DIFERENÇA DE TEMPLATES K-ÁRIOS NA REPRESENTAÇÃO DE FAMÍLIAS DE AUTÔMATOS CELULARES UNIDIMENSIONAIS

Ellias Matheus Braga e Silva (IC) e Pedro Paulo Balbi de Oliveira (Orientador)

Apoio: PIVITI Mackenzie

RESUMO

Discutem-se aqui os resultados da implementação de uma proposta de operação de diferença de templates de autômatos celulares unidimensionais *k*-ários. Para essa implementação ser possível foi antes desenvolvida a operação de negação para o caso *k*-ário, que resultou em resultados positivos, ainda que não para a totalidade dos casos. Apesar disso, os resultados obtidos foram suficientes para evidenciar a viabilidade e capacidade da operação, além de se demonstrar que a operação de negação não consegue de forma trivial retornar um único template que a represente.

Palavras-chave: Autômatos celulares unidimensionais; autômatos celulares *k*-ários; templates; diferença de templates; operação de negação.

ABSTRACT

We discuss the results of the implementation of a proposal for the operation of template difference for *k*-ary one-dimensional cellular automata. For this implementation to be possible, the negation operation for the *k*-ary case was initially developed, which resulted in positive results, although not for all the possible cases. Nevertheless, the results obtained were sufficient to make it evident the viability and capacity of the operation, in addition to demonstrating that the negation operation cannot trivially return a single template that represents it.

Keywords: One-dimensional cellular automata; *k*-ary cellular automata; templates; difference of templates; negation operation.

1. INTRODUÇÃO

Autômatos Celulares (AC's) são sistemas computacionais e sistemas dinâmicos discretos regidos por regras de transição simples e que, apesar disso, podem exibir comportamento imprevisivelmente complexo. A quantidade de autômatos possíveis vai de 256 possibilidades (espaço elementar de AC's unidimensionais) até números expressivamente grandes dependendo dos seus parâmetros de construção. Tais autômatos podem ser classificados em famílias, de acordo com suas regras de transição e com seus comportamentos.

As regras de transição, por sua vez, determinam os estados dos autômatos celulares nas próximas gerações baseada nos estados da geração atual e de sua vizinhança. Os comportamentos dos AC's podem ser analisados a partir de propriedades dinâmicas, i.e., as obtidas da evolução temporal, como a reversibilidade, ou de propriedades estáticas, aquelas obtidas diretamente da regra de transição, entre as quais podemos citar a *conservabilidade de estados* (BOCCARA; FUKS, 2002), a *simetria interna*, *o confinamento* (WOLFRAM, 2002), etc.

As famílias de autômatos podem ser representadas por *templates*, uma estrutura que consiste na representação *k*-ária de um autômato genérico cujos elementos são parcialmente substituídos por variáveis, de forma que consigam representar todos os autômatos que compartilhem uma ou mais propriedades estáticas, semelhante ao que foi feito em (BETEL; DE OLIVEIRA; FLOCCHINI, 2013) para o problema da paridade.

Os templates de AC's são passíveis de aplicação de operações como a *intersecção* e a *expansão*. O presente trabalho tem como objetivo a implementação de uma nova operação de templates e análise dos templates produzidos por ela dando continuidade aos trabalhos que o alicerçam conforme explicado na seção 1.3.

1.1. Problema de Pesquisa:

Outras pesquisas sobre o tema como (DE OLIVEIRA; VERARDO, 2014) apontaram a possibilidade de determinar templates de famílias de regras de autômatos baseados nas suas propriedades cuja manipulação evita o trabalho de enumerar um espaço de busca por completo à procura de autômatos com as propriedades esperadas (SOARES; VERARDO; DE OLIVEIRA, 2016).

Considerando isso, pode-se pensar na ideia de como representar uma família de regras que **não** possuem certa propriedade. Dessa forma pode-se ter um template que representa regras com as propriedades desejadas e as propriedades não desejadas.

1.2. Justificativa:

Muitos estudos utilizam metodologias que se baseiam em apenas um tipo específico de AC com propriedades específicas obtidas através de restrições em suas regras de transição como, por exemplo, é o caso dos AC's *confinados* (THEYSSIER, 2004). Nesse sentido, a existência de um template de AC's tem seu uso para qualquer estudo que se defina a partir de propriedades estáticas em suas análises, pois substitui o trabalho de enumeração de um espaço inteiro de busca (que tipicamente é muito grande) para encontrar esses autômatos específicos.

Além disso, é comprovada a importância dos autômatos celulares em geral pelo seu sucesso em simular eventos naturais e comportamentos de sistemas vivos - em alguns casos, como no Jogo da Vida de (BERLEKAMP; CONWAY; GUY, 1982) ou no AC elementar 110 apresentado em (WOLFRAM, 2002) cuja computabilidade universal foi demonstrada em (COOK, 2004).

Posto isso, o aumento das ferramentas de manipulação de templates significa um aumento na capacidade de abstração lógica dos sistemas que fazem uso deles, ou, em outras palavras, significa um avanço nas possibilidades de uso dos mesmos.

1.3. Objetivo:

O objetivo deste trabalho foi desenvolver um modelo de template capaz de representar famílias de regras a partir de propriedades desejadas e de propriedades não desejadas através do uso e extensão do software Wolfram Mathematica (WOLFRAM RESEARCH, 2018) com a biblioteca CATemplates (VERARDO; DE OLIVEIRA, 2019b).

Essa abordagem foi tratada anteriormente em outro trabalho (VERARDO; DE OLIVEIRA, 2019), cuja proposta dependia de uma operação de exceção representada por X que consiste em obterem-se os autômatos cuja estrutura gera autômatos fora do domínio do template, operações de intersecção representadas por

I e uma união dos resultados parciais obtidos por essas operações - para tal, dado um template T_m cujas propriedades são desejadas e um template T_s cujas propriedades não são desejadas, obtém-se a intersecção desses templates definida por T_{ms} e aplicam-se duas operações não concorrentes de exceção resultando em T^x_{ms} e de intersecção da negação de T_{ms} com T_m resultando em T'_m (T_m linha), após isso sendo necessária outra intersecção entre T^x_{ms} e T_{ms} resultando em T''_m e (T_m duas linhas) e só então a diferença de template é dada pela união de T''_m e T'_m (conforme *Fig. 1*).

Entretanto, como demonstrado naquele trabalho, uma operação de negação para *k* maior que 2 não é trivial e intersecções são operações que demandam comparações dos autômatos representados pelos templates envolvidos de forma que nos casos de propriedades não desejadas o resultado não foi satisfatório por ser computacionalmente intensivo, restritivo (tratava apenas casos binários) e conceitualmente não elegante.

Nesse sentido, foi proposta a continuação desses trabalhos anteriores a partir da análise e implementação de novos templates, pelo uso da operação da diferença de templates *k*-ários.

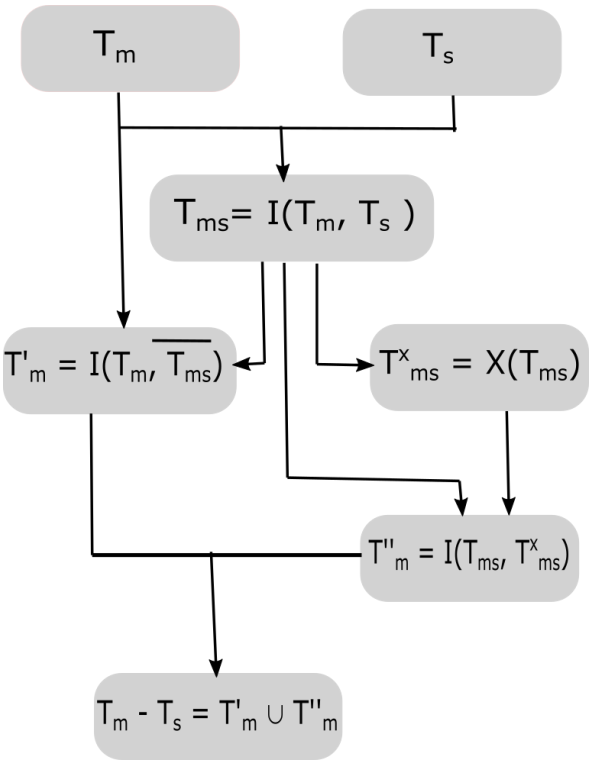


Figura 1: Diferença entre templates - Diagrama da abordagem anterior (*SOARES; VERARDO; DE OLIVEIRA, 2016*).

2. REFERENCIAL TEÓRICO

De acordo com (VERARDO; DE OLIVEIRA, 2019), um *template* é definido por uma *tupla* (*k, r, c, e*) na qual *k* e *r* são, respectivamente, o número de *estados* e o raio de vizinhança das regras do autômato celular no espaço sendo representado, *c* é o núcleo do template e *e* é uma função modificadora de expansão. O template consiste em uma generalização da tabela de transições de estados clássica tornando possível a representação de subespaços de autômatos celulares, ao invés de indivíduos isolados (abordagem clássica).

A ideia de template é concebida a partir da ideia de propriedade estática de autômatos. Propriedades estáticas são propriedades calculadas a partir da tabela de transição de ACs que permitem prever o comportamento durante a evolução de determinado autômato sem que seja necessário simulá-lo (SOARES; VERARDO; DE OLIVEIRA, 2016).

São exemplos de propriedades estáticas a conservabilidade de estados que é uma propriedade apresentada por alguns AC's em que a soma dos estados das células individuais em qualquer condição inicial não muda durante a evolução temporal (SOARES; VERARDO; DE OLIVEIRA, 2016); e o confinamento, que ocorre quando todas as transições da tabela de transições de um AC têm como resultado um estado que faça parte da respectiva vizinhança (THEYSSIER, 2004).

Uma tabela de transição de AC's é uma representação da regra de transição de um AC. Tal tabela é uma tabela de *n*-uplas com n sendo 2*r* + 1 para o caso unidimensional (sendo o espaço elementar representado por $r = 1$ e $k = 2$, ou seja, três células em cada vizinhança e dois estados possíveis para cada célula) cujos elementos são uma das combinações de vizinhança e o estado gerado por ela, ordenado lexicograficamente de acordo com a vizinhança (COSTA, 2014).

No presente trabalho foi utilizado o padrão de ordenação de Wolfram, que é dado pelo primeiro item da *n*-upla sendo aquele cuja vizinhança formada por todas as células no estado $k - 1$ (último estado possível) e o último com a vizinhança formada por zeros (o primeiro estado possível).

A representação *k*-ária de uma tabela de transição (LI; PACKARD, 1990), é a representação obtida ao descartar as vizinhanças mantendo apenas os estados resultantes; por exemplo, para *regra elementar 30* a tupla que a representa é:

(((1, 1, 1), 0), ((1, 1, 0), 0), ((1, 0, 1), 0), ((1, 0, 0), 1),

((0, 1, 1), 1), ((0, 1, 0), 1), ((0, 0, 1), 1), ((0, 0, 0), 0)))

Ao removermos as vizinhanças cuja organização é bem conhecida, temos a representação *k*-ária (no caso, binária):

(0, 0, 0, 1, 1, 1, 1, 0, 0)

É importante destacar que a representação *k*-ária só é possível quando *r* e *k* são conhecidos, de forma que seja possível reverter a representação *k*-ária em tabela de transições (SOARES; VERARDO; DE OLIVEIRA, 2016).

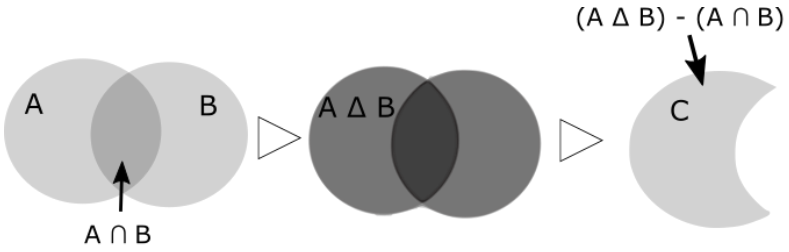
Os estados das células são tipicamente representados por números pertencentes ao intervalo discreto [0, $k - 1$], ou, quando se está visualizando a evolução temporal, por uma cor dentre *k* previamente definidas (SOARES; VERARDO; DE OLIVEIRA, 2016).

O conjunto de todas as possibilidades de autômatos unidimensionais pode ser particionado pelo uso de dois importantes parâmetros: o número de estados possíveis para cada célula de um AC em determinado momento, e o tamanho de sua vizinhança, que é definida pelo número de vizinhos que uma célula deve levar em consideração para o cálculo de seu próximo estado no tempo seguinte. Família é o termo que representa cada agrupamento resultante deste particionamento.

Dado um template, são possíveis duas operações básicas: a expansão, que consiste em resolver o template em todos os autômatos que ele representa, e a intersecção, que é uma função cuja entrada é dois templates e a saída é um template cujo conjunto de regras representadas é igual à intersecção entre o conjunto de regras representadas pelos dois templates iniciais.

A partir das operações básicas obtém-se a operação de diferença entre templates (*Fig. 2*), operação utilizada para obter-se um conjunto de autômatos representados pelo template cujas características são desejadas com exceção dos

autômatos que também são representados por um template cujas características **não** são desejadas. Essa operação pode ser definida como o resultado da diferença simétrica entre os conjuntos de regras representadas por dois templates, subtraída pelo resultado da subtração entre o conjunto de regras representadas pelo segundo template e a interseção entre o conjunto de regras representadas pelos dois templates.



A = Conjunto de regras representadas por T1 (template cujas propriedades são desejadas)

B = Cojunto de regras representadas por T2 (template cujas propriedades **não** são desejadas)

A Δ B = Diferença Simétrica entre A e B

A ∩ B = Conjunto de regras que pertencem a ambos os conjuntos(Intersecção)

C = Conjunto de regras que pertencem a A e que não pertencem a B (Diferença entre Templates)

Figura 2: Diferença entre templates.

3. METODOLOGIA

A abordagem dessa proposta pode ser dividida em duas fases distintas que se resumem em: (1) fase de embasamento teórico, e (2) fase de aplicação dos conceitos na análise e implementação de uma generalização da tabela de transições de estados de autômatos e do algoritmo para sua obtenção. Ambas as fases, foram divididas em etapas menores interdependentes.

Em princípio foi feito o estudo da linguagem e ambiente do software Wolfram Mathematica, a partir da documentação da linguagem, em conjunto com o estudo dos principais textos do referencial teórico.

Em seguida foram iniciados os estudos das bibliotecas CAMaT (DE OLIVEIRA, 2018) e CATemplates (VERARDO; DE OLIVEIRA, 2019b). Por fim, foram feitas as avaliações, implementações, testes da proposta, e, então, realizados os testes com os templates já existentes na biblioteca CATemplates,.

Durante a etapa inicial foram revisados e analisados todos os textos descritos nas referências bibliográficas utilizando metodologia de fichamento e mapa mental para fixação.

Durante a etapa de aprendizagem das bibliotecas relativas aos autômatos celulares foi usada a metodologia de uso de simulações e testes com propriedades estáticas e templates já identificados.

A partir dessa etapa começaram as análises da proposta deste documento, isto é, implementação, testes e avaliação de algoritmos que encontram subespaços de regras, e que apresentam propriedades em comum, seja pela presença ou ausência das propriedades estáticas.

Os primeiros casos de testes consistiram em testar a proposta inicial do uso de aritmética modular para realizar negações semelhantes à implementação binária já existente; entretanto, devido ao comportamento desse tipo de cálculo quando números são primos entre si, o software não conseguia resolver os cálculos. Após as etapas iniciais, percebida a impossibilidade, mesmo após diversas tentativas, a abordagem mudou para provar a possibilidade da existência de um template que representasse a negação de outro.

Para tal, fora utilizada a abordagem de expandir e negar o template por força bruta realizando a diferença de conjuntos entre o template base (que cobre todo o espaço de busca) e o template a ser negado.

Uma vez que essa abordagem funcionou, já nas etapas finais, a última proposta começou a ser implementada, baseada na negação de cada posição do template e das combinações dessas negações, cujas dificuldades serão apresentadas nas próximas seções.

Os testes consistem em comparar os resultados da expansão dos templates negados, subtraídos da expansão do template base com o resultado da expansão do

resultado da negação dos templates e foram feitos para *k*=2 e *k*=3, por considerar que *k*'s maiores seriam muito custosos ao hardware disponível.

Para esses últimos passos foi utilizada a *Wolfram Language*, linguagem de programação associada ao software Wolfram Mathematica, em conjunto com as bibliotecas CAMaT e CATemplates, com divisão da etapa de análise em caso unidimensional e caso *k*-ário.

Após isso foram incorporados os resultados na biblioteca CATemplates e foram feitos testes com propriedades já implementadas.

4. RESULTADO E DISCUSSÃO

Durante a pesquisa foram feitos testes com a implementação por aritmética modular, por base de expansão e com base na negação posicional. Durante a implementação notou-se a impossibilidade de realizar a negação com aritmética modular pois os cálculos não resolvem quando os valores não são primos entre si. Com a expansão foi possível, provando que, teoricamente, existe um template capaz de representar a negação de outro template de forma a representar todo o conjunto de autômatos com exceção dos representados pelo template negado – negação global em relação ao domínio do template. O processo consiste em expandir o template a ser negado, realizar a diferença de conjuntos entre a expansão do template base (template que representa todo o espaço de busca com os mesmos parâmetros do template a ser negado) com a expansão do template a ser negado e realizar uma operação de geração automática de templates a partir de um conjunto de autômatos resultante. O template resultante, porém possui formato complexo e incompatível com os templates existentes e a operação é muito custosa, precisando necessariamente percorre todo o espaço de busca, o que é incoerente com a proposta do template. Durante essa abordagem foi demonstrado que apesar de existir um template que reflita a negação de outro template, a obtenção desse possível template não é trivial, sendo necessária a expansão completa ou uma possível união de templates resultantes da negação. Para a última abordagem foi necessário mudar o foco da operação de diferença para a operação de negação, pois a diferença entre dois templates pode ser definida como a intersecção do template cujas propriedades são desejadas, com a negação do template cujas propriedades são indesejadas (conforme *Fig. 2*). Levando isso em conta, foi feita a implementação da negação posicional que consiste em realizar a permutação de negações locais no template negado, isto é, gerar negações de cada posição do template e criar templates permutando essas negações, sendo o resultado esperado

um conjunto de templates cuja união de suas expansões resultasse na negação global do template conforme *Fig. 3*.

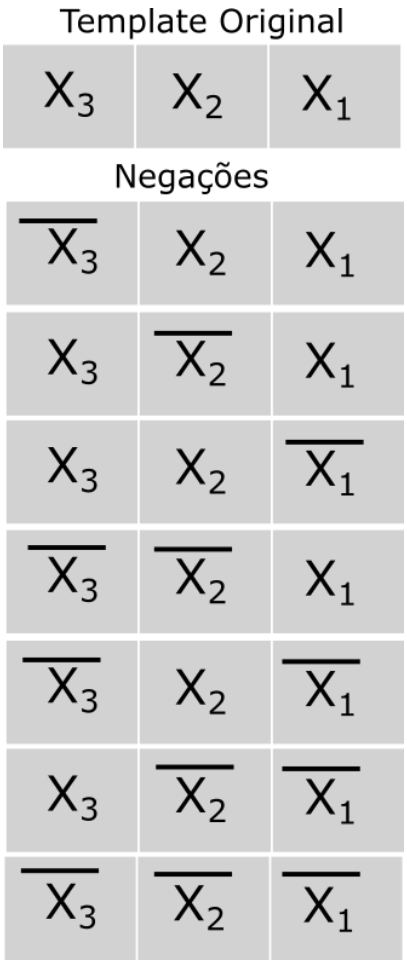


Figura 3: Exemplo de negação posicional.

Porém, durante o desenvolvimento dessa abordagem, porém foram encontrados diversos problemas que tornaram o algoritmo mais complexo e mais demorado de ser desenvolvido e testado. Parte do problema surgiu da não padronização da notação de templates atuais, tornando necessário destrinchar a operação de negação em diversos casos específicos para cada notação existente, que podem ser divididas em 4 casos básicos e suas combinações, quais sejam:

1. Posições representadas por uma variável simples:

A negação não gera templates da negação dessas posições (sozinhas). Ex.:

(0, 0, 0, 1, 1, 1, 1, X_1, 0)

2. Posições representadas por uma variável simples restringida por notação de conjuntos:

   A negação gera templates para cada valor possível, dado o *k* do template, com exceção dos valores restringidos na posição. Ex.:

   $$(0, 0, 0, 1, 2, 1, 1, X_1 \in \{ 0, 1\} , 2)$$

3. Posições representadas por uma variável simples cujo valor é modificado por uma operação matemática de uma variável:

   A negação precisa levar em consideração a posição em que a variável em questão estivesse sozinha (posição principal) e então avaliar todos os resultados possíveis (expansão local) negando os resultados de acordo com o *k* do template. Ex.:

   $$(0, 0, 0, 1, X_1 +1, 1, 1, X_1 , 2)$$

4. Posições representadas por uma operação matemática representada por mais de uma variável:

   A negação precisa levar em consideração todas as posições em que as variáveis em questão estivessem sozinhas, e então realizar a expansão local negando os resultados de acordo com o *k* do template. Ex.:

   $$(0, 0, 0, 1, X_1 + X_2, 1, 1, X_1 , 2)$$

Além disso, surgiu a necessidade de recombinar todas as negações para que seja possível atingir todo o espaço de busca, i.e., combinar a negação da primeira posição com a negação da segunda posição gerando os templates que representam aqueles cuja estrutura nega tanto a primeira como a segunda posição ao mesmo tempo, e assim por diante. Outro problema foi a falta de estrutura de dados necessárias para o funcionamento do algoritmo, as quais tiveram de ser criadas e ajustadas conforme o algoritmo era desenvolvido e os problemas encontrados, cuja refatoração não pôde ser feita de forma adequada devido ao tempo (essa abordagem foi a última a ser realizada, nas etapas finais do projeto). A descoberta dos últimos casos (a necessidade de tratar separadamente posições calculadas a

partir de uma operação matemática representada por mais de uma variável) e a necessidade de realizar as expansões locais foi um ponto chave para que o algoritmo fosse capaz de negar um template; porém, essa percepção ocorreu tardiamente durante os testes, no momento em que já estavam sendo incorporadas as funções da biblioteca CATemplates, de forma que o tempo estava escasso para refatorar o algoritmo considerando essa nova necessidade. Isso resultou num desempenho não adequado para templates cujo *k* é maior que 2, já que o uso dessas expansões locais tornou necessário ao software guardar todos os casos de negação na memória antes da geração dos templates, além de ter que combiná-los -- tarefa que não é eficiente usando as bibliotecas atuais do software Mathematica pela incapacidade de filtrar as redundâncias.

Além disso, apesar de o Mathematica trazer a facilidade de trabalhar com operações matemáticas por suas abstrações e paradigma simbólico, ele acaba por fazer uso recorrente da memória para garantir tais funcionalidades, tornando o desenvolvimento complicado, mesmo num hardware com memória relativamente alta para esse tipo de desenvolvimento (32GB). Por fim, considerando todas as necessidades, o fluxograma do algoritmo ficou conforme a *Fig. 4* abaixo:

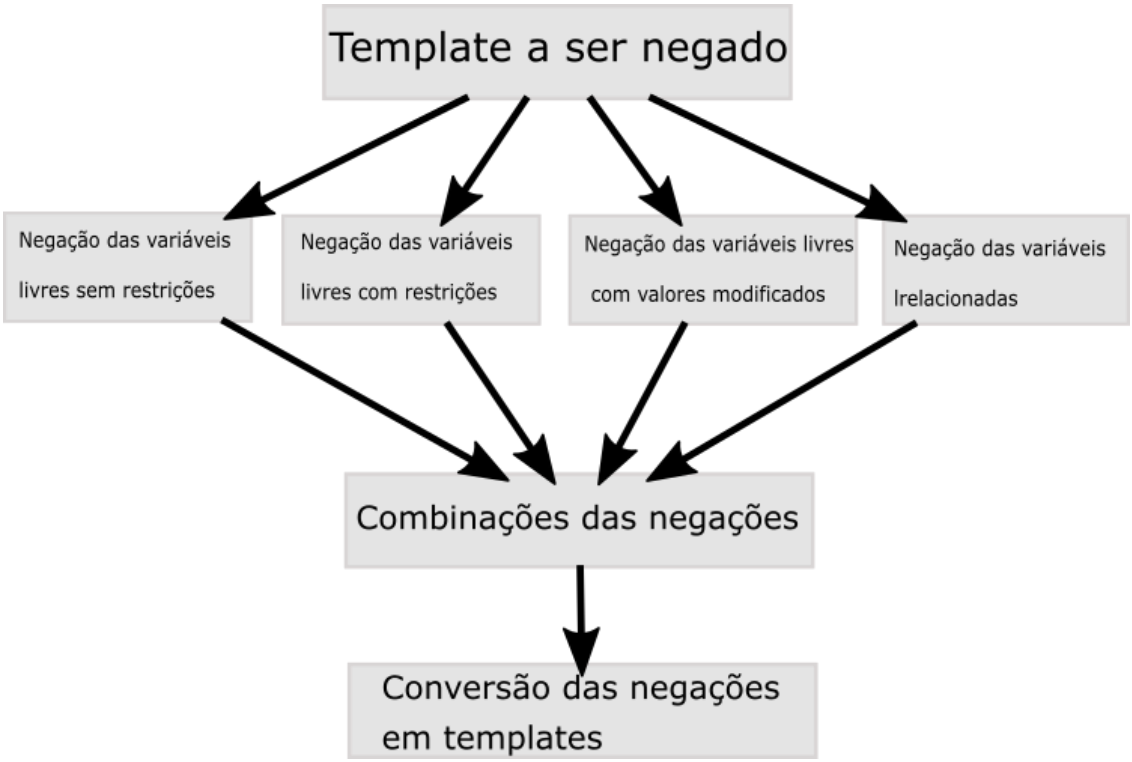


Figura 4: Fluxograma da negação de templates.

5. CONSIDERAÇÕES FINAIS

Considerando os pontos relatados, apesar de não ter sido possível testar a implementação para templates cujo *k* é maior que 2, operação de diferença provou-se possível tanto pela segunda abordagem (força bruta) como pela negação posicional (em teoria) que provou-se eficiente para *k* igual à 2 sem utilizar de cálculos específicos para esse tipo de parâmetro (binário) -- apesar de ter sido encontrado e proposto um ajuste na operação para que, para o caso binário, o algoritmo fique ainda mais eficiente e elegante. É importante ressaltar que melhorias na solução são necessárias e podem, com grandes possibilidades, tornar o algoritmo capaz de solucionar para *k* maiores que 2, pois o código foi desenvolvido e ajustado conforme os muitos impedimentos e condições não previstas anteriormente surgiam, não havendo tempo para refatoração das estruturas de dados utilizadas.

São notáveis as seguintes condições que surgiram no desenvolvimento da solução:

1. Ao negar-se cada posição do template, são necessárias as negações das relações em que cada posição pode estar envolvida, e isso torna necessário avaliar essas relações em conjunto com quaisquer posições em que as variáveis envolvidas apareçam (operação descrita como expansão local do template).

2. Mesmo para variáveis (posições) livres de relacionamentos, existem ao menos 3 casos de negações possíveis:

    a. Negar variáveis simples;

    b. Negar posições calculadas aritmeticamente; e

    c. Negar posições calculadas por notação de conjuntos.

Essa falta de padronização para definir as posições dos templates, que refletem a diversidade das propriedades que um autômato pode apresentar, aumentou a complexidade do algoritmo. Por fim, o trabalho conseguiu provar que a proposta de uma operação de diferença de templates é viável, inclusive para casos

k-ários (utilizando regras que não se aplicam somente ao contexto binário, ou seja, genéricas o suficiente para serem aplicadas a quaisquer valores de *k*), mas não pôde oferecer uma solução eficaz para todos os casos até o término deste artigo. Além disso, um dado importante levantado durante os testes é da não trivialidade para obter-se um único template a partir da negação proposta, tornando a negação de um template uma operação que retorna *n*-templates cuja união das expansões equivale a todos os autômatos do espaço de busca com exceção dos autômatos representados pelo template negado. Para trabalhos futuros são indicadas a revisão da notação de templates, de forma a diminuir o número de possibilidades de casos a serem tratados em suas operações; refatoração do algoritmo de suas estruturas de dados, para melhor desempenho, como a paralelização das negações, conforme mostrado na *Fig. 3*; e o desenvolvimento de uma operação de combinação de negações que desconsidere redundâncias, diminuindo, assim, a quantidade de dados que precisam ser alocados em memória.

Agradecimentos:

Este trabalho se beneficiou dos auxílios de pesquisa concedidos pela CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) a meu orientador, através dos projetos STIC-AmSud (CoDANet) no. 88881.197456/2018-01 e PrInt no. 88887.310281/2018-00.

6. REFERÊNCIAS

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 onedimensional cellular automata. Natural Computing, Springer Netherlands, v. 12, n. 3, p. 323–337, 2013.

BOCCARA, N.; FUKS, H. Number-conserving cellular automaton rules. Fundamenta Informaticae, 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.; VERARDO, M. Representing families of cellular automata rules. The Mathematica Journal, v. 16, n. 8, 2014.

DE OLIVEIRA, P. P. B. CAMaT.nb: Cellular Automata Mathematica Toolbox. Distribuição restrita disponibilizada pelo autor, 2018.

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

SOARES, Z.; VERARDO, M.; DE OLIVEIRA, P. P. B. "The difference operation between templates of binary cellular automata." New Advances in Information Systems and Technologies. Springer, V. 444, p. 707-715, 2016.

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 978-3540-22823-3.

VERARDO, M.; DE OLIVEIRA, P. P. B. A Fully Operational Framework for Handling Cellular Automata Templates. Complexity, v. 2019, 2019.

VERARDO, M.; DE OLIVEIRA, P. P. B. CATemplates: A Mathematica package that enables Cellular Automata researchers (and enthusiasts) to use the Template framework. Disponível em: https://github.com/mverardo/CATemplates, 2019.

WOLFRAM, S. A New Kind of Science. [S.l.]: Wolfram Media Inc., 2002.

WOLFRAM RESEARCH. Wolfram Mathematica 11.3. Disponível em http://www.wolfram.com/mathematica/, 2018.

Contatos: ellias_matheus@hotmail.com e pedrob@mackenzie.br