The difference operation between templates of binary cellular automata

Zorandir Soares[2], Maurício Verardo[2], and Pedro Paulo Balbi de Oliveira[12]

[1] Universidade Presbiteriana Mackenzie
Faculdade de Computação e Informática
[2] Pós-Graduação em Engenharia Elétrica e Computação
Rua da Consolação 896, Consolação
01302-907 São Paulo, SP - Brazil
zorandir@gmail.com
mauricio.verardo@gmail.com
pedrob@mackenzie.br

Abstract. The templates consist in a notation that has recently been introduced to represents families of cellular automata that share some property. This paper introduces the operation of difference between templates. This operation is applicable to representation of binary cellular automata families that share some properties but not others. This papers also introduce a process called *exception templates* that is required to this operation. Experimental results and examples of both techniques defined above are illustrated with examples in the space of elementary cellular automata.

Keywords: cellular automata, rule space, templates, difference between templates, exception templates.

1 Introduction

Cellular automata (CAs) are dynamical systems typically discrete in time, space and states. CAs can produce behavior of great complexity based on simple rules of local action [1]. The study of classical problems of cellular automata, as the problem of parity [2] and the density problem [3], can help in the understanding of how this complex behavior emerges.

The most basic approach to find a suitable solution to classic problems consist in test each of rules of a particular family of CAs in order to see if any of these rules are able to solve the problem. However, this approach shows to be inefficient or impractical for a large number of families in CAs, which is the most usual panorama in literature.

Evolutionary computing is constantly used to deal with larger families. This type of algorithm have proven itself very effective in density classification problem [4].

Another strategy to find suitable rules is restricting the search space by rules that have a given property. To represent a subspace with a particular property without the need of enumerate all the rules contained therein can use *templates* idea, as proposed in [5, 6].

There are a lot of operations that can be applied to templates such as expansion and intersection. In this paper we introduce the operation named *difference between templates* and the operation that generates the *exception templates*. Moreover, application of these operations are presented in this paper with practical examples.

This paper is organized in sections. The next section, 2, presents some basic concepts about cellular automata. Following, Section 3 presents more details about templates and its major operations. The Section 4 introduces the operation of difference between templates and the operation that generates what we call the *exception templates*, as well as their applications. Finally, the Section 5 makes closing remarks about the news algorithm.

2 Cellular Automata

Cellular Automata are simple mathematical idealizations of natural systems [7]. They consists of a lattice of discrete fields usually identical, called cells, each of which can take on a finite set of generally integers. The values (or states) of the cells evolve in discrete time steps usually according to deterministic rules that specify the state of each cell according to the states of their neighboring cells [7].

Conventionally, it is assumed that a cellular automaton cells have k states which are represented by integer values in the range of $[0, k-1]$ The state of a cell is changed by the local function of the automaton (a rule), formed by the state transitions set of a cell, based on its current state and in the states of adjacent cells. To update the values of a cell by local functions, usually a radius of action r is defined and r is the number of adjacent cells to be analyzed in each direction by local functions.

A family of cellular automata, or space of cellular automata, is defined by the radius r and the number of states k. One-dimensional cellular automata with $r = 1$ e $k = 2$ are called the family of elementary cellular automata.

To refer to the rules of a space, it is common to use the number obtained by the outputs decimal representation of the state transitions, sorting the lexicographically neighborhoods from largest to smallest state; for instance, in the elementary space, the number of the rules corresponds to the decimal sequence formed by the 8-bit output, arranged from the neighborhood 111 to 000.

The number of rules in a space is provided by $k^{k^{2r+1}}$, making it easy to realize that any change in the variable k and r generate families with a large amount of rules. An approach to handle this problem is to use static properties. The static properties consists of properties obtained directly from the set of CA state transitions. The use of static properties can highly restrict the initial search space. With templates can be represented a set of rules with certain static property.

However, in order to explain the functioning of templates and its operations it is interesting understand the properties of number conservation, internal symmetry and color blind, because these properties will be used as an example later.

2.1 Number Conservation

Number conservation is a static property that determines that the sum from the states of a particular cellular automata should not change during the evolution of space-time, regardless of the initial configuration.

According to Boccara and Fukś [8], a CA is conservative when each of its local function f, applied in every neighborhood $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ satisfy the conditions described in Eq. 1.

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

2.2 Internal symmetry

To grasp the internal symmetry property, the comprehension of the concept of dynamic equivalence of rules is needed. The following explanations are valid for binary rules, although it is possible to generalize to k states.

Given a transitions table of an CA, there are three changes that can be applied and results in a dynamically equivalent behavior to CAs: *reflection*, *combination* and *composition*. Reflection is the transformation obtained reflecting the bits of the neighborhoods transitions table. The conjugation is obtained by reversing all the states of the cells of the transitions table. Finally, the composition is obtained making reflection and conjugation, regardless of the order.

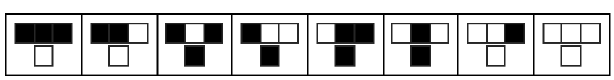

Fig. 1. Transitions table of the rule 60 of the elementary space.

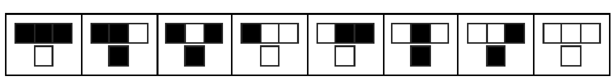

Fig. 2. Transitions table of the rule 102 from the elementary space, obtained through transformation by reflection applied to the transitions table of rule 60.

To illustrate these dynamic equivalence as well the transformations, consider the transition table of Rule 60 of the elementary space, as shown in Figure 1. When applying the transformation by reflection in rule 60 we obtain the Rule 102 of the elementary space, illustrated in Figure 2.

The internal symmetry is defined by the number of state transitions which remain identical after applied a specific transformation. For example, the rule 60 from the elementary CAs has a value of internal symmetry by reflection equal to 4, because it shares the state transitions $((1,1,1),0)$, $((1,0,1),1)$, $((0,1,0),1)$ e $((0,0,0),0)$ with the resulting rule of its transformation by reflection, the rule 102.

2.3 Color blind cellular automata

A CA is considered color blind when it is invariant to the application of permutations in the states of transition table [9]. Naturally, there is generally $k!$ permutations, but only one in the binary case, which can be described as $\{0 \to 1,\ 1 \to 0\}$.

The property color blind is directly related to transformation by conjugation, and the color blind rules have maximum internal symmetry by conjugation.

3 Templates

Template is a generalization of the state transition tables of CAs through variables. A template can represents a rule set, which in turn can presents particular property. The templates were created and implemented by De Oliveira and Verardo [5] and are currently available at the library *open source CATemplates* [10] on GitHub.

Formally, a *template* is an n-tuple formed by k^{2r+1} items, wherein each item i represents a function $g_i(x_{k^{2r+1}-1}, x_{k^{2r+1}-2}, \dots, x_1, x_0)$. The variables x_i can assume any state between 0 and $k-1$. Can be limited the possible values of x_i by the notation $x_i \in C$, in which C is a set representing the possible values of x_i.

As an example, the template of the elementary space $T_1 = (1,1,1,1,1-x_1,x_2,x_1,0)$ represents all the rules which have at position 0 (always from right to left) state 0, at positions 4, 5, 6 and 7 the state 1, at positions 1 and 2 any state in the interval $[0, k-1]$ and in position 3 the complementary state to the value of position 1. Therefore the template T_1 represents the set of elementary cellular automata in its decimal form $\{248, 242, 252, 246\}$ or in binary form as presented below:

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

The process of finding all the rules represented by a template is called *expansion*. There are implemented in the library *CATemplates* [10] several generators algorithms of templates representing rules with certain property. The template $T_{comp} = (1-x_0, 1-x_4, 1-x_2, x_4, 1-x_1, x_2, x_1, x_0)$ of the elementary CAs, for example, when expanded generates only rules that have a maximum internal symmetry by composition. But the template $T_{inv} = (1-x_0, 1-x_1, 1-x_2, 1-x_3, x_3, x_2, x_1, x_0)$ of the elementary CAs expands only the rules with color blind property.

Besides the possibility of expansion of templates, it has been developed by De Oliveira e Verardo [6] an algorithm to generate templates that represent the intersection between two sub-spaces of rules represented by templates. To illustrate, consider the search for a rule set that are both color blind and have maximum internal symmetry by composition. To find this rule set is necessary to perform two steps: first one must match the two templates that represent the desired properties, thus forming an equation system; then to solve the equations, relationships are generated between the variables that, when applied to templates received as input will result in the template representing the intersection.

As an example, consider the template T_{comp} representing rules with a maximum symmetry by composition and the template T_{inv}representing color blind rules, both with $k = 2$ and $r = 1$. The first step to finding the intersection between these two template is equate them generating the equation system represented by Eq. 2:

$$\begin{cases} 1-x_0 = 1-x_0 \\ 1-x_4 = 1-x_1 \\ 1-x_2 = 1-x_2 \\ \quad x_4 \quad = 1-x_3 \\ 1-x_1 = \quad x_3 \\ \quad x_2 \quad = \quad x_2 \\ \quad x_1 \quad = \quad x_1 \\ \quad x_0 \quad = \quad x_0 \end{cases} \tag{2}$$

Then, the system is solved yielding the solution set $S = \{x_3 = 1 - x1, x_4 = x_1\}$. This solution set is then applied as a set of replacements to the templates received as parameter. If the templates received as parameter does not show variables constraints, the result of both replacements can be chosen, ending the process and resulting in the template $(1-x_0, 1-x_1, 1-x_2, x_1, 1-x_1, x_2, x_1, x_0)$, which is the intersection of templates T_{comp} and $T_{inv}$and, therefore, all the rules of the elementary space which at the same time is color blind and present maximum symmetry by composition.

If at least one of the templates presents variables with restriction, a second step should be carried out. Therefore, the expressions that restrict the variables are extracted, generating a set which is then translated into a new equation system to be solved, whose solution set is then applied as a set of replacements in the templates passed as parameter.

Inspired by the intersection operation, we introduce in this article the operation of difference between templates.

4 Difference Between Templates and Exception Templates

The difference operation has two templates as an input parameter, which we will call $T_{minuend}$ and $T_{subtrahend}$. This operation output will result in a set of templates that represents all the rules represented by the template $T_{minuend}$ and are not represented by the $T_{subtrahend}$.

The difference operation is a process with several steps. The first consists in a process of intersection between the two selected templates set as parameters, resulting in a template T_i. If there is no intersection between the two original template, the result of the difference is the actual $T_{minuend}$. If there intersection, the intersection of template T_i is matched to the template $T_{minuend}$, thereby generating logical combinations of equations. Then the algorithm removes the tautological equations and applies a negation operation in the equation, which in the binary case is to just make the permutations $\rho = (0 \to 1, 1 \to 0)$ to the result of the equations. Then, the logical operator is exchanged from $\wedge$ to $\vee$ and the system generated by this process is solved, thereby generating a set of replacement sets that should be applied to the template $T_{minuend}$. If the replacement set is empty or invalid (ie, reference non existing rules in the space, as exemplified below), all the rules pertaining to $T_{minuend}$ also belong to the $T_{subtrahend}$ and, therefore, the algorithm returns an empty set.

To better understand the process, consider the template that represents the color blind rules $T_{inv} = (1-x_0, 1-x_1, 1-x_2, 1-x_3, x_3, x_2, x_1, x_0)$ and number conserving templates $T_{con} = (1, 1+x_2-x_3, 1-x_2, 1-x_1-x_2, x_3, x_2, x_1, 0)$, both of the elementary space. The first step to find the difference from T_{inv} to T_{con} is to make the intersection between them, which results in $T_i = (1, 1-x_1, 1-x_2, 1-x_1-x_2, x_1+x_2, x_2, x_1, 0)$. As the intersection is not null, the next step is to equate T_{inv} with T_i, which generates the equations system represented by Eq. 3:

$$\begin{cases} 1-x_0 = & 1 \\ 1-x_1 = & 1-x_1 \\ 1-x_2 = & 1-x_2 \\ 1-x_3 = & 1-x_1-x_2 \\ x_3 = & x_1+x_2 \\ x_2 = & x_2 \\ x_1 = & x_1 \\ x_0 = & 0 \end{cases} \tag{3}$$

However, this system must be represented by logical combinations of equation, as shown in Eq. 4:

$$\begin{gathered} 1-x_0 = 1 \wedge 1-x_1 = 1-x_1 \wedge 1-x_2 = 1-x_2 \wedge \\ 1-x_3 = 1-x_1-x_2 \wedge x_3 = x_1+x_2 \wedge x_2 = x_2 \wedge x_1 = x_1 \wedge x_0 = 0 \end{gathered} \tag{4}$$

Are eliminated then all tautological equations and is exchanged logical operators $\wedge$ to $\vee$, resulting in the system represented by Eq. 5:

$$1 - x_0 = 1 \vee 1 - x_3 = 1 - x_1 - x_2 \vee x_3 = x_1 + x_2 \vee x_0 = 0 \tag{5}$$

The last step, apply a negation operation on the equations that, in the binary case, corresponding to the permutation ρ or, equivalently, the implementation of the function $f(x) = 1-(x)$. Eq. 6 represents the logical combination of equations resulting from these operations.

$$1 - x_0 = 1 - 1 \vee 1 - x_3 = 1 - (1 - x_1 - x_2) \vee x_3 = 1 - (x_1 + x_2) \vee x_0 = 1 - 0 \tag{6}$$

Is solved then the logical combination from equations, generating the solution set $S = \{\{x_0 \to 1\}, \{x_3 \to -x_1 - x_2 + 1\}\}$. As S has more of a substitution set, both sets must be used to make substitutions in the template $T_{inv} = (1-x_0, 1-x_1, 1-x_2, 1-x_3, x_3, x_2, x_1, x_0)$, finally yielding the set of templates $\{(0, 1-x_1, 1-x_2, 1-x_3, x_3, x_2, x_1, 1), (1-x_0, 1-x_1, 1-x_2, x_1+x_2, 1-x_1-x_2, x_2, x_1, x_0)\}$

In many cases, only the steps described so far are sufficient to make the difference between both templates. But there are cases in which the template $T_{subtrahend}$, corresponding to T_{con} in the given example, the variables has substitutions that lead to invalid rules. This situation occurs, for example, when expanding the template $(1, 1-x_1, 1-x_2, 1-x_1-x_2, x_1+x_2, x_2, x_1, 0)$ by assigning the value 1 to the variables x_1 and x_2. By doing this is obtained in position 3 (from right to left) the value of 2, which is outside the entire range $[0, k-1]$, corresponding therefore to a rule that is not part of the treated space.

To work around this problem, it is necessary that after the first steps of difference operation, also check for exception templates at the intersection of the $T_{minuend}$ and $T_{subtrahend}$, ie, templates which represent a set of substitutions that take the template passed as parameter to display substitutions outside the range $[0, k-1]$.

For example, we will continue the difference operation between T_{inv} and T_{con}; to this end, consider the template $T_i = (1, 1-x_1, 1-x_2, 1-x_1-x_2, x_1+x_2, x_2, x_1, 0)$ the intersection between them, for $k = 2$. The first step of the difference operation occurs normally and generates the templates:

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

It is trivial to realize that any expansion of T_i having the set of substitutions $\{x_1 = 1, x_2 = 1\}$ will cause the position 3 and 4 of the template display values that do not belong to the interval $[0, k-1]$. Hence all templates that have $\{x_1 = 1, x_2 = 1\}$ are complementary to the template T_i. So it will generate the template $(x_7, x_6, x_5, x_4, x_3, 1, 1, x_0)$, which is the exception template of T_i.

Therefore every rule represented by the template $T_{minuend} - T_{inv}$ in the given example – and which is also represented by the exception template from the the intersection of $T_{minuend}$ with $T_{subtrahend} - T_i$ in this example - - should be represented by at least one of the templates resulting from the difference operation. For this, the algorithm that finds the difference between templates considers all exception templates found, intersects them with the $T_{minuend}$ and adds them to the set of templates obtained by the first steps of the difference operation. Thus, for the example presented above, the set of difference templates resultant is represented below:

$$\{(0, 1-x_1, 1-x_2, 1-x_3, x_3, x_2, x_1, 1),$$
$$(1-x_0, 1-x_1, 1-x_2, x_1+x_2, 1-x_1-x_2, x_2, x_1, x_0),$$
$$(1-x_0, 0, 0, 1-x_3, x_3, 1, 1, x_0)\}$$

One advantage of the difference operation between templates is with which it is possible to find answers to various non-trivial question, for example, determine which rules have the numerical conservability property, but do not present at the same time maximum symmetry by composition and color blind property. To answer the question simply pick the template of conservatives rules, and then subtract from it the intersection between the rules template with maximum symmetry by composition and the color blind template. In this particular case, the result is a set of templates representing all the conservative rules except the identity rule.

It is worth noting that the set of templates returned does not present a smaller search space than the the template $T_{minuend}$; however, the operation is able to represent the difference between two templates without the need to perform the expansion. Subsequently the resulting templates can be used by other operations, and this is the main advantage of the operation.

5 Final Remarks

This paper describes the templates of cellular automata, introduces the difference operation and show an approach to find exception templates. Both operations could prove valuable in many situations involving efforts to find rules in a large space, particularly in the classic problems of parity and density classification.

Also, it is shown the possibility of obtain non-trivial answers to questions about searches for rules with certain static properties by means of templates, without relying on the use of search algorithms or enumerative reviews in a whole family of CAs.

It is noteworthy that the difference operation can generate a large number of templates, which can, in specific cases, not effectively reduce the search space. But the approach is relevant in general terms, and so more effective the greater the complexity of the attributes specified in the rules, ie, the greater the amount of property that the rules space in question should either not present. Though it was possible to specify the desired properties by means of templates, by the previous work [5, 6], with the present work will be possible also specifying unwanted properties.

For now the difference operation between templates has been implemented just to the families of CAs with $k = 2$, but its generalization to higher values of k is underway.

Acknowledgements

The authors thank Mackpesquisa, the Fapesp, and federal agencies CAPES and CNPq for different forms of support received during the development of this work.

References

1. Wolfram, S.: A new kind of science. Volume 5. Wolfram media Champaign (2002)
2. Betel, H., De Oliveira, P.P.B., Flocchini, P.: Solving the parity problem in one-dimensional cellular automata. Natural Computing **12** (2013) 323–337
3. De Oliveira, P.P.B.: On density determination with cellular automata: Results, constructions and directions. Journal of Cellular Automata **9** (2014) 357–385
4. Wolz, D., De Oliveira, P.P.B.: Very effective evolutionary techniques for searching cellular automata rule spaces. J. Cellular Automata **3** (2008) 289–312
5. De Oliveira, P.P.B., Verardo, M.: Representing families of cellular automata rules. The Mathematica Journal **16** (2014)
6. De Oliveira, P.P.B., Verardo, M.: Template based representation of cellular automata rules. In Isokawa, T., Imai, K., Matsiu, N., Peper, F., Umeo, H., eds.: 20th International Workshop on Cellular Automata and Discrete Complex Systems, Himeji, Japão, Julho 7-9 (2014) 199–204
7. Wolfram, S.: Cellular automata and complexity: collected papers. Volume 1. Addison-Wesley Reading (1994)
8. Boccara, N., Fukś, H.: Number-conserving cellular automaton rules. Fundamenta Informaticae **52** (2002) 1–13
9. Salo, V., Törmä, I.: Color blind cellular automata. In: Cellular Automata and Discrete Complex Systems. Springer (2013) 139–154
10. Verardo, M., De Oliveira, P.P.B.: CATemplates. (2015) https://github.com/mverardo/CATemplates.