmico, em particular, a soluções de problemas globais.

Autômatos celulares são compostos por um reticulado de células cujos estados mudam conforme o tempo passa, de acordo com regras locais. O reticulado pode estar disposto em qualquer número de dimensões, tipicamente em uma, duas ou três, e pode ter um número finito ou infinito de células. No primeiro caso, torna-se necessário definir qual é o comportamento do sistema nas bordas do reticulado. A estratégia mais comum neste contexto é a de considerar que as bordas do reticulado se tocam, fazendo com que forme um anel no caso unidimensional, ou um toro no caso bidimensional. Neste caso, dizemos que o AC possui condições de contorno periódicas.

Os estados das células são tipicamente representados por números pertencentes ao intervalo discreto $[0, k-1]$ ou por uma cor dentre k previamente definidas. A regra local de um autômato celular age na vizinhança de cada célula, formada pelo conjunto de células adjacentes que influenciam em seu estado nos próximos passos de tempo. A vizinhança é normalmente expressa em termos de um raio r, que indica quantas células adjacentes em cada direção afetarão o estado da célula em questão.

Definindo valores para estes dois parâmetros, uma família (ou espaço) de autômatos celulares é estabelecida. No caso unidimensional, os valores $r = 1$ e $k = 2$ (3 células por vizinhança e 2 estados possíveis) dão origem ao chamado espaço elementar de autômatos celulares, que é a família mais extensivamente estudada até o momento, devido ao seu tamanho pequeno (apenas 256 regras) e fenomenologia extremamente rica (WOLFRAM, 2002).

Para os propósitos do presente texto, os autômatos celulares unidimensionais, binários com número fixo de células no reticulado e condições de contorno periódico são adotados

como padrão.

Todo AC é governado por uma regra que relaciona cada possível vizinhança de uma célula com o estado que ela deve assumir no próximo passo de tempo. Sua representação mais comum é a tabela de transição de estados, que é uma tabela de n-uplas, sendo $n = 2r + 1$ no caso unidimensional, cujos elementos são transições formadas por uma das possíveis vizinhanças e um estado resultante, ordenada lexicograficamente de acordo com a vizinhança. Aqui, é utilizada a ordenação lexicográfica de Wolfram, em que o primeiro item da n-upla é aquele cuja vizinhança é formada por todas as células no estado $k - 1$, e o último item é aquele cuja vizinhança é formada apenas por zeros.

Como exemplo, a tabela de transições do AC elementar 184 é a tupla

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

que também pode ser vista em sua representação icônica, mostrada na Figura 1, em que cada uma das oito transições de estados é representada com a vizinhança na parte de cima de cada ícone e o bit de saída na parte de baixo, transformando-se os 1s em pretos e os 0s em brancos.

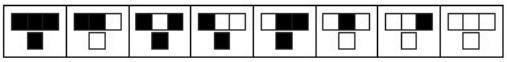

Figura 1: *Representação icônica da tabela de transições do AC elementar 184.*

Ao descartar as vizinhanças e manter apenas os estados resultantes das transições, a tabela é convertida em sua representação k-ária:

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

A representação k-ária só é compreensível caso os valores de k e r já sejam conhecidos, uma vez que isto torna possível a reconstrução e ordenação das vizinhanças.

Tratando uma tabela de transições em forma k-ária como uma sequência binária, é possível convertê-la para sua representação decimal. O número obtido serve como iden-

tificador único de uma regra em seu respectivo espaço. Neste exemplo, o número obtido ao converter a sequência binária 10111000 para decimal é 184. Por isso, este é conhecido como o AC elementar 184.

O tamanho de uma família também é determinável a partir dos parâmetros k e r. A vizinhança de um AC contém um número δ de células, o que torna possível inferir que cada tabela de transições de uma família deve conter k^δ itens. Considerando que cada item pode resultar em um de k estados, o número de regras dentro de um determinado espaço é dado por:

$$k^{k^\delta} \tag{3}$$

No caso unidimensional, $\delta = 2r+1$, portanto o número de itens na tabela de transições de um AC unidimensional é dado por:

$$k^{k^{2r+1}} \tag{4}$$

É fácil perceber que qualquer aumento nos valores de k e r resultam em famílias muito maiores. O espaço de ACs unidimensionais binários de raio 2, por exemplo, possui $2^{2^5} = 4.294.967.296$ ACs, uma quantidade muito maior do que os $2^{2^3} = 256$ do espaço elementar original. Este aumento representa um dos grandes desafios ao buscar ACs com propriedades ou comportamentos específicos, já que dificulta buscas de força bruta em qualquer espaço maior que o elementar.

Quando realizando buscas desta natureza, uma boa estratégia é a de utilizar propriedades estáticas, calculadas a partir da tabela de transições do AC, para prever o comportamento de um AC sem que seja necessário realizar sua evolução espaço-temporal propriamente dita. No presente trabalho, as propriedades estudadas (e mostradas em maiores detalhes na Seção 3) são a conservabilidade de estados, a simetria interna, a invariância a trocas de cor, o confinamento, a totalidade e a semi-totalidade.

No caso unidimensional, é possível visualizar a evolução do sistema usando um diagrama espaço temporal, em que o tempo segue de cima para baixo, e os estados das células são representados por cores. Para ACs binários, células no estado zero são tipicamente representadas pela cor branca, e as células no estado um são representadas pela cor preta.

Exemplos de diagramas espaço-temporais são dados na Figura 2.

Os ACs elementares podem ser classificados quanto ao seu comportamento dinâmico. De acordo com Wolfram (2002), eles se encaixam em quatro categorias distintas.

A primeira categoria engloba os ACs cuja evolução tipicamente leva para uma configuração de ponto fixo homogêneo. A segunda envolve todos os ACs cuja evolução tende a levar para configurações que apresentam algumas estruturas simples que, ou continuam iguais para sempre, ou se repetem em ciclos periódicos pequenos. Os ACs pertencentes à terceira classe são aqueles que apresentam diagramas espaço temporais que aparentam ser caóticos, apesar de mostrarem pequenas estruturas durante sua evolução. A quarta e última classe é reservada para ACs cuja evolução faz com que surjam estruturas locais que interagem, levando a comportamentos complexos; adicionalmente, estes ACs possuem transientes bastante longos, comparativamente aos das demais classes.

A Figura 2 mostra exemplos dos diagramas espaço-temporais de quatro ACs elementares pertencentes a classes de comportamento dinâmico distintas.

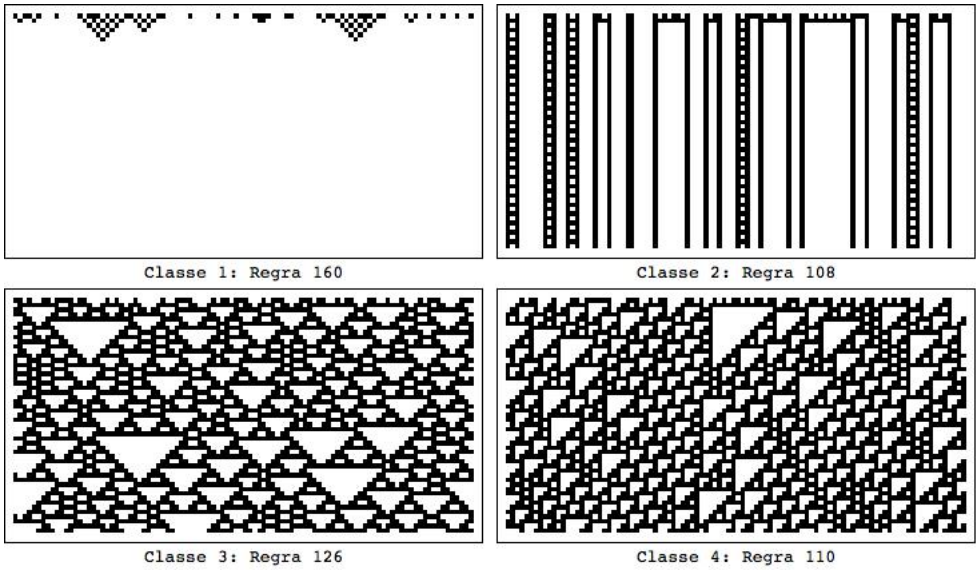

Figura 2: *Exemplos de ACs das quatro classes de comportamento dinâmico definidas por Wolfram (2002).*

Os ACs da quarta classe são especialmente importantes por apresentarem estruturas geradoras de complexidade. O AC elementar 110, por exemplo, é conhecido por ser uma máquina cuja computabilidade universal foi provada por Cook (2004) (depois de sua divulgação em (WOLFRAM, 2002)). Em outras palavras, mesmo sendo uma máquina

que se baseia na interação de componentes extremamente simples, o AC 110 tem poder computacional para resolver qualquer problema que uma máquina de Turing é capaz de resolver. Sua programação é feita por meio da condição inicial que lhe é atribuída.

Outro AC que tem a propriedade da computabilidade universal é o Jogo da Vida. Proposto por John Horton Conway em (BERLEKAMP; CONWAY; GUY, 1982), o jogo da vida é um autômato celular binário bidimensional de grande importância para a área da vida artificial. A ideia central que rege o jogo da vida é a de que as células podem estar mortas ou vivas. Uma célula viva continua viva caso tenha exatamente dois ou três vizinhos vivos. Caso tenha menos do que dois vizinhos vivos, a célula morre por isolamento. Se tiver mais do três, morre por superlotação do espaço. Uma célula nasce se tiver exatamente três vizinhos vivos (KARI, 2005).

Destacam-se no jogo da vida as estruturas que emergem quando certas condições iniciais são dadas. Alguns exemplos são mostrados na figura 3, encontrada em Kari (2005). Entre os exemplos dados, destacam-se os *gliders*, estruturas de partículas que se movem pelo reticulado conforme o tempo passa, e servem como peça fundamental para realizar computações. *Glider Guns* também são extremamente importantes, por criarem gliders durante a simulação.

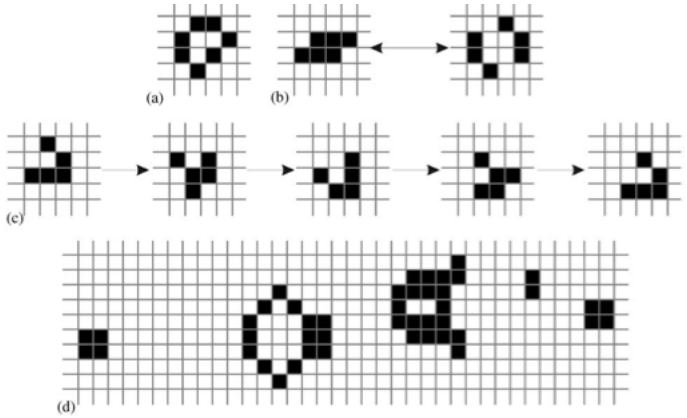

Figura 3: *Exemplos de estruturas que aparecem no jogo da vida, e provocam efeitos interessantes quando interagem: a) Estrutura fixa b) Oscilador de período dois c) Glider d) Glider Gun Imagem retirada de Kari (2005).*

Ainda que alguns ACs sejam conhecidos por terem computabilidade universal quando programados de maneira explícita, isto é, quando programados através de sua condição

inicial, os ACs também podem ser estudados pelos efeitos intrínsecos que suas regras locais causam no reticulado como um todo. Com o intuito de melhor compreender o poder computacional implícito nas regras dos autômatos celulares, alguns problemas de *benchmark* são definidos na literatura. Dentre eles, o mais comum é o problema da classificação de densidade (*Density Classification Task*, ou *DCT*).

Para resolver a DCT em sua definição clássica, um AC unidimensional, em reticulado cíclico, deve levar qualquer condição inicial arbitrária que tenha mais pretos do que brancos para um estado de ponto fixo onde todas as células estão no estado preto, e qualquer condição inicial que tenha mais brancos do que pretos para um estado de ponto fixo onde todas as células estão no estado branco.

É provado que para resolver a DCT perfeitamente, um AC precisaria ser conservativo, ou seja, não poderia mudar o número de células em cada estado durante sua evolução, dada qualquer condição inicial (FUKŚ, 2000). Este fato entra em contradição direta com a definição clássica da DCT, já que para levar uma condição inicial para um estado de todos brancos ou todos pretos, o AC precisa obviamente alterar a quantidade de células em cada estado conforme o tempo passa. Isso indica que a DCT não pode ser resolvida quando formulada de acordo com sua definição clássica ((DE OLIVEIRA, 2013) e (DE OLIVEIRA, 2014)).

A partir desta descoberta, os esforços no contexto da DCT mudaram para encontrar as melhores regras que resolvam o problema imperfeitamente, ou seja, que resolvam o problema para a maior quantidade possível de condições iniciais.

A melhor regra imperfeita para a DCT encontrada até o momento, conhecida como WdO1, foi reportada em (WOLZ; DE OLIVEIRA, 2008), por meio de um algoritmo evolutivo sofisticado que usava, entre outras propriedades importantes, a simetria interna de uma regra na sua função de *fitness*. Em uníssono com o fato de uma regra perfeita para a DCT necessitar de conservabilidade de estados, a WdO1 e outras boas regras conhecidas possuem distância de Hamming muito pequena de outras regras conservativas em seus respectivos espaços (KARI; LE GLOANNEC, 2012).

Em geral, a conservabilidade de estados e simetria interna de uma regra são duas propriedades estáticas importantes quando o objetivo é determinar a habilidade de um AC para resolver a DCT, e servem como bons exemplos para o conceito dos *templates* de

ACs. Foram, portanto, as primeiras propriedades abordadas pelos algoritmos geradores de templates do projeto, como descrito em (DE OLIVEIRA; VERARDO, 2014a) e (DE OLIVEIRA; VERARDO, 2014b).

3 PROPRIEDADES ESTÁTICAS DOS AUTÔMATOS CELULARES

Uma propriedade estática de um AC é uma propriedade calculada a partir da sua tabela de transições. As propriedades estáticas permitem que o comportamento de um AC durante sua evolução possa ser previsto, eliminando a necessidade de realizar uma simulação do sistema e verificar depois o resultado. A conservabilidade de estados, por exemplo, garante que a quantidade de células em cada estado nunca muda durante a evolução de um AC.

Esta seção descreve as propriedades estudadas pelo presente trabalho, utilizadas posteriormente como exemplos de utilização do framework de templates.

3.1 CONSERVABILIDADE DE ESTADOS

A conservabilidade de estados é uma propriedade apresentada por alguns ACs, em que a soma dos estados das células individuais em qualquer condição inicial não muda durante a evolução espaço temporal; em particular, para ACs binários, isso significa que o número de células no estado um sempre se mantém o mesmo. Este tipo de AC é útil, por exemplo, para modelar sistemas como o tráfego de carros, onde um carro não pode simplesmente aparecer ou desaparecer conforme o tempo passa (KARI; LE GLOANNEC, 2012). O AC elementar 184 é um exemplo de AC conservativo.

Para que um AC seja conservativo, foi estabelecido em (BOCCARA; FUKŚ, 2002) que sua regra local f com vizinhança $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ deve respeitar a seguinte condição necessária e suficiente, para todas as transições:

$$f(\alpha_1, \alpha_2, \ldots, \alpha_n) = \alpha_1 + \sum_{i=1}^{n-1} (f(0_1, 0_2, \ldots, 0_i, \alpha_2, \alpha_3, \ldots, \alpha_{n-i-1}) - f(0_1, 0_2, \ldots, 0_i, \alpha_1, \alpha_2, \ldots, \alpha_{n-i})) \quad (5)$$

onde $0_0, 0_1, ..., 0_i$ corresponde a uma sequência formada por i zeros.

Esta condição é chave para obter templates representantes dos ACs conservativos de um espaço.

Como exemplo, vamos usar as condições em questão para provar que o AC 184 é conservativo. Sua tabela de transições pode ser vista na Equação (6).

$$\begin{aligned}&(((1,1,1),1),((1,1,0),0),((1,0,1),1),((1,0,0),1),\\&((0,1,1),1),((0,1,0),0),((0,0,1),0),((0,0,0),0)))\end{aligned} \tag{6}$$

A aplicação das condições estabelecidas na Equação (5) na tabela de transições de um AC elementar qualquer gera o seguinte sistema de equações:

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

Quando simplificado, o sistema se torna:

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

Pode-se verificar que, como notado em (SCHRANKO; DE OLIVEIRA, 2010), qualquer transição iniciada por zero que não seja homogênea (ou seja, que não é formada apenas por zeros) gera uma condição tautológica no sistema estabelecido. Assim, no exemplo que segue (e também no algoritmo que será descrito na Seção 5.1), estas vizinhanças

são descartadas antes de continuar o processo de verificação (no caso do algoritmo, de geração) da conservabilidade do AC.

Continuando o processo, atribuímos os valores de f conforme a regra local estabelecida na regra do AC 184, obtendo o sistema:

$$\begin{cases} 0 &= 0 \\ 1 &= 1-0-0 \\ 1 &= 1-0 \\ 0 &= 1+(0-1) \\ 1 &= 1 \end{cases} \tag{9}$$

A partir daqui, é trivial identificar que todas as condições do sistema são verdadeiras e, portanto, o AC elementar 184 é conservativo.

A conservabilidade de estados tem um papel central no estudo da DCT, já que o argumento principal da impossibilidade de resolvê-la perfeitamente é o de que a conservabilidade é uma propriedade necessária em um AC para que ele tenha chance de atingir este objetivo, entrando em contradição com a própria definição do problema.

3.2 SIMETRIA INTERNA

Além da conservabilidade de estados, a simetria interna de uma regra também tem papel importante na resolução da DCT, e pode ser uma medida relevante em qualquer contexto onde uma propriedade é compartilhada por todos os membros de uma classe de equivalência dinâmica. Em (WOLZ; DE OLIVEIRA, 2008) e (KARI; LE GLOANNEC, 2012), por exemplo, regras com simetria interna máxima em relação à transformação composta foram centrais para as conclusões obtidas relacionadas à DCT.

Para entender completamente como essa propriedade funciona, uma explicação sobre transformações de regras e classes de equivalência dinâmica se faz necessária. O texto que segue é válido para regras binárias, apesar de os conceitos poderem ser generalizados para qualquer valor de k.

Dada uma tabela de transições de AC, podem ser aplicados três tipos de transformações que resultam em ACs com comportamento dinâmico equivalente. A primeira,

transformação por *conjugação*, é obtida ao inverter todos os estados das células da tabela de transições. O segundo tipo de transformação é obtido ao refletir os bits das vizinhanças da tabela. É chamada, portanto, de *reflexão*. A *composição da conjugação e da reflexão* forma um terceiro tipo de transformação, independente da ordem. Após a realização de qualquer uma das transformações, a tabela de transições deve ser novamente ordenada conforme a ordenação de Wolfram.

Quando aplicada a transições individuais, a conjugação (para regras binárias) é definida na expressão:

$$c(((\alpha_0, \alpha_1, \ldots, \alpha_{n-1}), \alpha_{out})) = (1 - \alpha_0, 1 - \alpha_1, \ldots, 1 - \alpha_{n-1}), 1 - \alpha_{out})) \tag{10}$$

em que o símbolo α_{out} representa a saída da transição. Já a reflexão é definida como:

$$r(((\alpha_0, \alpha_1, \ldots, \alpha_{n-1}), \alpha_{out})) = ((\alpha_{n-1}, \alpha_{n-2}, \ldots, \alpha_0), \alpha_{out})) \tag{11}$$

E a transformação composta resulta em:

$$cr(((\alpha_0, \alpha_1, \ldots, \alpha_{n-1}), \alpha_{out})) = ((1 - \alpha_{n-1}, 1 - \alpha_{n-2}, \ldots, 1 - \alpha_0), 1 - \alpha_{out})) \tag{12}$$

Todas as transformações podem ser aplicadas a tabelas de transições inteiras, mapeando estas aplicações individuais a cada transição, e depois reordenando a tabela na ordem lexicográfica. Podem também ser aplicadas à vizinhanças ou saídas individualmente. Todas as transformações foram implementadas como funções da biblioteca *CATemplates*.

Uma tabela, transição ou vizinhança é dita invariante a uma determinada transformação quando a aplicação da transformação não surte efeito. A aplicação da reflexão na vizinhança $(1, 1, 1)$, gera a própria vizinhança $(1, 1, 1)$, por exemplo.

Com as transformações definidas, torna-se trivial identificar quais ACs de um determinado espaço têm comportamento dinâmico equivalente. Por exemplo, considere a tabela de transições do AC elementar 110, mostrada na Figura 4. Ao aplicar a transformação por conjugação, obtemos a tabela de transições do AC elementar 137 (Figura 5). Ao aplicar a transformação por reflexão, obtemos o AC 124 (Figura 6). E ao aplicar a composição das duas, obtemos o AC elementar 193 (Figura 7).

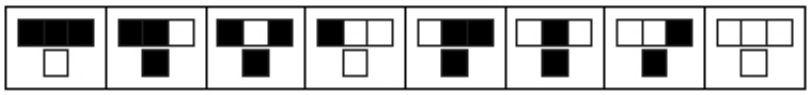

Figura 4: *Tabela de transições do AC elementar 110.*

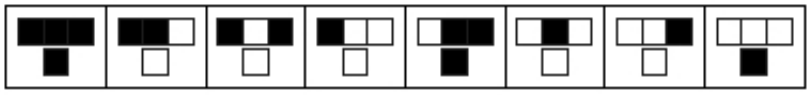

Figura 5: *Tabela de transições do AC elementar 137, obtida ao aplicar a transformação de conjugação na regra 110.*

Estes quatro ACs estão dentro de uma mesma classe de equivalência dinâmica. É fácil entender o motivo ao olhar para seus diagramas espaço-temporais, como pode ser visto na Figura 8. O espaço elementar de ACs, formado por 256 regras, pode ser particionado em 88 classes de equivalência dinâmica.

Ao comparar as tabelas de transição de um AC com outro que seja dinamicamente equivalente, obtido através de uma das transformações, é possível contar o número de transições de estados iguais. De certa maneira, isso é uma medida do quão simétrico um AC é internamente, com relação à transformação usada, seja ela qual for. Esta contagem recebe o nome de simetria interna de uma regra. Por exemplo, o AC elementar 110 tem um valor de simetria interna igual a 2 em relação à sua transformação por conjugação, já que compartilha as transições $((1,0,0),0)$ e $((0,1,1),1)$ com a regra resultante de sua conjugação, o AC elementar 137.

Ao repetir o processo com o AC elementar 150, no entanto, o resultado obtido é diferente. Considere-se a tabela de transições do AC elementar 150, mostrada na Figura 9.

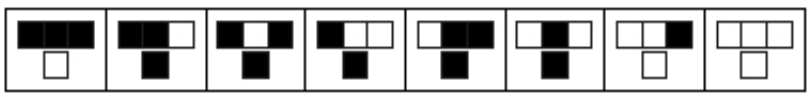

Figura 6: *Tabela de transições do AC 124, obtida ao aplicar a transformação de reflexão na regra 110.*

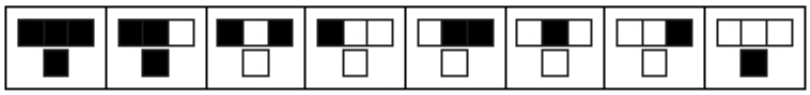

Figura 7: *Tabela de transições do AC elementar 193, obtida ao aplicar a transformação de conjugação seguida da transformação de reflexão na regra 110.*

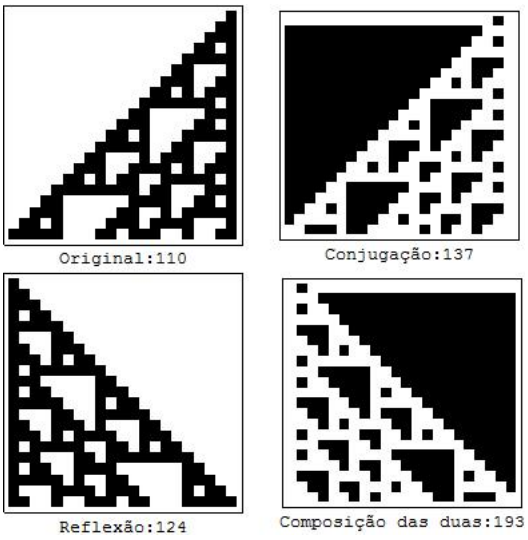

Figura 8: *Diagramas espaço-temporais dos ACs elementares 110, 137, 124 e 193. É fácil perceber que se tratam de ACs com comportamento dinâmico equivalente.*

É possível perceber que a transformação por conjugação, quando aplicada à regra 150, resulta na própria regra 150. Este fato indica que a regra 150 tem um valor de simetria interna por conjugação de 8, que é o maior valor possível em ACs do espaço elementar, já que sua tabela de transições tem 8 itens. O AC 150 é, portanto, invariante a conjugação. Na realidade, qualquer das três transformações, quando aplicada ao AC 150, resulta no próprio AC 150, indicando que ele tem valores de simetria interna máxima independente da transformação aplicada, tornando-se assim invariante a todas as transformações.

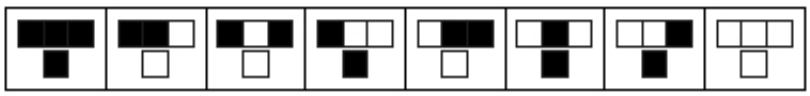

Figura 9: *Tabela de transições do AC elementar 150. Nota-se que qualquer transformação feita nesta regra resultará nela própria.*

As transformações são funções que relacionam transições. Tomando o exemplo da transformação por conjugação, considere o esquema da Figura 10.

Figura 10: *Relação entre as vizinhanças de um AC elementar, obtida através da aplicação da conjugação. Fica evidente que a conjugação relaciona a transição* $(1,1,1)$ *à transição* $(0,0,0)$*, a transição* $(1,1,0)$ *à* $(0,0,1)$*, e assim por diante.*

Para que as transições de um AC sejam simétricas de acordo com a conjugação, é necessário que os resultados de ambas sejam conjugações entre si. No caso binário, por exemplo, se $f(0,0,0) = 1$, então $f(1,1,1) = 0$.

De modo geral, a definição de um resultado para qualquer transição define, por consequência, o resultado da sua equivalente obtida através de qualquer transformação.

Em se tratando de ACs com simetria interna máxima, é necessário que todas as transições sejam simétricas. Sendo assim, torna-se evidente que, no caso da conjugação, quando um AC elementar apresenta o valor máximo de simetria, apenas quatro dos resultados de sua tabela de transições são variáveis, já que os outros quatro são dependentes dos valores destes primeiros, como mostra a Figura 11.

É possível então inferir que o número de regras com simetria máxima por conjugação no caso elementar é dado pela expressão $2^4 = 16$. Generalizando, a quantidade de regras com simetria máxima por conjugação de qualquer espaço binário é dado por $2^{2^{2r+1}/2} = 2^{2^{2r}}$.

A reflexão é a única transformação que gera vizinhanças invariantes. No caso elementar, por exemplo, as vizinhanças pertencentes ao conjunto $\{(1,1,1),(1,0,1),(0,1,0),(0,0,0)\}$ são invariantes a reflexão. Estas quatro vizinhanças não se relacionam a nenhuma outra por meio da reflexão, fazendo com que as respectivas transições contem na simetria interna de uma regra independentemente do valor de sua saída. Sendo assim, o valor mínimo de simetria interna quanto à reflexão para uma regra do espaço elementar é quatro. Genera-

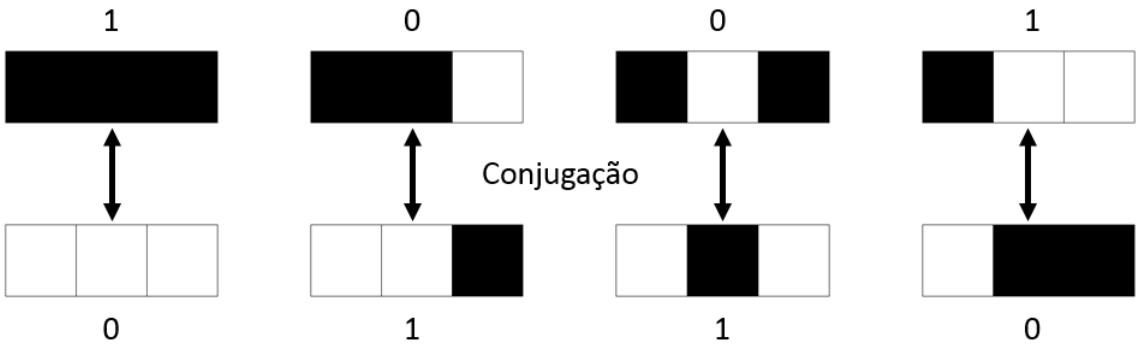

Figura 11: *Tabela de transições do AC elementar 150. É trivial verificar que este AC tem valor de simetria interna por conjugação máximo, uma vez que os resultados das transições da linha de cima são sempre opostos aos da linha de baixo, fazendo com que a conjugação conte em todas as transições.*

lizando, o valor mínimo de simetria interna de acordo com alguma transformação é igual ao número de vizinhanças invariantes de acordo com a transformação.

3.3 INVARIÂNCIA A TROCAS DE COR

Os ACs invariantes a trocas de cor (*color blind CA*) estão intimamente ligados à transformação de conjugação. Um AC é dito invariante a trocas de cor se for invariante à aplicação de uma permutação nos estados das células de sua tabela de transições (SALO; TÖRMÄ, 2014). No caso binário, existe apenas uma permutação possível, em que todas as células no estado zero são trocadas para o estado um e vice e versa. Esta permutação, que também pode ser escrita como $\{0 \to 1, 1 \to 0\}$, é exatamente a operação que a conjugação faz no caso binário, como visto na Seção 3.2. Fica evidente então que as regras invariantes a trocas de cor são exatamente as regras que possuem máxima simetria interna por conjugação.

Para casos onde $k > 2$, no entanto, existe um número maior de permutações de estado possível, fazendo necessária a generalização da operação de conjugação para uma determinada permutação.

A conjugação de uma transição de acordo com a permutação ρ_i é dada por:

$$c(\rho_i, ((\alpha_0, \alpha_1, \ldots, \alpha_{n-1}), \alpha_{out})) = (\rho_i(\alpha_0), \rho_i(\alpha_1), \ldots, \rho_i(\alpha_{n-1})), \rho_i(\alpha_{out}))) \tag{13}$$

Para todo valor de k, é possível obter um conjunto C_ρ das permutações de estado possíveis. No caso onde $k = 3$, por exemplo:

$$\begin{aligned} C_\rho = \{\{0 \to 0, 1 \to 2, 2 \to 1\}, \{0 \to 1, 1 \to 0, 2 \to 2\}, \{0 \to 1, 1 \to 2, 2 \to 0\}, \\ \{0 \to 2, 1 \to 0, 2 \to 1\}, \{0 \to 2, 1 \to 1, 2 \to 0\}\} \end{aligned} \tag{14}$$

Uma regra é invariante a trocas de cor se, e somente se, a aplicação da conjugação usando qualquer permutação do conjunto C_ρ gera como resultado a própria regra.

Aqui, cabe notar que, no caso binário, a conjugação estabelece sempre uma relação bidirecional entre as vizinhanças. Podemos dizer que a vizinhança $(0, 0, 0)$ está relacionada à vizinhança $(1, 1, 1)$ através da permutação $\rho = (0 \to 1, 1 \to 0)$ e vice e versa, uma vez que a aplicação da permutação em qualquer das vizinhanças gera a outra. No entanto, em casos não binários, a relação estabelecida por uma permutação pode ser unidirecional.

Considere o caso da vizinhança $(0, 0, 0)$ para $\rho = (0 \to 1, 1 \to 2, 2 \to 0)$. Quando a permutação é aplicada em $(0, 0, 0)$, o resultado é $(1, 1, 1)$. Entretanto, quando a permutação é aplicada na vizinhança $(1, 1, 1)$, o resultado é a vizinhança $(2, 2, 2)$. Este fato é importante para a compreensão do processo gerador de templates de regras invariantes a trocas de cor, descrito na Seção 5.3.

Qualquer regra que não use o valor das células diretamente é invariante a trocas de cor por definição. Em (SALO; TÖRMÄ, 2014), a seguinte definição de regra do espaço definido por $k = 3$ e $r = 1$ é apontada como pertencente a esta classe:

$$f(\alpha_0, \alpha_1, \alpha_2) = \begin{cases} \alpha_2 & \text{se } \alpha_0 = \alpha_1 \neq \alpha_2 \\ \alpha_0 & \text{se } \alpha_0 \neq \alpha_1 = \alpha_2 \\ \alpha_1 & \text{em caso contrário} \end{cases} \tag{15}$$

É trivial identificar que esta regra é invariante a trocas de cores, pois ela sempre escolhe o estado menos frequente da vizinhança como resultado, não importando qual é o

estado propriamente dito. Qualquer permutação, quando aplicada à tabela de transições desta regra gerará ela mesma.

3.4 CONFINAMENTO

Um AC é dito *confinado* (*captive*, em Inglês) quando todas as transições de sua tabela de transições têm como resultado um estado que faça parte da respectiva vizinhança (THEYSSIER, 2004).

Formalmente, seja f a função local de um AC para a vizinhança $(\alpha_0, \ldots, \alpha_{n-1})$, onde n é o tamanho da vizinhança, então um AC é dito confinado caso a condição a seguir seja válida.

$$f((\alpha_0, \ldots, \alpha_{n-1})) = \beta, \beta \in \{\alpha_0, \ldots, \alpha_{n-1}\} \tag{16}$$

Cabe notar que, no caso binário, qualquer AC cuja $f(0_0, 0_1, \ldots, 0_{n-1}) = 0$ e $f(1_0, 1_1, \ldots, 1_{n-1}) = 1$ é um AC confinado, independente do raio.

Os ACs confinados foram propostos pela primeira vez em (THEYSSIER, 2004) com o intuito de explorar esta subclasse e compreender quais tipos de propriedade ela apresenta. Dentre os diversos resultados do trabalho, prova-se que existem ACs confinados intrinsecamente universais, isto é, que conseguem simular qualquer outro AC de mesma dimensão.

Uma noção de densidade para uma propriedade de ACs é introduzida em (THEYSSIER, 2005), como uma maneira de medir o quão frequente aquela propriedade é num espaço de ACs. O trabalho utiliza esta noção para provar que a maioria dos ACs confinados é intrinsecamente universal.

3.5 TOTALIDADE E SEMI-TOTALIDADE

Um AC totalístico é um AC cujas transições levam em consideração apenas as somas dos estados da vizinhança correspondente, ou, de forma análoga, a média dos estados da vizinhança correspondente. A quantidade de ACs totalísticos de um determinado espaço definido por k e r é dada pela expressão $k^{(1+(k-1)(2r+1))}$ (WOLFRAM, 2002).

Os ACs totalísticos (*totalistic CAs*) são utilizados, por exemplo em Wolfram (2002), como uma maneira de restringir o espaço de ACs com $k = 3$ e $r = 1$, composto por $3^{27} = 7.625.597.484.987$ regras a um espaço de apenas 2.187 itens.

A Figura 12 mostra o exemplo da tabela de transições do AC totalístico 777 para $k = 3$ e $r = 1$.

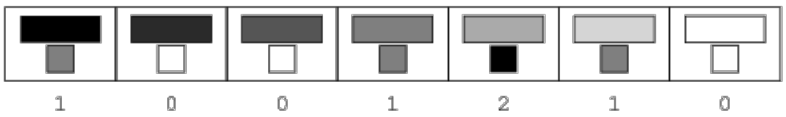

Figura 12: *Tabela de transições do AC totalístico 777 para $k = 3$ e $r = 1$. Cada transição depende apenas da soma (ou de forma análoga, da média) dos estados das células que compõem a vizinhança. Na figura, cada valor possível de média é representado por uma cor indo do branco (estado 0) ao preto (estado 2).*

Um AC semi-totalístico (*outer-totalistic* CA) é um AC cujas transições são definidas para o estado da célula do meio, junto com a soma (ou média) das células em volta. A quantidade de ACs semi-totalísticos de um determinado espaço definido por k e r é dada pela expressão $k^{k(1+(k-1)(2r))}$.

Um exemplo de AC semi-totalístico é o Jogo da Vida de (BERLEKAMP; CONWAY; GUY, 1982), que é definido em termos de quais células se encontram em estado "vivo" ou "morto", e quantos vizinhos estão "vivos" ou mortos. O jogo da vida é um exemplo de AC semi-totalístico que tem a propriedade de computabilidade universal.

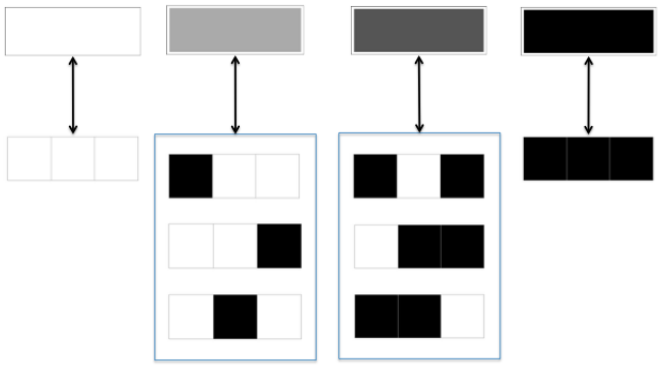

Figura 13: *Relação entre as vizinhanças de um AC totalístico e as vizinhanças de um AC comum.*

Um AC totalístico pode ser interpretado como um AC clássico que apresenta a propriedade da *totalidade*. De forma análoga, um AC semi-totalístico pode ser interpretado como um AC comum que apresenta a propriedade da *semi-totalidade*. A totalidade dita que todas as transições cuja soma dos estados das células é a mesma deve levar a um mesmo resultado. A semi-totalidade dita que toda transição cuja soma dos estados das células externas e o estado da célula central são iguais devem levar ao mesmo resultado.

A Figura 13 demonstra o relacionamento das vizinhanças estabelecido pela totalidade para os ACs elementares. A mesma ideia pode ser aplicada à semi-totalidade para obter as relações entre as vizinhanças.

4 TEMPLATES

Templates de autômatos celulares são uma generalização da representação em tabelas de transição de estados clássicas, que permitem que as transições tenham como resultado funções de uma ou mais variáveis. Como consequência, um *template* tem o poder de representar subespaços inteiros de ACs, ao invés de uma só regra.

Formalmente, um *template* que represente um subespaço contido em uma família dada por k e r é uma n-upla formada por k^n itens (sendo n o número de células na vizinhança), onde cada item i é uma função $g_i(x_0, x_1, \ldots, x_{k^n-1})$.

Por padrão, as variáveis livres x_i tem valores restritos ao intervalo discreto $[0, k-1]$. No entanto, templates aceitam expressões do tipo $x_i \in C$, onde C é um conjunto de valores cuja variável x_i pode assumir.

Cada template T representa um conjunto R_k, formado por cadeias cujo alfabeto é formado por números inteiros dentro do intervalo discreto $[0, k-1]$, interpretadas como tabelas de transições de ACs em formato k-ário.

Templates podem representar qualquer tipo de subconjunto de um espaço, incluindo o espaço inteiro. Um *template base* é simplesmente um template com variáveis livres em todas as suas transições, representando assim todas as regras do espaço. O template base do espaço elementar, por exemplo, seria:

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \tag{17}$$

Um *template constante* é um template que não tem variáveis livres, ou seja, é a própria tabela de transições em formato k-ário.

Como um exemplo simples, considere o template $T_1 = (0, 1 - x_1, 0, 1, x_3, 1, x_1, 0)$. Por ser uma 8-upla, T_1 representa o subespaço do espaço elementar formado por todas as regras que tem os bits fixos nas posições 0, 2, 4, 5 e 7 (contando da direita para a esquerda), variáveis livres nas posições 1 e 3, e que tenham o bit 6 sendo o complemento do bit 1.

O conjunto associado ao template T_1 seria $R_k = \{(0,1,0,1,0,1,0,0), (0,1,0,1,1,1,0,0), (0,0,0,1,0,1,1,0), (0,0,0,1,1,1,1,0)\}$, equivalente ao conjunto em formato decimal $\{84, 92,$

22, 30}, que é um subconjunto dos ACs elementares.

O processo de expansão, que será explicado em maiores detalhes na Seção 4.1, troca as variáveis livres de um template por todos os valores possíveis, e obtém o conjunto R_k que ele representa.

Todo template tem um número de substituições possíveis igual a k^m, sendo m o número de variáveis livres do template (sempre menor ou igual ao número de transições da tabela, dado por k^δ, como visto na Seção 2). Algumas dessas substituições, no entanto, podem dar origem a tabelas de transições k-árias inválidas que, quando descartadas, fazem com que o template consiga representar conjuntos de tamanhos diferentes de k^m, como é o caso do subconjunto de ACs conservativos de um dado espaço, que será abordado na Seção 5.1.

A aplicação da expansão no template base, por exemplo, resulta em $k^m = 2^8 = 256$ substituições, sendo cada uma equivalente a uma regra específica do espaço elementar original.

Cabe notar que os índices de cada variável do template base, quando convertidos para a base k, correspondem à vizinhança da transição. Desta forma, x_0 é equivalente à vizinhança $(0, 0, 0)$, x_1 é equivalente a $(0, 0, 1)$, e assim por diante. Esta propriedade estabelece um mapeamento direto entre as vizinhanças e os resultados das transições (representados pelas variáveis).

Todos os templates gerados pelos algoritmos estabelecidos na Seção 5 seguem este mapeamento como convenção para criação de variáveis livres. Desta maneira, a integridade do significado semântico de cada template é mantido tanto de maneira interna (como qualquer nome de variável já faria), quanto de maneira externa (entre dois ou mais templates). Em outras palavras, uma variável livre x_0 sempre se refere à vizinhança $(0, 0, 0)$, em qualquer template do espaço elementar.

Operações como a de interseção (apresentada na Seção 4.2), que operam em mais de um template ao mesmo tempo, fazem com que este tipo de integridade seja necessária.

A utilização do software *Wolfram Mathematica* (WOLFRAM RESEARCH, 2013) é de extrema valia na implementação dos algoritmos que operam com templates, por sua natureza funcional e simbólica, que remove qualquer barreira existente em outras lingua-

gens para tratar as variáveis como símbolos meta-programáveis e resolver as expressões algébricas em tempo de execução.

Em (BETEL; DE OLIVEIRA; FLOCCHINI, 2013), os autores encontraram analiticamente quais transições precisavam ser fixas, variáveis ou interdependentes em uma tabela de transições para que o AC tivesse chances de resolver o problema da paridade perfeitamente.

Ao fixar estas transições, restringiram o espaço de ACs unidimensionais binários de raio 2, composto por 4.294.967.296 regras, para apenas 64 candidatos, que falharam em testes posteriores. Apesar de terem usado uma estrutura conhecida como grafo de De Bruijn como principal forma de representação deste subespaço, poderiam facilmente ter utilizado os *templates* aqui propostos.

Os grafos mostrados na Figura 14, retirados de (BETEL; DE OLIVEIRA; FLOCCHINI, 2013), foram utilizados no trabalho para representar dois subespaços dos ACs binários unidimensionais de raio 2 (com 32 ACs cada), que teriam alguma chance de resolver o problema da paridade perfeitamente. De acordo com (BETEL; DE OLIVEIRA; FLOCCHINI, 2013), a, b, c e d são variáveis livres, e x e y são variáveis cujos valores devem ser opostos.

O primeiro grafo poderia ser representado pelo template 18, e segundo, pelo template 19.

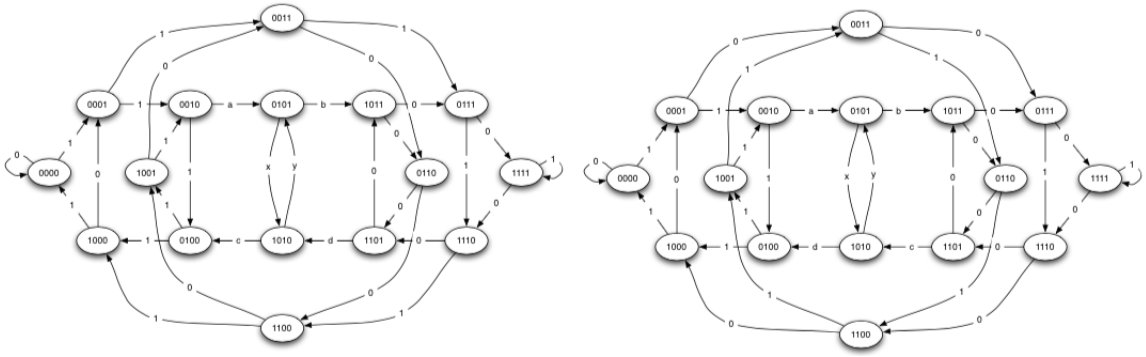

Figura 14: *Grafos de De Bruijn representantes de conjuntos de ACs unidimensionais binários de raio 2 candidatos à resolução do problema da paridade. Retirado de (BETEL; DE OLIVEIRA; FLOCCHINI, 2013).*

$$T_{paridade1} = (0, 1, 1, 1, 1, x_{26}, 0, 1, 1, 1, 1 - x_{10}, x_{20}, 0, 0, 1, 0, 1, 0, 1, 0, x_{11}, x_{10}, 0, 0, 1, 0, x_5, 0, 1, 0, 0, 1) \tag{18}$$

$$T_{paridade2} = (0, 1, 1, 0, 1, x_{26}, 1, 0, 1, 1, 1 - x_{10}, x_{20}, 0, 1, 1, 0, 1, 0, 1, 1, x_{11}, x_{10}, 0, 0, 1, 0, x_5, 0, 0, 0, 0, 1) \tag{19}$$

4.1 EXPANSÃO DE TEMPLATES

Todo template é associado a um conjunto de tabelas de transição em formato k-ário R_k, obtido através do processo de expansão de templates, como mostra a expressão 20. A expansão foi implementada como um algoritmo na linguagem do software *Wolfram Mathematica* (WOLFRAM RESEARCH, 2013), e é descrita aqui em maiores detalhes.

$$E(T) = R_k \tag{20}$$

A expansão é, por padrão, uma operação custosa, já que é responsável pela transformação da representação compacta dada por um template em uma representação expandida, dada por um conjunto.

A i-ésima expansão de um template, denotada como E_i para $0 <= i <= k^m - 1$, inicia com a extração de suas variáveis. Em seguida, i é convertido em um número na base k, e completado com zeros à esquerda, de maneira a obter uma quantidade de dígitos de tamanho m. Cada um dos dígitos é então atribuído a uma variável, gerando a i-ésima expansão do template, que pode ou não ser válida no contexto de uma tabela de transições k-ária.

Definido o processo para obter a i-ésima expansão, basta aplicá-lo a todo i no intervalo discreto $[0, k^m–1]$, e o resultado será o conjunto R_k associado ao template.

Considere o exemplo do template apresentado na Seção 4: $T_1 = (0, 1 - x_1, 0, 1, x_3, 1, x_1, 0)$, com $k = 2$. Para obter o conjunto R_k, a i-ésima expansão deve ser feita para $i \in \{0, 1, 2, 3\}$. No caso de $i = 2$, o algoritmo de expansão faz a conversão de 2 para seu equivalente binário $(1, 0)$ e atribui cada dígito a uma variável, obtendo $\{x_1 = 0, x_3 = 1\}$. Em seguida, transforma as variáveis do template em seus respectivos valores, obtendo:

$$E_2(T_1) = (0, 1 - 0, 0, 1, 1, 1, 1, 0) = (0, 1, 0, 1, 1, 1, 1, 0) \tag{21}$$

Repetindo o processo para cada valor possível de i, obtém-se conjunto $R_k = \{(0, 1, 0, 1, 0, 1, 0, 0), (0, 1, 0, 1, 1, 1, 0, 0), (0, 0, 0, 1, 0, 1, 1, 0), (0, 0, 0, 1, 1, 1, 1, 0)\}$.

Alguns templates podem gerar expansões inválidas. Considere $T_2 = (0, 0, 0, 0, 2 -$

$x_3, 1, 1, 1)$, com $k = 2$. A expansão associada a $i = 0$ seria:

$$E_0(T_2) = (0, 0, 0, 0, 2 - 0, 1, 1, 1) = (0, 0, 0, 0, 2, 1, 1, 1) \tag{22}$$

No entanto, como $k = 2$, as transições devem estar no intervalo discreto $[0, 1]$, fazendo com que esta seja uma expansão inválida. Esta expansão é descartada, e o conjunto R_k associado a T_2 é $R_k = \{(0, 0, 0, 0, 1, 1, 1, 1)\}$.

Outro caso onde podem ocorrer expansões inválidas acontece quando uma variável é restrita por um conjunto C. Considere o template $T_3 = (0, 0, 0, x_3 \in \{0, 2\}, 1, 1, 1, 1)$, com $k = 3$. A substituição obtida quando $i = 1$ seria:

$$E_1(T_3) = (0, 0, 0, 0, 1, 1, 1, 1) \tag{23}$$

No entanto, a expressão $x_3 \in \{0, 2\}$ deixa explícito que a variável x_3 só pode representar os valores 0 ou 2. Neste caso, a expansão associada a $i = 1$ é descartada, e o conjunto R_k obtido é $R_k = \{(0, 0, 0, 0, 0, 1, 1, 1), ((0, 0, 0, 0, 2, 1, 1, 1)\}$.

A existência de regras inválidas é importante pois permite que templates representem conjuntos com tamanhos menores que k^m.

Em alguns casos, como nos templates para regras conservativas (vistos na Seção 5.1), a obtenção de tabelas inválidas através de expressões que ultrapassam os intervalos permitidos é crucial para que condições nas tabelas de transições sejam atingidas e o os templates possam representar conjuntos de regras conservativas, cujos tamanhos nem sempre são potências de k.

Em outros, como nos templates para regras confinadas (vistos na Seção 5.4), as restrições de valores em variáveis dos templates por meio de expressões do tipo $x_i \in C$ são cruciais para conseguir representar transições que só podem ter alguns dos valores do intervalo permitido por padrão (principalmente nos casos em que $k > 2$).

4.2 INTERSECÇÃO ENTRE TEMPLATES

A interseção é uma operação que recebe dois templates definidos para o mesmo espaço, ou seja, que tenham os mesmos valores para k e r, e encontra um template cujo conjunto R_k seja equivalente à interseção entre os conjuntos R_k associados aos templates recebidos como entrada.

Formalmente, a interseção de templates é dada da seguinte maneira:

$$I(T_1, T_2) = T_3 \iff E(T_3) = E(T_1) \cap E(T_2) \tag{24}$$

O processo de interseção é realizado em duas etapas, sendo a segunda necessária apenas para templates que contenham expressões do tipo $x_i \in C$.

A primeira etapa consiste em igualar os templates recebidos como entrada, obtendo um sistema de equações que, quando resolvido automaticamente utilizando a função Solve (parte integral da *Wolfram Language* (WOLFRAM RESEARCH, 2013)), resulta nos relacionamentos entre as variáveis. Estes relacionamentos, quando aplicados a qualquer dos templates recebidos como entrada, resultam na interseção propriamente dita.

Caso o sistema não tenha solução, os templates dados como entrada representam conjuntos R_k disjuntos, e, por definição, não se intersectam.

Como exemplo, considere os templates $T_1 = (x_7, 0, x_5, 0, 1, 0, x_2, x_0)$ e $T_2 = (1, 0, 1–x_1, 1–x_3, x_3, 0, x_1, 0)$, ambos com $k = 2$ e $r = 1$. Primeiramente, ambos os templates são igualados, gerando o sistema:

$$\begin{cases} x_7 &= 1 \\ 0 &= 0 \\ x_5 &= 1–x_1 \\ 0 &= 1–x_3 \\ 1 &= x_3 \\ 0 &= 0 \\ x_1 &= x_1 \\ x_0 &= 0 \end{cases} \tag{25}$$

Quando resolvido pela função Solve, o conjunto solução S obtido é $S = \{x_0 = 0, x_3 = 1, x_2 = x_1, x_5 = 1–x_1, x_7 = 1\}$. Este conjunto é aplicado como um conjunto de substituições sobre os templates recebidos como entrada.

Em casos onde não existem restrições nas variáveis, ambas as substituições resultam em templates iguais, e qualquer um pode ser escolhido como o resultado $T_3 = (1, 0, 1–x_1, 0, 1, 0, x_1, 0)$, finalizando o processo.

Considere agora uma variação dos templates T_1 e T_2 que contém restrições nas variáveis $T_{restrito1} = (x_7, 0, x_5, 0, 1, 0, x_2 \in \{0, 1\}, x_0)$ e $T_{restrito2} = (1, 0, 1–x_1, 1–x_3, x_3, 0, x_1 \in \{0\}, 0)$.

Neste caso, o processo de interseção seguiria normalmente pela primeira etapa, obtendo o conjunto solução S do sistema de Equações 25, e aplicando estas substituições nos templates recebidos como entrada.

Neste caso, porém, os templates obtidos não seriam iguais, já que as expressões de restrição de variável ainda não foram tratadas. O conjunto obtido seria: $\{(1, 0, 1–x_1, 0, 1, 0, x_1 \in \{0, 1\}, 0), (1, 0, 1–x_1, 0, 1, 0, x_1 \in \{0\}, 0)\}$.

A segunda etapa do algoritmo consiste em extrair as expressões que estabelecem as restrições, e traduzi-las em um segundo sistema de equações que, quando resolvido, indica quais valores cada variável restrita pode ter. Novamente, caso o sistema não tenha solução, os templates são considerados disjuntos, portanto, sua interseção não existe.

Assim, o algoritmo segue extraindo as expressões que restringem as variáveis, obtendo o conjunto: $\{x_1 \in \{0, 1\}, x_1 \in \{0\}$. Este conjunto é traduzido para o sistema de equações:

$$\begin{cases} x_1 &= 1 \vee x_1 = 0 \\ x_1 &= 0 \end{cases} \tag{26}$$

O conjunto solução S para este sistema é $S = \{x = 0\}$. O algoritmo então remove as expressões de restrições antigas para a variável x_1, transformando-a no valor 0, gerando assim o resultado da interseção: $T_{restrito3} = (1, 0, 1, 0, 1, 0, 0, 0)$.

Apesar de ao final da segunda etapa do processo a variável x_1 ter apenas um valor válido, este nem sempre é o caso. O algoritmo está preparado para converter uma quantidade maior de valores em expressões de restrição de variável compatíveis com a sintaxe estabelecida na representação ($x_i \in C$).

Cabe aqui chamar atenção ao fato de que os templates $T_{restrito1}$ e $T_{restrito2}$ poderiam ter sido reescritos de maneira a não utilizar expressões de restrição de variáveis, como segue: $T_{restrito1} = (x_7, 0, x_5, 0, 1, 0, x_2, x_0)$ e $T_{restrito2} = (1, 0, 1, 1–x_3, x_3, 0, 0, 0)$.

Na verdade, qualquer template representante de espaços binários pode ser escrito sem utilizar restrições nas variáveis, já que as únicas possibilidades são valores específicos (0 ou 1), ou ambos (variáveis livres).

A escolha de utilizar templates binários neste momento representa uma maneira didática para explicar a segunda etapa do algoritmo sem precisar tratar de templates com $k > 2$, uma vez que o processo continuaria igual, mas aplicado a templates muito maiores.

5 ALGORITMOS DE GERAÇÃO DE TEMPLATES

O tamanho das famílias de ACs cresce de maneira exponencial conforme os parâmetros k e r aumentam. Muitas vezes, no entanto, deseja-se explorar apenas um subespaço de determinada família, cujos itens compartilhem de determinadas propriedades.

Em alguns casos, o intuito pode ser simplesmente estabelecer uma subclasse de ACs para explorar quais tipos de comportamento surgem e deixam de existir, quais tipos de problemas antes indecidíveis se tornam decidíveis, ou simplesmente como uma maneira de explorar um espaço muito grande sem precisar considerar todos seus integrantes.

Esta é a motivação por trás de trabalhos feitos com regras totalísticas em (WOLFRAM, 2002), por exemplo. Também é a motivação da exploração de regras confinadas em (THEYSSIER, 2004) e (THEYSSIER, 2005).

Em outros casos, uma propriedade pode ser utilizada como indicativo de que um AC tem um determinado comportamento, como ocorre em (WOLZ; DE OLIVEIRA, 2008) e (KARI; LE GLOANNEC, 2012), onde a simetria interna de uma regra serve como indicativo para seu poder de resolução da DCT.

É evidente que a existência de maneiras para obter subespaços de ACs sem precisar enumerar o espaço inteiro é extremamente valiosa para qualquer estudo que deseje focar apenas em um determinado subespaço. O framework dos templates é proposto no presente trabalho como uma maneira de atingir este objetivo.

Parte integrante do framework proposto no presente projeto é formada pelos algoritmos de geração de templates. Estes algoritmos automatizam a geração de templates representes de um subconjunto de um espaço definido por k e r, cujos itens do conjunto R_k correspondente compartilham de uma determinada propriedade estática.

Servem, portanto, como exemplos palpáveis de como templates para diversas propriedades podem ser gerados, provando sua capacidade de representação e seu valor como ferramenta.

As próximas subseções demonstram todos os algoritmos de geração de templates desenvolvidos durante o presente trabalho, baseados nas propriedades descritas na Seção 3.

5.1 TEMPLATES PARA REGRAS CONSERVATIVAS

Em (BOCCARA; FUKŚ, 2002), são estabelecidas relações que devem ser mantidas nos resultados de uma tabela de transições para que um AC seja conservativo. Pareando esta ideia com o conceito de templates, infere-se que é possível aplicar as mesmas relações nas variáveis de um template base para obter o conjunto de regras conservativas de um determinado espaço. O algoritmo de geração de templates para regras conservativas funciona alinhado a esta ideia.

O algoritmo recebe como entrada os valores de k e r que definem o espaço cujas regras conservativas serão geradas. Como ponto de partida, são geradas todas as vizinhanças do espaço em questão, e descartadas aquelas que geram tautologias para as condições de (BOCCARA; FUKŚ, 2002) de acordo com (SCHRANKO; DE OLIVEIRA, 2010).

Em seguida, as condições são aplicadas às transições restantes, gerando um conjunto de substituições que podem então ser aplicadas ao template base do espaço, dando origem ao resultado desejado.

Como exemplo do funcionamento do algoritmo, vamos utilizá-lo para encontrar as regras conservativas do espaço elementar (onde $k = 2$ e $r = 1$).

Primeiro, é obtido o conjunto das vizinhanças relevantes deste espaço, isto é, todas as não iniciadas pelo estado zero, acrescidas da vizinhança homogênea de zeros: $\{(0,0,0),(1,1,1),(1,1,0),(1,0,1),(1,0,0)\}$

Para cada uma dessas vizinhanças, uma equação é criada, usando como base as condições de Boccara e FukŚ (2002), que pode ser vista na Equação (5).

$$\begin{cases} x_0 &= 0 \\ x_7 &= 1 \\ x_6 &= 1 + x_2 - x_3 \\ x_5 &= 1 + x_0 - x_2 \\ x_4 &= 1 + 2x_0 - x_1 - x_2 \end{cases} \tag{27}$$

O algoritmo então usa a função Solve do Wolfram Mathematica (WOLFRAM RESE-

ARCH, 2013) para simplificar o sistema, e encontra o conjunto solução a seguir:

$$\{x_0 = 0, x_4 = 1 - x_1 - x_2, x_5 = 1 - x_2, x_6 = 1 + x_2 - x_3, x_7 = 1\} \tag{28}$$

Esta solução é então traduzida em regras de substituição, que, quando aplicadas ao template base, geram como resultado o template a seguir:

$$T_{conservativo} = (1, 1 + x_3 - x_4, 1 - x_3, 1 - x_2 - x_3, x_4, x_3, x_2, 0) \tag{29}$$

Quando expandido, $T_{conservativo}$ resulta no seguinte conjunto de regras em representação k-ária: $\{(1,0,1,1,1,0,0,0), (1,2,0,0,0,1,0,0), (1,1,0,0,1,1,0,0), (1,1,1,0,0,0,1,0), (1,0,1,0,1,0,1,0), (1,2,0,-1,0,1,1,0), (1,1,0,-1,1,1,1,0), (1,1,1,1,0,0,0,0)\}$.

É evidente que algumas destas regras não são válidas, pelo fato de existirem números fora do intervalo discreto $[0, k-1]$. Após descartar as regras inválidas, o conjunto se torna: $\{(1,0,1,1,1,0,0,0), (1,1,0,0,1,1,0,0), (1,1,1,0,0,0,1,0), (1,0,1,0,1,0,1,0), (1,1,1,1,0,0,0,0,0)\}$, equivalente aos ACs $\{184, 204, 226, 170, 240\}$, que são exatamente os ACs elementares conhecidos por apresentar conservabilidade de estados.

Como um segundo exemplo, vamos aplicar o algoritmo no espaço de ACs unidimensionais binários de raio 2.

Cada vizinhança deste espaço contém $2r + 1 = 5$ células, e para cada vizinhança a tabela de transições pode ter dois possíveis resultados. Assim, sabemos que a tabela de transições tem $k^{2r+1} = 2^5 = 32$ itens, e que o espaço é composto por $2^{32} = 4.294.967.296$ ACs. Desta maneira, o template base deste espaço contém 32 variáveis, como segue:

$$\begin{aligned}\{&x_{31}, x_{30}, x_{29}, x_{28}, x_{27}, x_{26}, x_{25}, x_{24}, x_{23}, x_{22}, x_{21}, x_{20}, x_{19}, x_{18}, x_{17},\\ &x_{16}, x_{15}, x_{14}, x_{13}, x_{12}, x_{11}, x_{10}, x_9, x_8, x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0\}\end{aligned} \tag{30}$$

Então, é criado o conjunto das vizinhanças relevantes do espaço: $\{(0,0,0,0,0), (0,0,0,0,1), (0,0,0,1,1), (0,0,1,0,1), (0,0,1,1,1), (0,1,0,0,1), (0,1,0,1,1), (0,1,1,0,1), (0,1,1,1,1), (1,0,0,0,1), (1,0,0,1,1), (1,0,1,0,1), (1,0,1,1,1), (1,1,0,0,1), (1,1,0,1,1), (1,1,1,0,1), (1,1,1,1,1)\}$, e para cada transição associada a uma destas vizinhanças, é

estabelecida a relação definida na Equação (5). Assim, é gerado um sistema formado por 17 equações:

$$
\begin{cases}
x_0 &= 0 \\
x_{31} &= 1 \\
x_{30} &= 1 + x_{14} - x_{15} \\
x_{29} &= 1 + x_{13} - x_{14} + x_6 - x_7 \\
x_{28} &= 1 + x_{12} - x_{14} + x_6 - x_7 \\
x_{27} &= 1 + x_{11} - x_{13} + x_2 - x_3 + x_5 - x_6 \\
x_{26} &= 1 + x_{10} - x_{13} + x_2 - x_3 + x_5 - x_6 \\
x_{25} &= 1 + x_9 - x_{12} + x_2 - x_3 + x_5 - x_6 \\
x_{24} &= 1 - x_{12} + x_2 - x_3 + x_4 - x_6 + x_8 \\
x_{23} &= 1 + x_0 - x_{11} - x_2 + x_3 - x_5 + x_7 \\
x_{22} &= 1 + x_0 - x_{11} - x_2 + x_3 - x_5 + x_6 \\
x_{21} &= 1 + x_0 - x_{10} \\
x_{20} &= 1 + x_0 - x_{10} + x_4 - x_5 \\
x_{19} &= 1 + 2x_0 - x_9 - x_2 + x_3 - x_4 \\
x_{18} &= 1 + 2x_0 - x_9 - x_4 \\
x_{17} &= 1 + 3x_0 - x_2 - x_4 - x_8 \\
x_{16} &= 1 + 4x_0 - x_1 - x_2 - x_4 - x_8
\end{cases}
\tag{31}
$$

Este sistema é simplificado usando a função Solve novamente, gerando o conjunto solução:

$$
\begin{aligned}
&\{\\
&x_0 = 0,\\
&x_{16} = 1 - x_1 - x_2 - x_4 - x_8,\\
&x_{17} = 1 - x_2 - x_4 - x_8,\\
&x_{18} = 1 - x_4 - x_9,\\
&x_{19} = 1 - x_2 + x_3 - x_4 - x_9,\\
&x_{20} = 1 - x_{10} + x_4 - x_5,\\
&x_{22} = 1 - x_{11} - x_2 + x_3 - x_5 + x_6,\\
&x_{21} = 1 - x_{10},\\
&x_{23} = 1 - x_{11} - x_2 + x_3 - x_5 + x_7,\\
&x_{24} = 1 - x_{12} + x_2 - x_3 + x_4 - x_6 + x_8,\\
&x_{25} = 1 - x_{12} + x_2 - x_3 + x_4 - x_6 + x_9,\\
&x_{26} = 1 + x_{10} - x_{13} + x_2 - x_3 + x_5 - x_6,\\
&x_{27} = 1 + x_{11} - x_{13} + x_2 - x_3 + x_5 - x_6,\\
&x_{28} = 1 + x_{12} - x_{14} + x_6 - x_7,\\
&x_{29} = 1 + x_{13} - x_{14} + x_6 - x_7,\\
&x_{30} = 1 + x_{14} - x_{15},\\
&x_{31} = 1\\
&\}
\end{aligned}
$$

Quando aplicadas ao template base, as substituições dão origem ao template caracterizado na Tabela 1.

Vizinhança	Valor no Template
$(1,1,1,1,1)$	1
$(1,1,1,1,0)$	$1 + x_{14} - x_{15}$
$(1,1,1,0,1)$	$1 + x_{13} - x_{14} + x_6 - x_7$
$(1,1,1,0,0)$	$1 + x_{12} - x_{14} + x_6 - x_7$
$(1,1,0,1,1)$	$1 + x_{11} - x_{13} + x_2 - x_3 + x_5 - x_6$
$(1,1,0,1,0)$	$1 + x_{10} - x_{13} + x_2 - x_3 + x_5 - x_6$

$(1,1,0,0,1)$	$1 - x_{12} + x_2 - x_3 + x_4 - x_6 + x_9$
$(1,1,0,0,0)$	$1 - x_{12} + x_2 - x_3 + x_4 - x_6 + x_8$
$(1,0,1,1,1)$	$1 - x_{11} - x_2 + x_3 - x_5 + x_7$
$(1,0,1,1,0)$	$1 - x_{11} - x_2 + x_3 - x_5 + x_6$
$(1,0,1,0,1)$	$1 - x_{10}$
$(1,0,1,0,0)$	$1 - x_{10} + x_4 - x_5$
$(1,0,0,1,1)$	$1 - x_2 + x_3 - x_4 - x_9$
$(1,0,0,1,0)$	$1 - x_4 - x_9$
$(1,0,0,0,1)$	$1 - x_2 - x_4 - x_8$
$(1,0,0,0,0)$	$1 - x_1 - x_2 - x_4 - x_8$
$(0,1,1,1,1)$	x_{15}
$(0,1,1,1,0)$	x_{14}
$(0,1,1,0,1)$	x_{13}
$(0,1,1,0,0)$	x_{12}
$(0,1,0,1,1)$	x_{11}
$(0,1,0,1,0)$	x_{10}
$(0,1,0,0,1)$	x_9
$(0,1,0,0,0)$	x_8
$(0,0,1,1,1)$	x_7
$(0,0,1,1,0)$	x_6
$(0,0,1,0,1)$	x_5
$(0,0,1,0,0)$	x_4
$(0,0,0,1,1)$	x_3
$(0,0,0,1,0)$	x_2
$(0,0,0,0,1)$	x_1
$(0,0,0,0,0)$	0

Tabela 1: *Template* para as regras conservativas do espaço de ACs unidimensionais binários de raio 2.

Cabe aqui chamar a atenção para um fato que fica ainda mais evidente neste último template. As variáveis correspondentes às vizinhanças descartadas no começo do processo

se tornam livres no template final. A interpretação desta ocorrência é a de que para uma regra conservativa, não importa qual o resultado destas transições, contanto que as transições restantes tenham suas dependências corretamente estabelecidas.

O template obtido na Tabela 1, por exemplo, contém 15 variáveis livres. Sendo assim, este template tem $k^m = 2^{15} = 32.768$ possíveis substituições, das quais apenas 418 são válidas.

No caso do raio 3, o template encontrado pelo algoritmo possui 63 variáveis, caracterizando 2^{63} substituições, número este que inviabiliza completamente a realização da sua expansão. Por este motivo, foi impossível verificar quantas regras inválidas foram encontradas pelo algoritmo. Para o raio 4, foi encontrado um template com 255 variáveis, totalizando 2^{255} substituições.

5.2 TEMPLATES PARA REGRAS COM VALORES ARBITRÁRIOS DE SIMETRIA INTERNA

As operações de conjugação, reflexão e composição das anteriores estabelecem relações entre as vizinhanças de um AC que indicam quais resultados de uma regra de transições devem ser iguais ou diferentes para que um AC tenha um determinado valor de simetria interna, conforme explicado na Seção 3.2.

Estas relações podem ser exploradas para criar um algoritmo gerador de conjuntos de templates que representam todas as regras que compartilham de um determinado valor de simetria interna.

O algoritmo descrito nesta seção gera templates representantes de regras com valores arbitrários de simetria para espaços binários.

O algoritmo recebe três entradas: o valor de r que define um espaço binário, uma das três transformações de simetria, e o valor de simetria interna ϕ para o qual se deseja encontrar o conjunto de templates. Como exemplo, o algoritmo será aplicado para encontrar todas as regras do espaço elementar com valor de simetria interna igual a 6, de acordo com a reflexão.

O primeiro passo do algoritmo é gerar todas as vizinhanças do espaço, obtendo o

conjunto:

$$\begin{aligned}\{&(1,1,1),(1,1,0),(1,0,1),(1,0,0),\\&(0,1,1),(0,1,0),(0,0,1),(0,0,0)\}\end{aligned} \tag{32}$$

Em seguida, a transformação é aplicada para cada vizinhança, gerando um conjunto de pares. Continuando o exemplo, a reflexão é aplicada a cada vizinhança, gerando o conjunto:

$$\begin{aligned}\{&((1,1,1),(1,1,1)),((1,1,0),(0,1,1)),((1,0,1),(1,0,1)),((1,0,0),(0,0,1)),\\&((0,1,1),(1,1,0)),((0,1,0),(0,1,0)),((0,0,1),(1,0,0)),((0,0,0),(0,0,0))\}\end{aligned} \tag{33}$$

Destes oito pares, a quantidade γ de vizinhanças invariantes à transformação é contada. Estas vizinhanças são então removidas do conjunto, obtendo:

$$\{((1,1,0),(0,1,1)),((1,0,0),(0,0,1)),((0,1,1),(1,1,0)),((0,0,1),(1,0,0))\} \tag{34}$$

Caso o valor de ϕ seja menor do que o número de vizinhanças invariantes γ, o algoritmo retorna um conjunto vazio, uma vez que o valor mínimo de simetria interna é igual à quantidade de vizinhanças invariantes a transformação, como visto na Seção 3.2.

As transformações estabelecem relações bidirecionais entre as vizinhanças, portanto pares formados pelas mesmas vizinhanças em ordem diferente são considerados repetições. Uma vez que as relações são bidirecionais, isso sempre acontecerá com metade do conjunto. O algoritmo remove as repetições do conjunto, gerando para o exemplo:

$$\{((1,1,0),(0,1,1)),((1,0,0),(0,0,1))\} \tag{35}$$

Cada vizinhança é então transformada na variável de template correspondente, e a transformação é aplicada ao menor item do par, gerando:

$$C_{eq} = \{(x_6, x_3), (x_4, x_1)\} \tag{36}$$

O conjunto C_{eq} representa as equivalências entre variáveis de um template de acordo com a reflexão. Caso procurássemos um template representante de regras com simetria máxima, todas as equivalências do conjunto deveriam ser verdadeiras, como no sistema a seguir:

$$\begin{cases} x_4 &= x_1 \\ x_6 &= x_3 \end{cases} \tag{37}$$

No entanto, para valores arbitrários de simetria, como no exemplo proposto, este sistema deve ser particionado de maneira que algumas das equivalências sejam verdadeiras e outras não, gerando:

$$\begin{cases} x_4 &= x_1 \\ x_6 &\neq x_3 \end{cases} \tag{38}$$

$$\begin{cases} x_4 &\neq x_1 \\ x_6 &= x_3 \end{cases} \tag{39}$$

A relação de desigualdade é representada num template por meio de uma função de uma variável que sempre dê um resultado diferente. No caso binário, isto é atingido subtraindo 1 da variável. Portanto, na verdade, os sistemas propostos são equivalentes a:

$$\begin{cases} x_4 &= x_1 \\ x_6 &= 1 - x_3 \end{cases} \tag{40}$$

$$\begin{cases} x_4 &= 1 - x_1 \\ x_6 &= x_3 \end{cases} \tag{41}$$

Para montar estas equivalências, o algoritmo particiona o conjunto C_{eq} em $(\phi - \gamma)/2$, partições, e une essas partições com seus respectivos complementos com relação a C_{eq}, já adicionando as desigualdades nos pares do complemento. Para o exemplo proposto, o resultado é:

$$C_{trans} = \{((x_4, x_1), (x_6, 1 - x_3)), ((x_4, 1 - x_1), (x_6, x_3))\} \tag{42}$$

O conjunto C_{trans} representa todas as transformações que, quando aplicadas ao template base, darão origem aos templates representantes de regras com simetria interna por reflexão igual a 6. Ao realizar estas transformações, obtemos o conjunto de templates:

$$\{(x_7, 1 - x_3, x_5, x_1, x_3, x_2, x_1, x_0), (x_7, x_3, x_5, 1 - x_1, x_3, x_2, x_1, x_0)\} \tag{43}$$

Quando expandido, este conjunto de templates representa exatamente as regras com valor de simetria interna por reflexão igual a 6 do espaço elementar.

Uma versão inicial do algoritmo apresentado nesta seção, que encontrava templates com simetria interna máxima de um espaço, foi descrito em (DE OLIVEIRA; VERARDO, 2014a) e (DE OLIVEIRA; VERARDO, 2014b). Esta versão inicial podia receber como entrada qualquer composição das três transformações, e usava um processo equivalente à interseção de templates para gerar um template representante desta combinação. No entanto, a extração da funcionalidade de interseção para um algoritmo próprio fez com que esta etapa não seja necessária durante a geração do template, uma vez que na nova versão o usuário pode gerar o conjunto da transformação desejada e depois realizar a interseção usando o algoritmo implementado conforme explicado na Seção 4.2.

5.3 TEMPLATES PARA REGRAS INVARIANTES A TROCAS DE COR

O algoritmo para obtenção de regras invariantes a cor é semelhante ao descrito na Seção 5.2. Na verdade, como notado na Seção 3.3, as regras com valor máximo de simetria interna por conjugação são exatamente as regras invariantes a cor de um espaço. A grande diferença entre o processo em questão e o descrito nesta seção é que este usa especificamente a conjugação em sua versão generalizada para espaços não binários.

O algoritmo recebe como entrada os valores de k e r que determinam o espaço para o qual se deseja encontrar os ACs invariantes a cor. Vamos aplicar o algoritmo ao espaço de $k = 3$ e $r = 1$ para exemplificar seu funcionamento.

Como primeiro passo, o algoritmo gera o conjunto C_ρ de todas as permutações possíveis

do espaço:

$$C_\rho = \{\{0 \to 0, 1 \to 2, 2 \to 1\}, \{0 \to 1, 1 \to 0, 2 \to 2\}, \{0 \to 1, 1 \to 2, 2 \to 0\}, \\ \{0 \to 2, 1 \to 0, 2 \to 1\}, \{0 \to 2, 1 \to 1, 2 \to 0\}\} \tag{44}$$

Para uma permutação ρ do conjunto C_ρ, o algoritmo inicia seu processo gerando o conjunto de todas as vizinhanças do espaço, e encontrando as relações unidirecionais entre elas. Particiona, então, o conjunto de vizinhanças de acordo com estes relacionamentos. O resultado deste processo para o caso da permutação $\rho = \{0 \to 1, 1 \to 2, 2 \to 0\}$ é:

$$\{\{(0,2,2),(1,0,0),(2,1,1)\},\{(0,2,1),(1,0,2),(2,1,0)\},\{(0,2,0),(1,0,1),(2,1,2)\}, \\ \{(0,1,2),(1,2,0),(2,0,1)\},\{(0,1,1),(1,2,2),(2,0,0)\},\{(0,1,0),(1,2,1),(2,0,2)\}, \\ \{(0,0,2),(1,1,0),(2,2,1)\},\{(0,0,1),(1,1,2),(2,2,0)\},\{(0,0,0),(1,1,1),(2,2,2)\}\} \tag{45}$$

Cada vizinhança deste conjunto é convertida para sua representação em variáveis de template, representantes das saídas das transições correspondentes às vizinhanças:

$$\{\{x_8, x_9, x_{22}\}, \{x_7, x_{11}, x_{21}\}, \{x_6, x_{10}, x_{23}\}, \\ \{x_5, x_{15}, x_{19}\}, \{x_4, x_{17}, x_{18}\}, \{x_3, x_{16}, x_{20}\}, \\ \{x_2, x_{12}, x_{25}\}, \{x_1, x_{14}, x_{24}\}, \{x_0, x_{13}, x_{26}\}\} \tag{46}$$

Cada item do conjunto obtido representa uma relação unidirecional estabelecida pela permutação ρ, em que cada variável pode ser reescrita como uma função da anterior. Supondo que exista uma função algébrica f_ρ que represente a permutação ρ, o algoritmo

gera, para cada partição, as relações:

$$
\begin{aligned}
&\{\{x_8 = x_8, x_9 = f_\rho(x_8), x_{22} = f_\rho(f_\rho(x_8))\},\\
&\{x_7 = x_7, x_{11} = f_\rho(x_7), x_{21} = f_\rho(f_\rho(x_7))\},\\
&\{x_6 = x_6, x_{10} = f_\rho(x_6), x_{23} = f_\rho(f_\rho(x_6)\},\\
&\{x_5 = x_5, x_{15} = f_\rho(x_5), x_{19} = f_\rho(f_\rho(x_5))\},\\
&\{x_4 = x_4, x_{17} = f_\rho(x_4), x_{18} = f_\rho(f_\rho(x_4))\},\\
&\{x_3 = x_3, x_{16} = f_\rho(x_3), x_{20} = f_\rho(f_\rho(x_3))\},\\
&\{x_2 = x_2, x_{12} = f_\rho(x_2), x_{25} = f_\rho(f_\rho(x_2))\},\\
&\{x_1 = x_1, x_{14} = f_\rho(x_1), x_{24} = f_\rho(f_\rho(x_1))\},\\
&\{x_0 = x_0, x_{13} = f_\rho(x_0), x_{26} = f_\rho(f_\rho(x_0))\}\}
\end{aligned}
\tag{47}
$$

A função f_ρ é qualquer função que gere os mesmos resultados que a permutação. Para gerar esta função automaticamente, o algoritmo utiliza a função InterpolatingPolynomial do Wolfram Mathematica (WOLFRAM RESEARCH, 2013). A função InterpolatingPolynomial recebe como entrada um conjunto de pontos, e gera um polinômio que passa por estes pontos. Para a permutação $\rho = \{0 \to 1, 1 \to 2, 2 \to 0\}$ em questão, que representa três pares de pontos a serem passados para a função, o polinômio encontrado é $f_\rho(x) = 1/2 * (2 + 5x - 3x^2)$.

Estas equivalências são aplicadas então ao template base, gerando um template representante de todas as regras que são invariantes a troca de cores de acordo com a permutação ρ:

$$
\begin{aligned}
T_\rho = (\\
&1/2 * (2 + 5/2(2 + 5x_0 - 3x_0^2) - 3/4(2 + 5x_0 - 3x_0^2)^2),\\
&1/2 * (2 + 5/2(2 + 5x_2 - 3x_2^2) - 3/4(2 + 5x_2 - 3x_2^2)^2),\\
&1/2 * (2 + 5/2(2 + 5x_1 - 3x_1^2) - 3/4(2 + 5x_1 - 3x_1^2)^2),\\
&1/2 * (2 + 5/2(2 + 5x_6 - 3x_6^2) - 3/4(2 + 5x_6 - 3x_6^2)^2),\\
&1/2 * (2 + 5/2(2 + 5x_8 - 3x_8^2) - 3/4(2 + 5x_8 - 3x_8^2)^2),\\
&1/2 * (2 + 5/2(2 + 5x_7 - 3x_7^2) - 3/4(2 + 5x_7 - 3x_7^2)^2),\\
&1/2 * (2 + 5/2(2 + 5x_3 - 3x_3^2) - 3/4(2 + 5x_3 - 3x_3^2)^2),\\
&1/2 * (2 + 5/2(2 + 5x_5 - 3x_5^2) - 3/4(2 + 5x_5 - 3x_5^2)^2),\\
&1/2 * (2 + 5/2(2 + 5x_4 - 3x_4^2) - 3/4(2 + 5x_4 - 3x_4^2)^2),\\
&1/2 * (2 + 5x_4 - 3x_4^2),\\
&1/2(2 + 5x_3 - 3x_3^2),\\
&1/2 * (2 + 5x_5 - 3x_5^2),\\
&1/2(2 + 5x_1 - 3x_1^2),\\
&1/2 * (2 + 5x_0 - 3x_0^2),\\
&1/2(2 + 5x_2 - 3x_2^2),\\
&1/2 * (2 + 5x_7 - 3x_7^2),\\
&1/2(2 + 5x_6 - 3x_6^2),\\
&1/2 * (2 + 5x_8 - 3x_8^2),\\
&x_8,\\
&x_7,\\
&x_6,\\
&x_5,\\
&x_4,\\
&x_3,\\
&x_2,\\
&x_1,\\
&x_0\\
)
\end{aligned}
$$

No caso binário, o processo terminaria aqui, uma vez que existe apenas uma permutação possível. No entanto, para que uma regra seja invariante a cor, ela precisa ser invariante a qualquer permutação aplicada. O conjunto destas regras pode ser visto como a interseção entre todos os conjuntos de regras invariantes a uma permutação individual.

Sendo assim, o algoritmo repete o processo para cada uma das permutações possíveis restantes, obtendo um conjunto de templates que é submetido posteriormente à interseção. Neste caso, a função Solve utilizada no algoritmo de interseção não consegue estabelecer relações entre as variáveis, e retorna como resultado todas as possíveis atribuições de variáveis que satisfazem os sistemas. Assim, o resultado desta interseção acaba sendo um conjunto de 81 templates constantes, equivalentes às 81 regras invariantes a trocas de cor do espaço, testadas individualmente.

5.4 TEMPLATES PARA REGRAS CONFINADAS

O confinamento é uma propriedade que relaciona o resultado de cada item da tabela de transições com sua respectiva vizinhança. Qualquer AC cujos resultados das transições façam parte do conjunto de estados que aparecem em sua vizinhança é, por definição, confinado.

Sendo assim, basta restringir os valores de cada variável do template base a apenas o conjunto de estados existentes na vizinhança correspondente para obter um template que represente os ACs confinados de um determinado espaço.

O algoritmo recebe os valores de k e r que definem o espaço para o qual as regras confinadas serão geradas. Em seguida, gera todas as vizinhanças do espaço, e verifica quais estados existem em cada uma. Segue, então, a seguinte regra para determinar qual é o valor do template correspondente a cada vizinhança: caso a vizinhança apresente apenas um estado, sua posição correspondente no template é um valor fixo. Caso a vizinhança contenha todos os estados possíveis, ou seja, todos os valores do intervalo discreto $[0, k-1]$ sua posição correspondente no template é uma variável livre. Caso a vizinhança contenha mais do que um valor, mas não todo o intervalo discreto $[0, k-1]$, sua posição correspondente no template será uma variável restrita aos valores que aparecem na vizinhança, por meio de expressões do tipo $x_i \in C$.

No caso binário, qualquer template será formado por variáveis livres em todas as posições, com exceção das posições correspondentes às vizinhanças formadas apenas por células no estado 0 e células no estado 1, onde são fixas.

Em valores maiores de k, no entanto, aparecem casos onde as variáveis devem ser restritas para gerar os templates corretos.

O processo descrito, quando aplicado ao espaço elementar, resulta no template a seguir:

$$T_{confinadok2} = (1, x_6, x_5, x_4, x_3, x_2, x_1, 0) \tag{48}$$

Quando aplicado ao espaço de ACs de raio 1 com $k = 3$, resulta no seguinte template:

$$T_{confinadok3} = (2, x_{25} \in \{1,2\}, x_{24} \in \{0,2\}, x_{23} \in \{1,2\}, x_{22} \in \{1,2\}, x_{21}, x_{20} \in \{0,2\},$$
$$x_{19}, x_{18} \in \{0,2\}, x_{17} \in \{1,2\}, x_{16} \in \{1,2\}, x_{15}, x_{14} \in \{1,2\}, 1, x_{12} \in \{0,1\}, x_{11}, x_{10} \in \{0,1\},$$
$$x_9 \in \{0,1\}, x_8 \in \{0,2\}, x_7, x_6 \in \{0,2\}, x_5, x_4 \in \{0,1\}, x_3 \in \{0,1\}, x_2 \in \{0,2\}, x_1 \in \{0,1\}, 0) \tag{49}$$

5.5 TEMPLATES PARA REGRAS TOTALÍSTICAS E SEMI-TOTALÍSTICAS

O subespaço de regras totalísticas é formado por regras definidas apenas para as somas dos estados das vizinhanças de um AC. Como visto na Seção 3.5, é possível representar uma regra totalística com uma tabela de transições comum, fazendo com que os resultados das transições de cada vizinhança sejam iguais ao resultado da transição da soma na tabela do AC totalístico.

Seguindo esta ideia, é possível gerar um template representante de todas as regras totalísticas (e, seguindo um processo parecido, semi-totalísticas) de um determinado espaço, fazendo apenas com que as transições cuja vizinhança tenha a mesma soma levem ao mesmo resultado.

O algoritmo recebe os valores de k e r que definem o espaço cujo template de regras totalísticas será criado. Então, enumera todas as vizinhanças do espaço e, para cada uma,

calcula sua soma. O resultado das somas dos estados de uma vizinhança determinam quais transições devem ser iguais para que uma regra seja totalística.

Dada uma vizinhança qualquer, o algoritmo segue a seguinte regra para definir qual será a variável associada. Caso a vizinhança seja a única que obteve um determinado valor de soma, uma variável x_i será criada para esta transição, sendo i igual ao valor decimal da vizinhança. Em caso contrário, a transição receberá uma variável x_i onde i é igual ao valor decimal da menor vizinhança que tenha o mesmo resultado da soma.

O template gerado por este processo mantém a integridade semântica com outros templates, uma vez que qualquer variável livre usa o subscrito equivalente à sua vizinhança em valor decimal, e os campos relacionados usam a variável representante da menor vizinhança. O processo descrito, quando aplicado ao espaço elementar, gera o template a seguir.

$$T_{totECA} = (x_7, x_3, x_3, x_1, x_3, x_1, x_1, x_0) \tag{50}$$

Quando expandido, o template em questão gera $k^m = 2^4 = 16$ regras, que correspondem às 16 regras totalísticas do espaço elementar.

Para o espaço de $k = 3$ e $r = 1$, o algoritmo gera o template a seguir.

$$\begin{aligned} T_{tot31} = (&x_{26}, x_{17}, x_8, x_{17}, x_8, x_5, x_8, x_5, x_2, x_{17}, \\ &x_8, x_5, x_8, x_5, x_2, x_5, x_2, x_1, x_8, x_5, x_2, x_5, x_2, x_1, x_2, x_1, x_0) \end{aligned} \tag{51}$$

Quando expandido, este template gera $k^m = 3^7 = 2187$ regras, correspondentes às 2187 regras totalísticas do espaço.

Para encontrar as regras semi-totalísticas de um determinado espaço, o mesmo processo básico é seguido, com um detalhe diferente.

As vizinhanças são somadas como nas regras totalísticas, porém, a célula do meio é desconsiderada, e seu estado concatenado ao final da soma. Seguindo este processo, vizinhanças cuja soma das células externas e o estado da célula do meio sejam iguais, como por exemplo, $(0, 2, 1)$ e $(1, 2, 0)$ geram a mesma *string* ($0 + 1 = 1 + 0 = 1$, concatenando 2 ao final, a *string* de ambas é 12). O algoritmo segue então a mesma regra aplicada

às totalísticas: caso a vizinhança seja a única que obteve um determinado código, uma variável x_i será criada para esta transição, sendo i igual ao valor decimal da vizinhança; caso contrário, a transição receberá uma variável x_i onde i é igual ao valor decimal da menor vizinhança que tenha o mesmo código.

Aplicando este algoritmo ao espaço elementar, o template a seguir é obtido.

$$T_{totExtEE} = (x_7, x_3, x_5, x_1, x_3, x_2, x_1, x_0) \tag{52}$$

Quando expandido, o template gera $k^m = 2^6 = 64$ regras, que correspondem às regras semi-totalísticas do espaço elementar.

Para o espaço de $k = 3$ e $r = 1$, o algoritmo gera o template a seguir.

$$\begin{aligned} T_{totExt31} = (&x_{26}, x_{17}, x_8, x_{23}, x_{14}, x_5, x_{20}, x_{11}, x_2, x_{17}, x_8, x_7, \\ &x_{14}, x_5, x_4, x_{11}, x_2, x_1, x_8, x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \end{aligned} \tag{53}$$

Quando expandido, o template gera $k^m = 3^{15} = 14.348.907$ regras, que correspondem às regras semi-totalísticas do espaço.

6 CONCLUSÕES E TRABALHOS FUTUROS

No presente trabalho, o conceito de templates de autômatos celulares foi introduzido: uma forma de representação de conjuntos de ACs, construída como uma generalização da tabela de transições em formato k-ário, inspirada pelas construções algébricas utilizadas em (LI; PACKARD, 1990).

O conceito foi proposto como uma maneira de representar conjuntos de ACs cujos itens compartilham de determinadas propriedades dinâmicas, fazendo com que qualquer busca por um AC com determinado comportamento possa ser realizada num espaço delimitado por uma determinada propriedade.

Além de ferramentas conceitualmente poderosas, os templates aqui propostos são apoiados pelo desenvolvimento da biblioteca *CATemplates*, um pacote do Wolfram Mathematica (WOLFRAM RESEARCH, 2013) que, até o momento, implementa algoritmos geradores de templates para cinco tipos de propriedades: a conservabilidade de estados, a simetria interna (por conjugação, reflexão e composição das transformações), a invariância a trocas de cor, o confinamento, e a totalidade (e, de maneira análoga, da semi-totalidade).

Além disso, a biblioteca *CATemplates* disponibiliza duas operações aplicáveis aos templates: a expansão, que permite que um template qualquer seja transformado no conjunto que representa, e a interseção, que permite encontrar templates representantes da interseção dos conjuntos representados pelos templates originais.

O algoritmo de geração de templates para regras com valores arbitrários de simetria interna, atualmente, gera apenas templates para espaços binários de ACs. Uma possível generalização deste algoritmo poderia surgir ao aplicar conceitos parecidos com os aplicados na geração de templates para regras invariantes a cor: a generalização da conjugação, e a obtenção de polinômios representantes de funções de permutação.

A operação de interseção serve como uma maneira de manipular templates, de forma a representar conjuntos menores antes da aplicação da expansão que, por padrão, é uma operação custosa. No entanto, por depender da utilização da função Solve, disponibilizada pelo Wolfram Mathematica (WOLFRAM RESEARCH, 2013), a interseção entre templates pode em alguns casos acabar por resolver o sistema de equações montado, gerando um conjunto de constantes ao invés de um template mais geral. Este foi o caso quando a

interseção foi utilizada para gerar templates para regras invariantes a cor em espaços não binários. Atualmente, a utilização de operações com álgebra modular na resolução desse tipo de sistema está sendo experimentada como uma forma de encontrar templates mais genéricos.

Além disso, qualquer propriedade estática de um AC que relacione as vizinhanças da tabela de transições pode ter uma função geradora de templates equivalente. A exploração de novas propriedades ainda pode revelar novas extensões para a representação, como ocorreu durante o desenvolvimento do projeto. Entre estas, considera-se o *balanceamento*, que representa a frequência relativa com que cada estado aparece na representação k-ária da regra, bem como propriedades relacionadas ao comportamento dinâmico de regras unidimensionais, como o *domínio da vizinhança*, a *sensitividade* e a *atividade absoluta* (OLIVEIRA; DE OLIVEIRA; OMAR, 2001).

As propriedades usadas como exemplo são baseadas em relações já conhecidas entre as transições da tabela de um AC, o que é uma condição para que qualquer propriedade possa ser representada por um template. Como contraponto, a noção de reversibilidade em regras unidimensionais não parece ser, pelo menos a princípio, uma propriedade que possa ser abordada por meio de templates, já que atualmente não se sabe como caracterizar a reversibilidade em termos da tabela de transições de um AC.

Alguns dos algoritmos de geração de templates acabam por ser mais complexos do que outros. Em essência, as propriedades que estabelecem relações simples entre as transições (como a totalidade, a invariância a cores no caso binário e o confinamento) são traduzidas em algoritmos mais leves e eficientes do que as propriedades que dependem da simplificação de um sistema de equações (como na conservabilidade e na invariância a cores em casos não binários).

Existe também uma certa limitação quanto à verificação da integridade dos templates encontrados em espaços muito grandes. Apesar dos processos seguidos garantirem que as regras representadas por um template tenham uma determinada propriedade, a prova da integridade de um template é dada principalmente por meio da verificação manual dos itens do espaço. Embora esta seja uma estratégia aplicável a espaços menores, ela é inviável em espaços maiores.

Outro importante ponto a ser levantado é a performance da operação de expansão de

template, que, por sua própria natureza, não escala bem ao tentar expandir templates representantes de conjuntos muito grandes.

Este fato, no entanto, não se contrapõe ao objetivo inicial do projeto, uma vez que, através dos algoritmos de geração de template, foram encontrados templates representantes de espaços da ordem de 2^{528} em questão de segundos. Nota-se então que o framework aqui proposto tem alto potencial de manipulação de subespaços, uma vez que templates desta ordem podem ser manipulados e diminuídos antes da realização de sua expansão.

Para evitar problemas causados por falta de memória durante a expansão de um conjunto muito grande, o algoritmo pode ser aplicado apenas a uma i-ésima substituição, ou a uma faixa de substituições, permitindo ao usuário realizar a expansão em pedaços e, possivelmente, gravando os resultados em algum tipo de memória persistente, como um arquivo ou banco de dados. Seguindo esta estratégia, o algoritmo também pode ser paralelizado, diminuindo o tempo necessário para realizar a expansão de um template que tenha muitas variáveis.

Versões iniciais dos algoritmos de expansão de templates, de geração de templates para regras conservativas e para regras com simetria interna máxima já foram descritas em (DE OLIVEIRA; VERARDO, 2014a), e apresentadas na *Automata 2014: 20th International Workshop on Cellular Automata and Discrete Complex Systems* (DE OLIVEIRA; VERARDO, 2014b).

Todo o código fonte do projeto está disponível no repositório público do projeto (VERARDO; DE OLIVEIRA, 2014), onde qualquer contribuição pode ser feita e será bem-vinda.

REFERÊNCIAS BIBLIOGRÁFICAS

BERLEKAMP, E. R.; CONWAY, J. H.; GUY, R. K. *Winning Ways for Your Mathematical Plays*. [S.l.]: Academic Press, 1982. 927-960 p.

BETEL, H.; DE OLIVEIRA, P. P. B.; FLOCCHINI, P. Solving the parity problem in one-dimensional cellular automata. *Natural Computing*, Springer Netherlands, 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. 52, p. 1–13, 2002.

COOK, M. Universality in elementary cellular automata. *Complex Systems*, v. 15, n. 1, p. 1–40, 2004.

DE OLIVEIRA, P. P. B. Conceptual connections around density determination in cellular automata. *Lecture Notes in Computer Science*, v. 8155, p. 15–57, 2013.

DE OLIVEIRA, P. P. B. On density determination with cellular automata: results, constructions and directions. *Journal of Cellular Automata*, v. 9, n. 5-6, p. 357–385, 2014.

DE OLIVEIRA, P. P. B.; VERARDO, M. Representing families of cellular automata rules. *The Mathematica Journal*, v. 16, n. 8, 2014.

DE OLIVEIRA, P. P. B.; VERARDO, M. Template based representation of cellular automata rules. *Proc. 20th International Workshop on Cellular Automata and Discrete Complex Systems (Himeji, Japan, July 7th-9th, 2014.)*, 2014.

FUKŚ, H. A class of cellular automata equivalent to deterministic particle systems. *Hydrodynamic limits and related topics*, p. 57–69, 2000.

KARI, J. Theory of cellular automata: A survey. v. 334, p. 3–33, april 2005.

KARI, J.; LE GLOANNEC, B. Modified traffic cellular automaton for the density classification task. *Fundamenta Informaticae*, IOS Press, v. 116, n. 1-4, p. 141–156, 2012.

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

OLIVEIRA, G. M. B.; DE OLIVEIRA, P. P. B.; OMAR, N. Definition and application of a five-parameter characterization of one-dimensional cellular automata rule space. *Artificial Life*, v. 7, n. 3, p. 277–301, 2001.

SALO, V.; TÖRMÄ, I. Color blind cellular automata. *Journal of Cellular Automata*, v. 9, n. 5-6, 2014.

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

THEYSSIER, G. Captive cellular automata. In: *Mathematical Foundations of Computer Science 2004*. [S.l.: s.n.], 2004, (Lecture Notes in Computer Science, v. 3153). p. 427–438. ISBN 978-3-540-22823-3.

THEYSSIER, G. How common can be universality for cellular automata? In: DIEKERT, V.; DURAND, B. (Ed.). *STACS 2005*. [S.l.: s.n.], 2005, (Lecture Notes in Computer Science, v. 3404). p. 121–132.

VERARDO; DE OLIVEIRA. *CATemplates*. 2014. `https://github.com/mverardo/CATemplates`.

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

WOLFRAM RESEARCH. *Wolfram Mathematica 9.0.3*. 2013. `http://www.wolfram.com/mathematica/`.

WOLZ, D.; DE OLIVEIRA, P. P. B. Very effective evolutionary techniques for searching cellular automata rule spaces. *Journal of Cellular Automata*, 2008.