Representing Families of Cellular Automata Rules

Pedro P. B. de Oliveira
Maurício Verardo

This article introduces the notion of a representation of cellular automata rules based on a template. This enhances the standard representation based on a rule table, in that it refers to families of cellular automata, instead of a rule alone. The key for obtaining the templates is the role of the built-in equation-solving capabilities of *Mathematica*. Operations applicable to the templates are defined, and examples of their use are given in the context of finding representations for rule sets that share the properties of maximum internal symmetry or number conservation. The perspectives for using templates in further contexts are also discussed and current limitations are addressed.

1. Introduction

A cellular automaton (CA) is a dynamical system with arbitrarily complex global behavior, despite being governed by very simple local rules [1]. In order to better understand how that kind of complex behavior emerges, many explorations have been made in the context of the power implicit in CA rules. For instance, classical benchmark problems have been used for this, including the density classification task [2, 3] and the parity problem [4]. The density classification task tries to discover the most frequent bit in the initial configuration of the lattice; the parity problem tries to find the parity of the number of 1s in the initial configuration of the lattice. One of the approaches in these contexts is to evaluate every possible CA of a given family in terms of its capabilities to solve the target problem. This approach is possible in small CA families, like the elementary space (composed of 256 CAs), but is not feasible in larger families, like the one-dimensional binary CA family with radius 3, composed of 2^{128} rules.

As a strategy to search for CAs in large rule families, evolutionary computation has been extensively used, relying on measures of properties of the candidate rules, such as their degree of internal symmetry, so as to discard or keep candidates according to these property values. This was a key aspect, for instance, that led to finding WdO, currently the best one-dimensional radius-3 rule for the density classification task [5].

An alternative is to constrain the search space to only the CAs that are known to present specific properties. The challenge here is how to constrain the space without the need to enumerate the entire subspace of interest. Here, we introduce the concept of a CA template as a possible way to achieve this goal. A CA template is a data structure associated with the rule tables of the members of a CA family that relies on the use of variables. The introduction of these variables makes it possible for a CA template to represent a set of rules, unlike the standard k-ary rule table representation that can only represent one individual CA. By making use of *Mathematica*'s built-in equation-solving capabilities and algorithms that allow finding equality relations among CAs with a given property, we are able to create templates that represent number-conserving CAs (those that, in a sense, preserve the number of states of the initial configuration; more details below), as well as those with maximal internal symmetry (those displaying invariance under some transformations in their rule tables; also to be explained below). These two cases are given here as examples of the applicability of the template idea, but other properties can also be accounted for.

In the following section, basic notions about CAs are given, followed by a section that presents details about important properties related to the density classification task. Section 4 explains the notion of *template* and presents the implemented algorithms. Section 5 concludes the text, with a discussion on the advantages and limitations of using templates, and gives some ideas for future work.

■ 2. Cellular Automata

Cellular automata constitute a class of decentralized dynamical systems, usually discrete in space, time, and states [1]. As systems governed by relatively simple rules, CAs represent a meaningful model for tackling the issue of how interaction among simple components can lead to the solution of global problems.

CAs are composed of a regular lattice of cells whose states change through time, according to a local rule. The lattice can be deployed in any number of dimensions (most commonly one, two, or three) and may have an infinite or fixed number of cells. Cells' states are commonly represented by numbers or colors out of k possibilities ranging from 0 to $k-1$. The local rule of the CA acts on the neighborhood of every cell, which is the set of neighboring cells meant to influence its subsequent states. The neighborhood is usually expressed by its radius (or range) r, meaning the range of cells on each side affecting the one in question. By defining values for these two parameters, a CA rule space or family is defined. The values of $r = 1$ and $k = 2$ in the one-dimensional case (i.e. a neighborhood has three cells; a cell has two possible states) give rise to the elementary rule space, which is the most well-studied family, due to its small size of only 256 rules but extremely rich phenomenology [1].

For present purposes, whenever we refer to cellular automata, we mean one-dimensional, binary ($k = 2$) CAs, with a fixed number of cells in the lattice and periodic boundary conditions (i.e. the lattice is closed at its ends, like a ring).

Every CA is governed by a rule that relates the neighborhood of a cell to the state it takes on at the next time step. Its most common representation is the rule table, which is an explicit listing of every possible state configuration of the neighborhoods, lexicographically ordered, and a corresponding cell state for each. Here we use Wolfram's lexicographical ordering, where the leftmost neighborhood is formed by the neighborhood configuration where all cells are in the $(k-1)$ state, all the way down to the rightmost neighborhood with all cells in the 0 state.

As an illustration, this is the rule table of the elementary CA for rule 184.

```
{{{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}};
```

This is the ordered set of output cell states from that rule table, the *k*-ary form.

```
{1, 0, 1, 1, 1, 0, 0, 0};
```

By converting the binary sequence that defines the *k*-ary form into a decimal representation, one obtains the CA rule number, which serves as a unique identifier of a CA in a given rule space [1].

```
FromDigits[{1, 0, 1, 1, 1, 0, 0, 0}, 2]
```

```
184
```

In order to handle operations concerning rule tables, various *Mathematica* functions are defined. So, given a rule table in its *k*-ary form, the function `RuleTableFromkAry` transforms it to its classical representation.

```
RuleTableFromkAry[kAryRuleTable_, k_Integer: 2, r_: 1] :=
 MapThread[List[#1, #2] &,
  {Tuples[Range[k - 1, 0, -1], ⌊2 r + 1⌋], kAryRuleTable}]
```

```
RuleTableFromkAry[{1, 0, 1, 1, 1, 0, 0, 0}]
```

```
{{{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}}
```

The function `kAryFromRuleTable` reverses the process.

```
kAryFromRuleTable[ruleTable_] := Last /@ ruleTable
```

```
kAryFromRuleTable[{{{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}}]

{1, 0, 1, 1, 1, 0, 0, 0}
```

Given a CA's rule number, RuleTableFromRuleNumber determines its rule table.

```
RuleTableFromRuleNumber[rnum_Integer, k_Integer: 2, r_: 1] :=
 RuleTableFromkAry[PadLeft[IntegerDigits[rnum, k], k^(2*r+1)],
  k, r]

RuleTableFromRuleNumber[184]

{{{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}}
```

The inverse function RuleNumberFromRuleTable yields the rule number from the rule table.

```
RuleNumberFromRuleTable[ruletable_, k_Integer: 2] :=
 FromDigits[kAryFromRuleTable[ruletable], k]

RuleNumberFromRuleTable[
 {{{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}}]

184
```

WellFormedRuleTableQ is a predicate that checks whether a rule table in *k*-ary form is valid according to its values of *r* and *k*.

```
WellFormedRuleTableQ[kAryRuleTable_, k_Integer: 2, r_: 1] :=
 Fold[And[#1, #2] &, True,
  MemberQ[Range[0, k - 1], #] & /@ kAryRuleTable]

WellFormedRuleTableQ[
   kAryFromRuleTable[RuleTableFromRuleNumber[#]]] & /@
 {110, 137, 124, 150, 193}

{True, True, True, True, True}
```

```
WellFormedRuleTableQ[{-1, 0, 1, 0, 1, 0, 1, 2}]
```

```
False
```

`RuleOutputFromNeighbourhood` is a utility function to get the output corresponding to a particular neighborhood in a rule table.

```
RuleOutputFromNeighbourhood[neighbourhood_List,
   kAryRuleTable_, k_Integer: 2, r_: 1] :=
  Extract[kAryRuleTable,
   {k^⌊2 r+1⌋ - FromDigits[neighbourhood, k]}]
```

```
RuleOutputFromNeighbourhood[{1, 1, 1},
 kAryFromRuleTable[RuleTableFromRuleNumber[110]]]
```

```
0
```

Finally, `AllNeighbourhoods` is a utility function giving all possible neighborhoods of a certain rule space.

```
AllNeighbourhoods[k_Integer : 2, r_ : 1] :=
   Tuples[Range[k - 1, 0, -1], ⌊2 r + 1⌋];
```

```
AllNeighbourhoods[2]
```

```
{{1, 1, 1}, {1, 1, 0}, {1, 0, 1}, {1, 0, 0},
 {0, 1, 1}, {0, 1, 0}, {0, 0, 1}, {0, 0, 0}}
```

```
AllNeighbourhoods[2, 2]
```

```
{{1, 1, 1, 1, 1}, {1, 1, 1, 1, 0},
 {1, 1, 1, 0, 1}, {1, 1, 1, 0, 0}, {1, 1, 0, 1, 1},
 {1, 1, 0, 1, 0}, {1, 1, 0, 0, 1}, {1, 1, 0, 0, 0},
 {1, 0, 1, 1, 1}, {1, 0, 1, 1, 0}, {1, 0, 1, 0, 1},
 {1, 0, 1, 0, 0}, {1, 0, 0, 1, 1}, {1, 0, 0, 1, 0},
 {1, 0, 0, 0, 1}, {1, 0, 0, 0, 0}, {0, 1, 1, 1, 1},
 {0, 1, 1, 1, 0}, {0, 1, 1, 0, 1}, {0, 1, 1, 0, 0},
 {0, 1, 0, 1, 1}, {0, 1, 0, 1, 0}, {0, 1, 0, 0, 1},
 {0, 1, 0, 0, 0}, {0, 0, 1, 1, 1}, {0, 0, 1, 1, 0},
 {0, 0, 1, 0, 1}, {0, 0, 1, 0, 0}, {0, 0, 0, 1, 1},
 {0, 0, 0, 1, 0}, {0, 0, 0, 0, 1}, {0, 0, 0, 0, 0}}
```

```
AllNeighbourhoods[3]
```

```
{{2, 2, 2}, {2, 2, 1}, {2, 2, 0}, {2, 1, 2},
 {2, 1, 1}, {2, 1, 0}, {2, 0, 2}, {2, 0, 1},
 {2, 0, 0}, {1, 2, 2}, {1, 2, 1}, {1, 2, 0},
 {1, 1, 2}, {1, 1, 1}, {1, 1, 0}, {1, 0, 2}, {1, 0, 1},
 {1, 0, 0}, {0, 2, 2}, {0, 2, 1}, {0, 2, 0}, {0, 1, 2},
 {0, 1, 1}, {0, 1, 0}, {0, 0, 2}, {0, 0, 1}, {0, 0, 0}}
```

All these functions are handy to perform rule table manipulation and are used throughout this article.

In the one-dimensional case, it is possible to visualize the system's evolution using a space-time diagram, in which time goes from top to bottom, and cell states are represented by colors. For binary CAs, white cells are in the 0 state and black cells in the 1 state. In order to obtain and plot the space-time diagram resulting from a rule execution on a given lattice, one can use *Mathematica*'s built-in functions `CellularAutomaton` and `ArrayPlot`.

■ 3. Cellular Automaton Properties

In order to better understand the computational power implicit in a CA rule, benchmark problems have been defined for it to tackle; among them, the most common is the density classification task (DCT). In the classical definition of DCT, a one-dimensional binary CA has to lead an arbitrary initial odd-sized configuration into a fixed-point state of all blacks, if the initial condition has a larger number of black cells, or into a fixed-point state of all whites otherwise.

It has been proved that in order to solve the DCT perfectly, a CA would need to be number conserving, that is, it should not change the number of cells in each state from any given initial condition [6]. This fact stands as a contradiction against the classical definition of the DCT, since in order for it to evolve to an all-black or all-white configuration, it would obviously need to change the number of cells in each state throughout time. This means that DCT is unsolvable when formulated according to its classical definition [2, 3].

Currently, the best imperfect DCT solver (known as Wd0) was found in [5], by means of a sophisticated evolutionary algorithm that used, among other important properties, the internal symmetry of a rule in its fitness function. In tune with the fact that a perfect DCT solver would need to be number conserving, Wd0 and other good DCT solvers are known to have a very small Hamming distance from number-conserving rules of the same rule space [7].

All in all, number conservation and internal symmetry are two important properties when determining the ability of a CA to solve the DCT, and serve as good examples for the notion of CA templates. Both are described in detail in the following subsections. But notice, upfront, that these two properties are amenable to being addressed in templates, since they derive from well-established relations among state transitions.

□ 3.1 Number Conservation

Number conservation is a property presented by some CAs, in which the sum of the states of the individual cells in any initial configuration does not change during the space-time evolution; in particular, for binary CAs, this means that the number of 1s always remains the same. This kind of CA is useful, for instance, to model systems like car traffic, in which a car cannot appear or disappear as time goes by [7]. Elementary CA 184 is an example of a number-conserving CA.

In order for a one-dimensional CA rule to be number conserving, it is established in [8] that the local rule f with neighborhood size n must respect the following necessary and sufficient conditions for every state transition:

$$f(x_{1,} x_2, x_3, \ldots, x_n) =$$
$$x_1 + \sum_{i=1}^{n-1} f(0_1, \ldots, 0_i, x_2, \ldots, x_{n-i+1}) - f(0_1, \ldots, 0_i, x_1, x_2, \ldots, x_{n-i}),$$

where $0_1, 0_2, \ldots, 0_i$ corresponds to a sequence of 0s of length i.

A simplification of the original algorithm from [8] is provided in [9]. Basically, it was shown that for any given rule, it suffices to analyze the state transitions associated with the neighborhood made up of only 0s and the neighborhoods not starting with 0. This is a total, therefore, of $k^n - k^{n-1} + 1$ neighborhoods instead of k^n, as stated in [8]. This is the condition we employ to obtain templates that represent number-conserving CAs, as will be shown below.

□ 3.2 Internal Symmetry

Apart from number conservation, a rule's internal symmetry also plays an important role in solving the DCT. In order to fully understand how this property works, an explanation about rule transformations and dynamically equivalent rules is required; the presentation is restricted to binary rules, even though this notion extends to the arbitrary k-ary case.

Given the rule table of a CA, one can apply three types of transformations on it that will result in dynamically equivalent rules. For the binary case, `BlackWhiteTransform` is obtained by switching the state of all cells in a rule table. The second type of transformation, `LeftRightTransform`, is obtained by reversing the bits of the neighborhoods in a rule table and reordering the set of state transitions. The composition in either order of the latter two transformations (they commute) yields the third type, `LeftRightBlack⋱WhiteTransform` or `BlackWhiteLeftRightTransform`.

```
BlackWhiteTransform[ruleTable_] :=
 Reverse[{1 - #[[1]], 1 - #[[2]]} & /@ ruleTable]

LeftRightTransform[ruleTable_] :=
 Reverse[SortBy[{Reverse[#[[1]]], #[[2]]} & /@ ruleTable, First]]
```

```
LeftRightBlackWhiteTransform[ruleTable_] :=
 LeftRightTransform[BlackWhiteTransform[ruleTable]]

BlackWhiteLeftRightTransform[ruleTable_] :=
 BlackWhiteTransform[LeftRightTransform[ruleTable]]
```

Here is how they work on rule 110.

```
RuleTableFromRuleNumber[110]

{{{1, 1, 1}, 0}, {{1, 1, 0}, 1}, {{1, 0, 1}, 1}, {{1, 0, 0}, 0},
 {{0, 1, 1}, 1}, {{0, 1, 0}, 1}, {{0, 0, 1}, 1}, {{0, 0, 0}, 0}}

BlackWhiteTransform[RuleTableFromRuleNumber[110]]

{{{1, 1, 1}, 1}, {{1, 1, 0}, 0}, {{1, 0, 1}, 0}, {{1, 0, 0}, 0},
 {{0, 1, 1}, 1}, {{0, 1, 0}, 0}, {{0, 0, 1}, 0}, {{0, 0, 0}, 1}}

LeftRightTransform[RuleTableFromRuleNumber[110]]

{{{1, 1, 1}, 0}, {{1, 1, 0}, 1}, {{1, 0, 1}, 1}, {{1, 0, 0}, 1},
 {{0, 1, 1}, 1}, {{0, 1, 0}, 1}, {{0, 0, 1}, 0}, {{0, 0, 0}, 0}}

LeftRightBlackWhiteTransform[RuleTableFromRuleNumber[110]]

{{{1, 1, 1}, 1}, {{1, 1, 0}, 1}, {{1, 0, 1}, 0}, {{1, 0, 0}, 0},
 {{0, 1, 1}, 0}, {{0, 1, 0}, 0}, {{0, 0, 1}, 0}, {{0, 0, 0}, 1}}

BlackWhiteLeftRightTransform[RuleTableFromRuleNumber[110]]

{{{1, 1, 1}, 1}, {{1, 1, 0}, 1}, {{1, 0, 1}, 0}, {{1, 0, 0}, 0},
 {{0, 1, 1}, 0}, {{0, 1, 0}, 0}, {{0, 0, 1}, 0}, {{0, 0, 0}, 1}}
```

This checks the first one, `BlackWhiteTransform`.

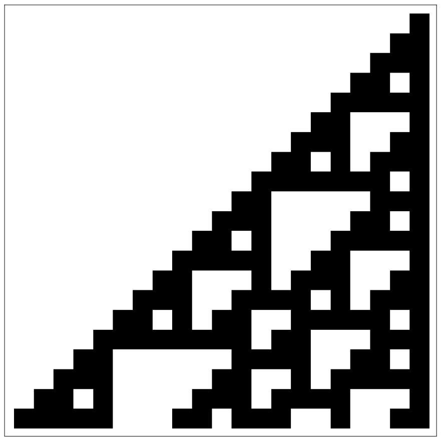
```
kAryFromRuleTable@RuleTableFromRuleNumber[110]

{0, 1, 1, 0, 1, 1, 1, 0}

Reverse[1 - %]

{1, 0, 0, 0, 1, 0, 0, 1}
```

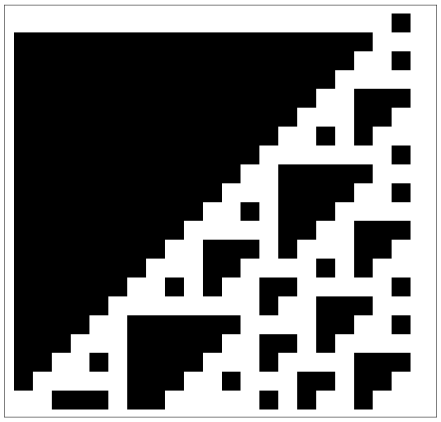
With these transformations, it becomes straightforward to see which CAs in a given space have equivalent dynamical behavior. For instance, by applying the three transformations on a given CA, say elementary rule 110, elementary rules {137, 124, 193} are obtained. These four rules are said to be in the same dynamical equivalence class. It is easy to see why, by looking at their space-time diagrams.

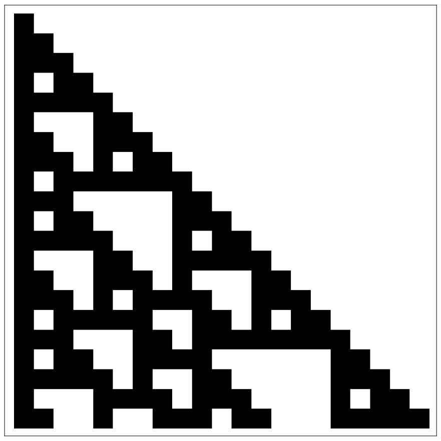
```
With[{rule110 = RuleTableFromRuleNumber[110]},
 {RuleNumberFromRuleTable[BlackWhiteTransform[rule110]],
  RuleNumberFromRuleTable[LeftRightTransform[rule110]],
  RuleNumberFromRuleTable[
   BlackWhiteTransform[LeftRightTransform[rule110]]]}]
```

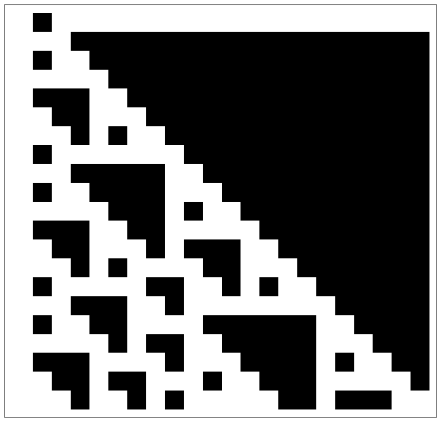
```
{137, 124, 193}
```

```
Grid[Partition[Labeled[
      ArrayPlot[CellularAutomaton[#〚2〛, {{1}, 0}, 20]],
      Text@Row[{#〚1〛, ": ", #〚2〛}]
] & /@ {{"Original", 110}, {"Black-White", 137},
     {"Left-Right", 124}, {"Composed", 193}}, 2]]
```


Original: 110


Black-White: 137


Left-Right: 124


Composed: 193

By comparing the rule table of a CA with the one that resulted from its equivalent rule obtained out of a given transform, it is possible to count the number of state transitions they share. In a sense, this provides a measure of the amount of *internal symmetry* of a CA with respect to that transformation, whichever it is. For instance, elementary CA 110 has an internal symmetry value of 2 with respect to the black-white transformation, since it shares two state transitions with its black-white symmetrical rule, which is elementary rule 137.

```
With[{
   rule110 = RuleTableFromRuleNumber[110],
   rule137 = RuleTableFromRuleNumber[137]
  },
  Total@MapThread[If[#[[2]] == #2[[2]], 1, 0] &,
    {rule110, rule137}]]
```

```
2
```

Repeating this process with rule 150, on the other hand, yields a different result. Rule 150 has an internal symmetry value of 8 according to the black-white transformation. This is the maximum possible value of this measure with elementary CAs. This is quite predictable, as the black-white transformation of rule 150 is rule 150 itself. In fact, any of the three transformations applied to rule 150 yields rule 150 itself, indicating it has the maximum internal symmetry value according to any of the three transformations.

```
RuleNumberFromRuleTable[
 BlackWhiteTransform[RuleTableFromRuleNumber[150]]]
```

```
150
```

```
RuleNumberFromRuleTable[
 LeftRightTransform[RuleTableFromRuleNumber[150]]]

150

With[{rule150 = RuleTableFromRuleNumber[150]},
 Total@MapThread[If[#⟦2⟧ == #2⟦2⟧, 1, 0] &,
   {rule150, rule150}]]

8
```

The degree of internal symmetry of a rule can be a relevant measure in any context where a property is shared among all members of a class of dynamical equivalence. In [5] and [7], for instance, rules with maximal internal symmetry with the composite transformation were key for their findings related to DCT.

■ 4. Cellular Automata Templates

A CA template is an enhancement over the rule table representation, obtained by allowing it to have variables in the place of simple cell states as its results. As a consequence, a CA template has the power to represent whole subsets of CA rule spaces, instead of only a single rule.

As a simple example, consider the template $\{0, 1 - x_1, 0, 1, x_2, 1, x_1, 0\}$. It represents the subset of the elementary CAs with fixed bits at positions 1, 3, 5, 6, and 8 in the list, free variables at positions 2 and 4, and complement bits at positions 2 and 7.

Using *Mathematica*'s built-in transformation rules, one can obtain the four CAs represented by this template, as well as their corresponding rule numbers.

```
With[
 {
  transformationRules =
   {{x1 → 0, x2 → 0}, {x1 → 0, x2 → 1}, {x1 → 1, x2 → 0},
    {x1 → 1, x2 → 1}},
  template = {0, 1 - x1, 0, 1, x2, 1, x1, 0}
 },
 template /. transformationRules
]

FromDigits[#, 2] & /@ %

{{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}}

{84, 92, 22, 30}
```

The function `RuleTemplateVars` lists the variables in a template.

```
RuleTemplateVars[template_List] :=
 Union[Select[template, Head@# === Symbol &]]

(*RuleTemplateVars[template_List]:=
 Union[If[Length[#]===0,#,#[[2]]]&/@
   Select[
    Flatten[
     If[Length[#]===0,#,Table[#[[i]],{i,1,Length[#]}]]&/@
      template],¬NumericQ[#]&]]*)

RuleTemplateVars[{0, 1 - x1, 0, 1, x2, 1, x1, 0}]

{x1, x2}
```

Extracting the variables from a template and applying a value to each, the template is transformed into one of its represented rule tables. Every template has a number of possible substitutions equal to $k^{\texttt{Length[RuleTemplateVars[template]]}}$; however, as will be seen later, some of those may not be valid.

The function `ExpandTemplate` performs this operation by applying values to each variable of a given template. It may receive as an optional argument an integer called `ithSubstitution` in the range 0 to $k^{\texttt{Length[RuleTemplateVars[template]]}}$, representing which substitution should be made. If omitted, it performs all the possible substitutions for a given template.

```
ExpandTemplate[template_List, k_Integer: 2] := With[
  {substitutionRange =
    Range[1, k^Length[RuleTemplateVars[template]]]},
  ExpandTemplate[template, k, #] & /@ substitutionRange
 ]

ExpandTemplate[template_List, k_Integer: 2,
  ithSubstitution_Integer] := Module[
  {templateVariables, substitutions, transformationRules},
  templateVariables = RuleTemplateVars[template];
  substitutions = IntegerDigits[ithSubstitution, k,
    Length[templateVariables]];
  transformationRules =
   MapThread[(#1 → #2) &,
    {templateVariables, substitutions}];
  template /. transformationRules
 ]
```

```
ExpandTemplate[{0, 1 - x1, 0, 1, x2, 1, x1, 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}, {0, 1, 0, 1, 0, 1, 0, 0}}
```

After the expansion, one can obtain the list of valid rules represented by the template by using the function `RuleNumbersFromTemplate`.

```
RuleNumbersFromTemplate[expandedTemplate_List,
  k_Integer: 2, r_Integer: 1] :=
 FromDigits[#, k] & /@ Select[
   expandedTemplate,
   WellFormedRuleTableQ[#, k, r] &
  ]
```

```
RuleNumbersFromTemplate[
 ExpandTemplate[{0, 1 - x1, 0, 1, x2, 1, x1, 0}], 2, 1]
```

```
{92, 22, 30, 84}
```

With *Mathematica*'s built-in symbolic computation features, it is easy to create templates that represent a whole space. The space of elementary CAs would be represented by the following template.

```
BaseTemplate[k_Integer: 2, r_: 1] :=
 Symbol["x" <> ToString[#]] & /@ Range[k^(2 r+1), 1, -1]
```

```
BaseTemplate[2, 1]
```

```
{x8, x7, x6, x5, x4, x3, x2, x1}
```

In [4], the authors analytically found which transitions needed to be fixed, variable, or dependent on other transitions in a CA rule table, in order to have a chance to solve the parity problem perfectly. By fixing those transitions, they restrained the rule space of one-dimensional, binary, radius-2 CAs, composed of 4,294,967,296 rules, to only 16 candidates for perfect parity solvers. Although they used the de Bruijn graph as the primary structure to represent this rule space subset, it could have been easily represented with CA templates.

Empowered by *Mathematica*'s built-in equation-solving capabilities, algorithms can be developed that find the fixed, variable, and dependent state transitions on a rule table, thus leading to templates that are representatives of CAs that share the properties of number conservation and maximal internal symmetry; these are shown below.

◻ 4.1 Templates for Number-Conserving Rules

In [8], Boccara and Fukś established necessary and sufficient conditions that a CA rule table must meet in order to be conservative (which is another way to say number conserving). These conditions can be translated into an algorithm `BFConservationTemplate` that finds a set of equations that, when solved by *Mathematica*, yields the equivalent of a template that represents all conservative CAs of a determined space.

```
BFConservationTemplate[k_Integer: 2, r_: 1,
  intemplate_List: {}] := Module[
  {basetemplate, vars, relevantNeighbourhoods, equations},
  basetemplate = If[intemplate ≠ {}, intemplate,
    BaseTemplate[k, r]];
  vars = RuleTemplateVars[BaseTemplate[k, r]];
  relevantNeighbourhoods =
   Join[{Table[0, {2 r + 1}]},
    Cases[AllNeighbourhoods[k, r], {x_ /; x ≠ 0, ___}]];
  equations = (Equal @@ #) & /@
    ({RuleOutputFromNeighbourhood[#, basetemplate, k, r],

        First[#] +
         2 r
         ∑ (RuleOutputFromNeighbourhood[
         i=1
            Join[Table[0, {i}], Take[#, {2, 2 r - i + 2}]],
            basetemplate, k, r] -
           RuleOutputFromNeighbourhood[
            Join[Table[0, {i}], Take[#, {1, 2 r - i + 1}]],
            basetemplate, k, r])} & /@

                        relevantNeighbourhoods);
  First[basetemplate /. Quiet[Solve[equations, vars]]]
 ]
```

By running this function for the elementary space, the following template is obtained.

```
BFConservationTemplate[2, 1]

{1, 1 + x3 - x4, 1 - x3, 1 - x2 - x3, x4, x3, x2, 0}
```

When expanded, the latter yields the following representations.

```
ExpandTemplate[%]

{{{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}}}
```

However, it is clear that not all k-ary representations above are valid, since some of them rely on state values outside the range $\{0, k-1\}$, namely, the states 2 and -1. Hence, by discarding those three, we get the complete set of five number-conserving rules of the elementary space.

```
RuleNumbersFromTemplate[
 ExpandTemplate[BFConservationTemplate[2, 1]]]

{184, 204, 226, 170, 240}
```

It is important to notice that this kind of strategy can only be employed on properties that derive directly from the CA rule table.

□ 4.2 Templates for Rules with Maximal Internal Symmetry

As the internal symmetry of a CA is also a property that derives directly from its rule table, it is a valid candidate to be generalized into a template. By listing a CA rule table along with its respective transformations, it is possible to establish equality relations between them that, when solved by *Mathematica*, yield a template that represents all CAs that have the maximal possible value of internal symmetry, according to any subset of the three transformations.

By establishing that all of the results of the rule tables have to be the same in both the CA and its transformed counterpart, the following function `MaxSymmTemplate` achieves the goal of finding a template that represents all CAs of a given space that present the *maximum* value of internal symmetry, according to a list of transformations received as arguments.

```
MaxSymmTemplate[transformation_, k_Integer: 2, r_: 1,
   intemplate_List: {}] :=
 MaxSymmTemplate[{transformation}, k, r, intemplate]
```

```
MaxSymmTemplate[transformations_List, k_Integer: 2,
  r_: 1, intemplate_List: {}] := Module[
  {basetemplate, vars, ruleVariations, organisedmapping,
   eqrelations, equations, substitutions},
  basetemplate = If[intemplate ≠ {}, intemplate,
    BaseTemplate[k, r]];
  vars = RuleTemplateVars[basetemplate];
  ruleVariations =
   #[RuleTableFromkAry[basetemplate, k, r]] & /@
    Join[{# &}, transformations];
  organisedmapping =
   Reverse@
    SortBy[Union[#] & /@ GatherBy[Flatten[ruleVariations, 1],
       #[[1]] &], #[[1, 1]] &];
  eqrelations = (#[[2]] & /@ #) & /@ organisedmapping;
  equations = Apply[Equal, #] & /@ eqrelations;
  substitutions = Quiet[Solve[equations, vars]];
  Map[#[[2]] &, Flatten[Union[#], 1] & /@
    Flatten[organisedmapping /. substitutions, 1]]
 ]
```

In order to find a template that represents all elementary CAs with maximum symmetry according to the black-white transformation, it suffices to run `MaxSymmTemplate`, then expand the template to generate the rule numbers.

```
MaxSymmTemplate[BlackWhiteTransform, 2, 1]

{1 - x1, 1 - x2, 1 - x3, 1 - x4, x4, x3, x2, x1}

RuleNumbersFromTemplate[
 ExpandTemplate[{1 - x1, 1 - x2, 1 - x3, 1 - x4, x4, x3, x2, x1}]]

{232, 212, 204, 178, 170, 150,
 142, 113, 105, 85, 77, 51, 43, 23, 15, 240}
```

The verification of this result can be achieved by guaranteeing that all these rule numbers yield the same rule tables when transformed.

```
RuleTableFromRuleNumber[#] ==
   BlackWhiteTransform[RuleTableFromRuleNumber[#]] & /@
 {232, 212, 204, 178, 170, 150, 142, 113, 105, 85, 77,
  51, 43, 23, 15, 240}

{True, True, True, True, True, True, True, True,
 True, True, True, True, True, True, True, True}
```

We can analogously obtain a template representing all CAs with maximum symmetry according to all transformations, from which their expansions also lead to the corresponding rule numbers.

```
MaxSymmTemplate[{BlackWhiteTransform, LeftRightTransform,
  BlackWhiteTransform[LeftRightTransform[#]] &}, 2,
 1]

{1 - x1, 1 - x2, 1 - x3, x2, 1 - x2, x3, x2, x1}
```

```
RuleNumbersFromTemplate[
 ExpandTemplate[{1 - x1, 1 - x2, 1 - x3, x2, 1 - x2, x3, x2, x1}]]

{204, 178, 150, 105, 77, 51, 23, 232}
```

And again, their validity can be checked.

```
RuleTableFromRuleNumber[#] ==
    BlackWhiteTransform[RuleTableFromRuleNumber[#]] ==
    LeftRightTransform[RuleTableFromRuleNumber[#]] ==
    BlackWhiteTransform[
     LeftRightTransform[RuleTableFromRuleNumber[#]]] & /@
 {204, 178, 150, 105, 77, 51, 23, 232}

{True, True, True, True, True, True, True, True}
```

□ 4.3 Composition of Templates

Both the `BFConservationTemplate` and the `MaxSymmTemplate` functions can take another template as an optional argument, which is meant to be used as the starting point of the algorithms. This is the current way to compose the intersection of templates that share a common structure. For instance, in order to generate all the elementary conservative CAs with maximum internal symmetry values according to the black-white transformation, it becomes straightforward to use the template for number-conserving rules of the elementary space as the starting point of `MaxSymmTemplate`. This leads to a template that, once again, can be expanded so as to yield the target rule numbers.

```
MSBFTemplate = MaxSymmTemplate[BlackWhiteTransform,
  2, 1, BFConservationTemplate[2, 1]]

{1, 1 - x2, 1 - x3, 1 - x2 - x3, x2 + x3, x3, x2, 0}
```

```
RuleNumbersFromTemplate[ExpandTemplate[MSBFTemplate]]
```

```
{204, 170, 240}
```

Alternatively, the template with maximal internal symmetry could be used as the starting point of the `BFConservationTemplate` algorithm to obtain the same result.

```
BFMSTemplate = BFConservationTemplate[2, 1,
  MaxSymmTemplate[BlackWhiteTransform, 2, 1]]
```

```
{1, 1 - x2, 1 - x3, 1 - x2 - x3, x2 + x3, x3, x2, 0}
```

```
RuleNumbersFromTemplate[ExpandTemplate[BFMSTemplate]]
```

```
{204, 170, 240}
```

5. Concluding Remarks

The concept of CA templates was introduced, a rule table enhancement capable of representing a subset of a CA rule space, where the rules in the set can share a common property. Although the examples used for illustration only referred to one-dimensional, binary rules (the elementary space), the idea seems readily applicable to larger CAs with a larger number of states and more dimensions.

We have shown some of the operations applicable to CA templates, as well as some cases of use, in the form of *Mathematica* functions that yield templates representing subsets of the elementary space of CAs with properties related to number conservation and maximum internal symmetry. With respect to the latter, templates can be derived for any subset of the three symmetry-related transformations.

Templates for the rules in the same dynamical class in the elementary space have appeared previously in the CA literature, such as in [10]. But in these cases, the notion was not at all couched in the conceptual framework we have put forward, which allows templates to be effectively defined for rules having maximal internal symmetry value, let alone the possibility of representing further CA properties.

The properties used as examples here can be couched in terms of well-established relations among the state transitions of the CA, which are a necessary condition for a property to be addressed in the form of templates. As a counterpoint, the notion of reversibility of one-dimensional rules does not seem to be, at least in principle, amenable to template representation, since it is currently not known how to characterize reversibility in terms of the rule table of a CA.

It stands as future work to find new algorithms that would allow template representations of other properties, as well as the enhancement of the current algorithm related to internal symmetry templates, so as to extend the current constraint of only generating maximal internal symmetry toward also allowing the generation of templates with specific values of internal symmetry, not necessarily maximal.

Currently, because of computational demands, template expansion does not scale up well to very big templates; this should also be addressed in a follow-up. In particular, it might be worth defining operations of union and intersection of templates, which might be used to preprocess a template before the operation of template expansion.

■ Acknowledgments

Pedro de Oliveira thanks FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo). Maurício Verardo is thankful for a fellowship provided by CAPES, the Brazilian agency of its Ministry of Education.

■ References

[1] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media Inc., 2002.

[2] P. P. B. de Oliveira, "Conceptual Connections around Density Determination in Cellular Automata," *Cellular Automata and Discrete Complex Systems* (*Lecture Notes in Computer Science*), **8155**, 2013 pp. 1–14. doi:10.1007/978-3-642-40867-0_ 1.

[3] P. P. B. de Oliveira, "On Density Determination with Cellular Automata: Results, Constructions and Directions," *Journal of Cellular Automata*, forthcoming.

[4] H. Betel, P. P. B. de Oliveira, and P. Flocchini, "Solving the Parity Problem in One-Dimensional Cellular Automata," *Natural Computing*, **12**(3), 2013 pp. 323–337. doi:10.1007/s11047-013-9374-9.

[5] D. Wolz and P. P. B. de Oliveira, "Very Effective Evolutionary Techniques for Searching Cellular Automata Rule Spaces," *Journal of Cellular Automata*, **3**(4), 2008 pp. 289–312.

[6] H. Fukś, "A Class of Cellular Automata Equivalent to Deterministic Particle Systems," in *Hydrodynamic Limits and Related Topics*, (S. Feng, A. T. Lawniczak, and S. R. S. Varadhan, eds.), Providence, RI: American Mathematical Society, 2000 pp. 57–69.

[7] J. Kari and B. Le Gloannec, "Modified Traffic Cellular Automaton for the Density Classification Task," *Fundamenta Informaticae*, **116** (1–4), 2012 pp. 141–156. doi:10.3233/FI-2012-675.

[8] N. Boccara and H. Fukś, "Number-Conserving Cellular Automaton Rules," *Fundamenta Informaticae*, **52**(1–3), 2002 pp. 1–13.

[9] A. Schranko and P. P. B. de Oliveira, "Towards the Definition of Conservation Degree for One-Dimensional Cellular Automata Rules," *Journal of Cellular Automata*, **5**(4–5), 2010 pp. 383-401.

[10] W. Li and N. Packard, "The Structure of the Elementary Cellular Automata Rule Space," *Complex Systems*, **4**(3), 1990 pp. 281–297. www.complex-systems.com/pdf/04-3-3.pdf.

P. P. B. de Oliveira and M. Verardo, "Representing Families of Cellular Automata Rules," *The Mathematica Journal*, 2014. dx.doi.org/doi:10.3888/tmj.16-8.

About the Authors

Pedro de Oliveira has been a faculty member since 2001 of the School of Computing and Informatics and of the Postgraduate Program in Electrical Engineering at Mackenzie Presbyterian University, São Paulo, Brazil. His research interests are cellular automata, evolutionary computation, and cellular multi-agent systems. Pedro is an alumnus of the 2003 NKS Summer School.

Maurício Verardo is a post-graduate student in Electrical Engineering at Mackenzie Presbyterian University, working with cellular automata ever since his undergraduate senior project for his computer science degree from Mackenzie. Maurício is an alumnus of the 2011 NKS Summer School.

Pedro P. B. de Oliveira
Faculdade de Computação e Informática & Pós-Graduação em Engenharia Elétrica
Universidade Presbiteriana Mackenzie
Rua da Consolação, 930
São Paulo, 01302-907 - Brazil
pedrob@mackenzie.br

Maurício Verardo
Pós-Graduação em Engenharia Elétrica
Universidade Presbiteriana Mackenzie
Rua da Consolação, 930
São Paulo, 01302-907 - Brazil
mauricio.verardo@gmail.com