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. A cellular automata template is a generalization of the state transition tables of cellular automata. Templates has recently been introduced to represent families of cellular automata that share a given property. This paper introduces the operation of difference between templates, used to find cellular automata that have a given property but lack another one. This paper also introduce a process called *exception templates* that is required to this operation. Experimental results of both techniques 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, like the parity problem [2] and the density classification problem [3] can help to understand how this complex behavior emerges.

The most basic approach to find a suitable solution to classic problems consists in testing each of the 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 larger families of CAs, which is the most usual panorama in literature.

Evolutionary computing is constantly used to deal with larger families. This type of algorithm has proven very effective to find solutions for the 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 of the space rules, one can use *templates*, as proposed by De Oliveira and Verardo [5, 6].

There are two operations that can be applied to templates: *expansion* and *intersection* [5, 6]. In this paper we introduce a new 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 consist of a lattice of discrete cells, each of which can be a state of a finite set of states in a given time step. 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 r is defined as the number of adjacent cells to be analyzed in each direction by local functions.

A family (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 space 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.

In other words, as neighbourhood are sets of adjacent cells that influence in the state of the current cell in the next time step, all CA has a rule that relates each possible neighborhood with its state after a time step. This rule can be represented by rule tables. Rule tables are n-tuples, where $n = 2r + 1$ in one-dimensional case, where the elements of that tuple are all possible neighborhoods and their respective state transitions.

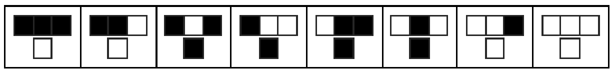

Fig. 1. Rule table of the elementary CA 60.

For example, consider the rule table of the elementary CA 60, as shown below:

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

This rule table can also be illustrated as in Figure 1, that is the iconic representation of rule table of the elementary CA 60.

The number of rules in a space is provided by $k^{k^{2r+1}}$, making it easy to realize that any increase in the values of k and r generates families with a large amount of rules. An approach to handle this problem is to use static properties as a hint on how the CA will behave during its evolution. A static property is obtained directly from the CA's rule table. 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.

For present purposes, whenever we refer to a CA template in this paper, we mean one-dimensional and binary CA template.

In order to explain how templates work, it is important to understand the properties of number conservation, internal symmetry and color blindness, that will be used in later examples.

2.1 Number Conservation, Internal Symmetry and Color Blind Cellular Automata

Number conservation is a static property that determines that the sum of the states of a particular cellular automata should not change during the CA's evolution, regardless of the initial configuration.

According to Boccara and Fukś [8], a CA is conservative when its local function f, applied to every neighborhood $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$, considering $n = 2r+1$, satisfies the conditions described in Eq. 1.

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

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

Given the rule table of a CA, there are three transformations that can be applied to it, such that the resulting rule table is represents an equivalent rule to the original CA: *reflection*, *conjugation* and *composition*. Reflection is the transformation obtained by mirroring the bits of the neighborhoods in the rule table. The conjugation is obtained by reversing all the states of the cells of the rule table. The last one is obtaines by composing both conjugation and reflection, regardless of the order.

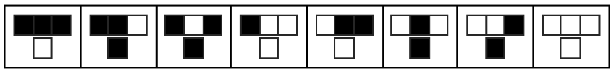

Fig. 2. Rule table of elementary CA 102 obtained through transformation by reflection applied to the rule table of elementary CA 60.

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

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

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

Then, 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 $\{0 \to 1,\, 1 \to 0\}$.

3 Templates

A template is a generalization of the state transition tables of CAs through variables or functions. A template represents a set of CAs which might share of a given static property. Templates were created and implemented by De Oliveira and Verardo [5] and are currently available in the open source library *CATemplates* [10] on GitHub.

Formally, a *template* is an n-tuple formed by k^{2r+1} items, wherein each item i represents a function $g_i(x_{k^{2r+1}-1}, x_{k^{2r+1}-2}, \ldots, x_1, x_0)$. By default, a template variable x_i can assume any state between 0 and $k-1$, but their values can be limited 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 several generator algorithms of templates representing rules with certain property implemented in the library *CATemplates* [10].

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 reflection and by conjugation. But the template $T_{cb} = (1 - x_0, 1 - x_1, 1 - x_2, 1 - x_3, x_3, x_2, x_1, x_0)$ 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 generates templates that represent the intersection between two sub-spaces of rules represented by distinct templates. To illustrate, consider the search for a rule set composed by CAs that are both color blind and have maximum internal symmetry by composition of reflection and conjugation. To find this rule set, it is necessary to perform two steps: first one must match the two templates that represent the desired properties, thus forming an equation system. When the equation system is solved, 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_{cb} representing color blind rules. The first step to find the intersection between these two template is equate them generating the equation system represented by Eq. 2:

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

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_{cb} and, therefore, represents all the rules of the elementary space which are 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_{int}. If there is no intersection between the two original template, the result of the difference is the actual $T_{minuend}$. If there is an intersection, it 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 $T_{cb} = (1-x_0, 1-x_1, 1-x_2, 1-x_3, x_3, x_2, x_1, x_0)$ 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 generated using the library *CATemplates*. The first step to find the difference from T_{cb} to T_{con} is to make the intersection between them, which results in $T_{int} = (1, 1-x_1, 1-x_2, 1-x_1-x_2, x_1+x_2, x_2, x_1, 0)$. As the intersection is not null, the next step is to equate T_{cb} with T_{int}, 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 the equations, as shown in Eq. 4:

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

All tautological equations are then eliminated, and the $\wedge$ are switched 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 is to 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.

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

The equations are solved, generating the solution set $S = \{\{x_0 \to 1\}, \{x_3 \to -x_1-x_2+1\}\}$. As S has more than one item, both solutions must be used to make substitutions in the template $T_{cb} = (1-x_0, 1-x_1, 1-x_2, 1-x_3, x_3, x_2, x_1, x_0)$, 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, 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. This expansions yields a 2 at position 3 (from right to left) of the template, which is outside of the 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 the difference operation, we 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_{cb} and T_{con}. To this end, consider the template $T_{int} = (1, 1 - x_1, 1 - x_2, 1 - x_1 - x_2, x_1 + x_2, x_2, x_1, 0)$ the intersection between them, for $k = 2$.

The first step of the difference operation occurs normally and generates the templates:

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

In this case, any expansion of T_{int} 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_{int}. 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_{int}.

Therefore every rule represented by the template $T_{minuend} - T_{cb}$ in the given example – and which is also represented by the exception template from the the intersection of $T_{minuend}$ with $T_{subtrahend} - T_{int}$ 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:

$$\begin{aligned} &\{(0, 1 - x_1, 1 - x_2, 1 - x_3, x_3, x_2, x_1, 1), \\ (1 - x_0, 1 - x_1, 1 - x_2, &x_1 + x_2, 1 - x_1 - x_2, x_2, x_1, x_0), \\ &(1 - x_0, 0, 0, 1 - x_3, x_3, 1, 1, x_0)\} \end{aligned}$$

One advantage of the difference operation between templates is that it is possible to find answers to various non-trivial questions, 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 – introduced by De Oliveira e Verardo [5, 6] –, introduces the difference operation and shows 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, we show the possibility to 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 the work done in [5, 6] made possible to specify templates of desired properties, the present work makes possible to find templates that exclude unwanted properties.

Currently, the difference operation works with binary CAs, 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.