Merging Cellular Automata Rules to Optimise a Solution to the Modulo-n Problem

Claudio M. Martins, Pedro De Oliveira

Merging Cellular Automata Rules to Optimise a Solution to the Modulo-*n* Problem

Claudio L.M. Martins[2] and Pedro P.B. de Oliveira[1,2]

Universidade Presbiteriana Mackenzie
Faculdade de Computação e Informática[1] &
Pós-Graduação em Engenharia Elétrica e Computação[2]
Rua da Consolação 896, Consolação
01302-907 São Paulo, SP - Brazil
claudio.luis.martins@terra.com.br
pedrob@mackenzie.br

Abstract. Understanding how the composition of cellular automata rules can perform predefined computations can contribute to the general notion of emerging computing by means of locally processing components. In this context, a solution has been recently proposed to the Modulo-*n* Problem, which is the determination of whether the number of 1-bits in a binary string is perfectly divisible by the positive integer *n*. Here, we show how to optimise that solution in terms of a reduction of the number of rules required, by means of a *merging* operation involving of the rules´ active state transitions. The potential for a more general usage of the merging operation is also addressed.

Keywords: Cellular automata, emergent computation, rule composition, Modulo-*n* problem, MOD*n* problem, merging, active state transitions, parity problem.

1 Introduction: The Modulo-*n* Problem Solution to be Optimised

Cellular automata (CAs) are discrete dynamical systems with a grid-like regular lattice of identical finite automata cells, each cell having an identical pattern of connections to its neighbours. The next state of each cell is given by the transition rule of the automaton, according to the current cell state and those of its neighbouring cells. CAs may perform arbitrary computations, even out of the action of simple local rules [6].

One of these computations consists in solving the parity problem, herein denoted the MOD2 problem, which consists of determining the parity of the number of 1s in a binary string: even parity, when the number of 1s modulo-2 is 0, or odd parity, when the number of 1s modulo-2 is 1.

In its formulation for cellular automata, this computational problem is considered solved when any odd-sized (N) binary string initialising a cyclic lattice, is converted, after some time steps, into 0^N or 1^N, if the initial amount of 1-bits is even, or odd, respectively. The MOD2 problem is ill-defined for even-sized lattices because the initial configuration 1^N would have an even number of 1s, which would be a contradiction because 1^N should be the final configuration for lattices with odd number of

1s. Although it has been proved in [5] that a one-dimensional rule with radius at least 4 is required to solve MOD2, [2] and [4] showed how to solve the problem with a composition of only two elementary rules. This makes it evident the power of rule compositions.

As a general case, we refer to the MOD*n* problem, which consists of determining whether the number of 1-bits in a cyclic binary string is multiple of *n*, with the constraint that it is always ill-defined for lattice sizes multiple of *n*.

In [1], we described a generalised solution to the MOD*n* problem, based upon the application of a set of one-dimensional CA rules, in a pre-determined order, which amounts to composing the individual rules employed. This solution is only constrained in that the lattice size *N* cannot be a multiple of *n* nor a multiple of any factor of *n*. Such a general solution (S_n) for the MOD*n* problem, for any binary string σ with size *N*, is given below, where we name rules R_n^0 and R_n^1 as the *Replacement* rules, and G_1^0, G_1^1, G_2^0, G_2^1 and so on, as the *Grouping* rules, whose meanings are defined in the next section.

$$S_n = E_{254}^{\left\lfloor \frac{N}{2} \right\rfloor} \left(G_{n-2}^{1\left\lfloor \frac{N}{2} \right\rfloor} \ldots G_1^{1\left\lfloor \frac{N}{2} \right\rfloor} R_n^{1\left\lfloor \frac{N}{n} \right\rfloor} G_{n-2}^{0\left\lfloor \frac{N}{2} \right\rfloor} \ldots G_1^{0\left\lfloor \frac{N}{2} \right\rfloor} R_n^{0\left\lfloor \frac{N}{n} \right\rfloor} \right)^{\left\lceil \frac{N}{n} \right\rceil} \sigma. \qquad (1)$$

The solution means that starting with the size-*N* initial configuration σ, the following sequence of rule applications should be performed:

1. Apply rule R_n^0 for $\lfloor \frac{N}{n} \rfloor$ time steps, followed by rules G_1^0, G_2^0 and so on, up to G_{n-2}^0, for $\lfloor \frac{N}{2} \rfloor$ time steps each.
2. Apply rule R_n^1 for $\lfloor \frac{N}{n} \rfloor$ time steps, followed by rules G_1^1, G_2^1 and so on, up to G_{n-2}^1, for $\lfloor \frac{N}{2} \rfloor$ time steps each.
3. Repeat the two previous procedures $\lceil \frac{N}{n} \rceil$ times.
4. Finalise the process, by applying elementary CA rule 254 for $\lfloor \frac{N}{2} \rfloor$ time steps.

Since the sequence of rules superscripted with 0 operate on the 0-bits and the ones superscripted with 1 operate on the 1-bits, the stages 1 and 2 above may be inverted, with the same global outcome.

In this paper, we propose a simplification of the solution above, by performing a *merging* procedure of the rules´ *active* state transitions, that is, those that replace the state of the centre cell in the neighbourhood.

In the next section, we present the *Replacement* rules, the *Grouping* rules and the result of their composition. In Section 3, we discuss the active state transitions of the rules present in S_n, as well as how they can be used to simplify their representations. The merging of these rules is then discussed in Section 4, as well as how the active transitions should be modified so as to render viable mergings. We conclude in Section 5 with various remarks, in particular addressing some conditions we must respect in the choice of rules to be merged and the active transitions to be modified.

2 Replacement and Grouping Rules

Two key roles are required for rules to solve the MOD*n* problem: to transform certain blocks of states of a configuration, and to group together specific kinds of blocks; these roles are achieved by the *Replacement* rules (*R*) and the *Grouping* rules (*G*), respectively. Elementary rule 254 just finalises the problem, by preserving the configuration 0^N, and transforming all the others to 1^N.

Replacement rules R_n^0 can replace *n* end 0s, of a sequence of *n* or more consecutives 0s, with *n* 1s, while, analogously, *Replacement* rules R_n^1 can replace *n* end 1s, of a sequence of *n* or more consecutives 1s, with *n* 0s. Both are, therefore, MOD*n*-conserving rules. Considering the *n* end bits, 0s or 1s, that need to be replaced of a sequence of *n* or more consecutive identical bits, the following cases are possible: *n* bits from the left, *n*-1 from the left and 1 from the right, *n*-2 from the left and 2 from the right, …, 2 from the left and *n*-2 from the right, 1 from the left and *n*-1 from the right, and *n* bits from the right.

Knowing that a one-dimensional CA rule that changes *n* end bits from one side of the string must have at least radius *n*, and that it suffices radius *n*-1 for a rule that is to change *n*-1 bits from one side and 1 from the other side, or *n*-2 bits from one side and 2 from the other side, and so on, we can consider only the smallest possible radius of these rules.

So, there are *n*-1 rules that replace *n* end 0s, of a sequence of *n* or more 0s, with *n* 1s, namely, rule $R_{n-1,1}^0$ (*n*-1 bits from the left and 1 from the right), that transforms the strings $10^{n-1}0^x01$ into $11^{n-1}0^x11$, rule $R_{n-2,2}^0$ (*n*-2 bits from the left and 2 from the right), that transforms the strings $10^{n-2}0^x001$ into $11^{n-2}0^x111$, and so on, up to rule $R_{1,n-1}^0$ (1 bit from the left and *n*-1 from the right), that transforms the strings $100^x0^{n-1}1$ into $110^x1^{n-1}1$; in all cases, for any integer $x \geq 0$.

By applying *Replacement* rules for $\lfloor \frac{N}{n} \rfloor$ iterations, no sequence with *n* or more consecutive identical bits is left in the lattice, except if its configuration is 0^N or 1^N.

In order to eliminate simultaneous occurrence of different blocks of the same bit left by the *Replacement* rules, the smaller blocks will be grouped into larger ones, moving themselves through the lattice, according to the *Grouping* rules.

Grouping rules are those that can shift to the left or to the right, strings of *m* identical bits, in order to group them with larger strings of the same bit.

We refer to a *Grouping* rule that can shift *m* 0s as G_m^0; but since this movement may be possible to the left or to the right, we denote it $\overleftarrow{G}_m^0$ when the movement is to the left, and $\vec{G}_m^0$ when the movement is to the right. Analogously, in order to shift and group *m* 1s, rules $\overleftarrow{G}_m^1$ and $\vec{G}_m^1$ are employed, or just G_m^1, indistinctly.

Rules that can only move an isolated bit, 0 or 1 (G_1^0 or G_1^1, respectively), should have, at least, radius 2. Rules that can only move an isolated pair of bits should have at least radius 3, and so on. These rules are also MOD*n*-conserving rules.

For either 0 or 1, *n*-2 *Grouping* rules will be used, because, after the application of the *Replacement* rules, no strings of consecutive identical bits larger than *n*-1 will remain in the lattice. So, we just have to move strings smaller than *n*-1 consecutive identical bits to group them into the larger blocks of the same bit.

In order to solve the MOD2 problem, since $n = 2$, no *Grouping* rules are necessary, as demonstrated in [4].

By composing only *Replacement* and *Grouping* rules, with no application of the elementary rule 254, any initial configuration is transformed into another that belongs to a reduced group of final configurations. We disregard differences due to rotational symmetry, which means that final configurations as 1100000000, 0110000000, ..., 0000000110 are considered the same as 0000000011.

Simplifying the possible relationships between initial and final configurations before applying rule 254, we have the following, where $|\sigma|_1$ stands for the number of 1s in string σ:

If MOD$n(N)$ = MOD$n(|\sigma|_1)$ and both $\neq 0$ (ill-defined problem):

— MOD$n(|\sigma|_1) \neq 0$: 1^N (meaning that, when MODn $(|\sigma|_1)=1$, convergence is to 1^N)

If MOD$n(N) \neq$ MOD$n(|\sigma|_1)$:

— MOD$n(|\sigma|_1) = 0$: 0^N

— MOD$n(|\sigma|_1) \neq 0$: $0^{N\text{-MOD}n(|\sigma|1)}1^{\text{MOD}n(|\sigma|1)}$ or some *necklaces*

We have already disregarded the lattices with size N multiple of n (ill-defined problem), where MOD$n(N) = 0$.

The predominance of 0s instead of 1s is because, at the end, we use the *Replacement* rules that operate on the 1s, transforming them into 0s.

Necklaces are configurations of the form $(0^A1^B)^C$, for integers A, B and C, where $A < n$ and $B < n$, or $A < n$ and $B > n$, but B is not multiple of n, or $B < n$ and $A > n$ but A is not multiple of n. For necklace configurations, the *Grouping* rules just cause shifts on the lattice, with no further effect; also, the *Replacement* rules cause no effect when $A < n$ and $B < n$, or may lead to periodic regimes only when $A > n$ or $B > n$, by continuously transforming $(0^A1^B)^C$ into $(0^{A'}1^{B'})^C$, back and forth.

If N is a prime number, no *necklace* will remain in the lattice.

Therefore, for any initial configurations where MOD$n(|\sigma|_1) = 0$, the problem is already solved, as defined. However, for all the other configurations where MOD$n(|\sigma|_1) \neq 0$ should be converted into 1^N, elementary rule 254 can be used, without affecting the configuration 0^N.

All the details, lemmas and their proofs, and further explanations regarding this section can be found in [1].

3 Active State Transitions and the Simplified CA Representations

A solution for the MOD3 problem was reported in [3] and further optimised and generalised in [1]. In order to solve the MOD3 problem we need radius 2 rules as *Replacement* rules and *Grouping* rules. A radius 2 rule has 32 state transitions that can be active or not: by active transition, we mean those that change the state value of the centre cell of the neighbourhood.

$R^0_{2,1}$ **Rule 4.059.296.252**

Transition	Neighbourhood					Output bit / Note	
31	1	1	1	1	1	1	
30	1	1	1	1	0	1	
29	1	1	1	0	1	1	
28	1	1	1	0	0	1	
27	1	1	0	1	1	0	
26	1	1	0	1	0	0	
25	1	1	0	0	1	0	
24	1	1	0	0	0	1	1st 0 from the left
23	1	0	1	1	1	1	
22	1	0	1	1	0	1	
21	1	0	1	0	1	1	
20	1	0	1	0	0	1	
19	1	0	0	1	1	0	
18	1	0	0	1	0	0	
17	1	0	0	0	1	1	2nd 0 from the left
16	1	0	0	0	0	1	2nd 0 from the left
15	0	1	1	1	1	1	
14	0	1	1	1	0	1	
13	0	1	1	0	1	1	
12	0	1	1	0	0	1	
11	0	1	0	1	1	0	
10	0	1	0	1	0	0	
9	0	1	0	0	1	0	
8	0	1	0	0	0	1	1st 0 from the left
7	0	0	1	1	1	1	
6	0	0	1	1	0	1	
5	0	0	1	0	1	1	
4	0	0	1	0	0	1	
3	0	0	0	1	1	1	1st 0 from the right
2	0	0	0	1	0	1	1st 0 from the right
1	0	0	0	0	1	0	
0	0	0	0	0	0	0	

Active Transition	Neighbourhood					Output bit / Note	
24	1	1	0	0	0	1	1st 0 from the left
17	1	0	0	0	1	1	2nd 0 from the left
16	1	0	0	0	0	1	2nd 0 from the left
8	0	1	0	0	0	1	1st 0 from the left
3	0	0	0	1	1	1	1st 0 from the right
2	0	0	0	1	0	1	1st 0 from the right

Active Transitions	Neighbourhood					Output bit / Note	
24 & 8	*	1	0	0	0	1	1st 0 from the left
17 & 16	1	0	0	0	*	1	2nd 0 from the left
3 & 2	0	0	0	1	*	1	1st 0 from the right

$\overleftarrow{G}^0_1$ **Rule 3.704.675.536**

Transition	Neighbourhood					Output bit / Note	
31	1	1	1	1	1	1	
30	1	1	1	1	0	1	
29	1	1	1	0	1	0	1 on the left
28	1	1	1	0	0	1	
27	1	1	0	1	1	1	Isolated 0
26	1	1	0	1	0	1	Isolated 0
25	1	1	0	0	1	0	
24	1	1	0	0	0	0	
23	1	0	1	1	1	1	
22	1	0	1	1	0	1	
21	1	0	1	0	1	0	1 on the left
20	1	0	1	0	0	1	
19	1	0	0	1	1	0	
18	1	0	0	1	0	0	
17	1	0	0	0	1	0	
16	1	0	0	0	0	0	
15	0	1	1	1	1	1	
14	0	1	1	1	0	1	
13	0	1	1	0	1	0	1 on the left
12	0	1	1	0	0	1	
11	0	1	0	1	1	1	Isolated 0
10	0	1	0	1	0	1	Isolated 0
9	0	1	0	0	1	0	
8	0	1	0	0	0	0	
7	0	0	1	1	1	1	
6	0	0	1	1	0	1	
5	0	0	1	0	1	0	1 on the left
4	0	0	1	0	0	1	
3	0	0	0	1	1	0	
2	0	0	0	1	0	0	
1	0	0	0	0	1	0	
0	0	0	0	0	0	0	

Active Transition	Neighbourhood					Output bit / Note	
29	1	1	1	0	1	0	1 on the left
27	1	1	0	1	1	1	Isolated 0
26	1	1	0	1	0	1	Isolated 0
21	1	0	1	0	1	0	1 on the left
13	0	1	1	0	1	0	1 on the left
11	0	1	0	1	1	1	Isolated 0
10	0	1	0	1	0	1	Isolated 0
5	0	0	1	0	1	0	1 on the left

Active Transitions	Neighbourhood					Output bit / Note	
27, 26, 11 & 10	*	1	0	1	*	1	Isolated 0
29, 21, 13 & 5	*	*	1	0	1	0	1 on the left

Figure 1: State transitions of rules $R^0_{2,1}$ and $\overleftarrow{G}^0_1$, with their simplified representations through their active transitions.

CA rules can be represented just through their active transitions, and grouped whenever possible, i.e., placed together at the same row of the state transition table, using symbol * to stand for either possibilities, 0 or 1 (see Figure 1).

This figure shows all the state transitions of the rules $R^0_{2,1}$ and $\overleftarrow{G}^0_1$, separating and placing together only the active transitions as shown further down in the figure. The *Replacement* rule $R^0_{2,1}$ has only 6 active transitions, highlighted out of the 32 possibilities, and can be represented by the three rows and the end of the figure. The *Grouping* rule $\overleftarrow{G}^0_1$ has 8 active transitions, and can be represented just by two rows. The written notes indicate which bit is been replaced by the *Replacement* rule or changed to make the shift by the *Grouping* rule.

The minimum required radius for a rule to perform as expected depends on the number of cells (excluding those with the * character), to the right or to the left, of the centre cell of the neighbourhood of all grouped active transitions. The highest value is the required radius. At the end of Figure 1 we can see that the minimum radius required to both rules is 2.

4 Merging *Replacement* and *Grouping* Rules

4.1 The Merging Operation

The *merging* process should, in fact, be generally regarded as a two-stage process, consisting of *joining* together the active transitions of the rules involved, with subsequent *editing* of some of them, if required. Details are given throughout this section.

The *Replacement* and *Grouping* rules to be joined in just one *Merged* rule do not operate on the same strings simultaneously, because these strings have different sizes for any value of *n*, i.e., while *Replacement* rules replace *n* bits (from strings with *n* or more consecutive identical bits), *Grouping* rules move *m* bits (from strings with just an isolated string of *m* bits), and *m* is always equal or smaller than *n*-2.

Therefore, all effects of the *Merged* rule – such as the possibility of partition reduction, the formation of strings with only 0s or only 1s, or the formation of some necklaces – are the same when applying the separated rules. One exception is the formation of *partial necklaces* (described in Section 4.2), that occurs because of the impossibility to join some isolated strings to their larger blocks (created by the *Grouping* rules), due to changes on the lattice, through the simultaneous action of the *Replacement* and *Grouping* rules.

As a consequence, the same lemmas and proofs described in [1] should be considered, but the formation of *partial necklaces* must now be added to the possibilities of final configuration after applying the *Merged* rule. In the next subsections we address some specific cases, such as the merging of the rules that solve the problem MOD3 and the problem MOD4, so as to convey the underlying issues of the process more clearly.

4.2 The MOD3 Case

As is the case here, we may consider that a task of a CA rule can be performed by one or more *active state transitions*. Accordingly, the task performed by rule $R^0_{2,1}$, for example, is to eliminate any string with three or more consecutive 0s in the lattice, conserving the Modulo-3 property, while the task performed by the rule $\overleftarrow{G}^0_1$ is to shift isolated 0s to the left in order to group them with larger strings of the same bit, also conserving the Modulo-3 property. In what follows, we go about joining these functions into a single rule.

Figure 1 shows that *Replacement* rule $R^0_{2,1}$ and *Grouping* rule $\overleftarrow{G}^0_1$ have different active state transitions. The resulting rule of the merging of a *Replacement* rule and one or more *Grouping* rules is generically termed in this work as M^0_n or M^1_n, according to the specific bit it manipulates. In the case of the MOD3 problem, for instance, merging rules $R^0_{2,1}$ and $\overleftarrow{G}^0_1$ results the rule $\overleftarrow{M}^0_{2,1}$.

Figure 2 shows the simplified representation of rule $\overleftarrow{M}^0_{2,1}$ from the perspective of its active transitions.

$\overleftarrow{M}^0_{2,1}$ **Rule 3.721.649.628**

Active Transitions	Neighbourhood					Output bit	Note
27, 26, 11 & 10	*	1	0	1	*	1	Isolated 0
29, 21, 13 & 5	*	*	1	0	1	0	1 on the left
24 & 8	*	1	0	0	0	1	1st 0 from the left
17 & 16	1	0	0	0	*	1	2nd 0 from the left
3 & 2	0	0	0	1	*	1	1st 0 from the right

Figure 2: Simplified representation of the rule $\overleftarrow{M}^0_{2,1}$.

Our goal is to ensure that these 3 alternatives can perform the same tasks: $\left(\overleftarrow{G}^{0\,N}_1 \; R^{0\,N}_{2,1}\right)^N \sigma$, $\left(R^{0\,N}_{2,1} \; \overleftarrow{G}^{0\,N}_1\right)^N \sigma$, or $\left(\overleftarrow{M}^{0\,N}_{2,1}\right)^N \sigma$.

The application to all possible initial configurations of a given size of the possible rule sequences – that is, the *Replacement* rule followed by the *Grouping* rule, or vice versa, or yet the *Merged* rule only – allows us to compare all resulting final configurations; this is what is summarised in Figure 3, for $N = 13$ (therefore, with respect to all 2^{13} different initial configurations).

Analysis of the data in Figure 3 indicates that the Modulo-3 property is preserved for all initial configurations, thus increasing their number of 1s, up to 11, 12 or 13 occurrences, correspondingly to the initial values of Modulo-3 equal to 2, 0 or 1, respectively. This occurs because the *Replacement* rule can transform three 0s into three 1s. The alternative that begins with the *Grouping* rule has an advantage in this

replacement task, in that it shifts isolated 0s and groups them into larger chains before replacing the 0s. This increases the amount of strings with three or more consecutive 0s, therefore improving the effectiveness of the *Replacement* rule.

Hence, the final configurations of the different alternatives are not necessarily the same, even disregarding differences due to rotational symmetries, because the number of 1s may be different. Even with the same number of 1s, the spaces between isolated 0s may be also different.

By a superficial analysis, we would say that there is no equivalence among the outcomes of the three possible processing alternatives; but, if we go back to our initial goals, it is possible to observe that, for all possible initial conditions, the application of any of the three alternatives transforms the lattice into a final configuration as desired. In other words, all final configurations have at most two consecutive 0s (because of the *Replacement* rule) and there are no isolated 0s and isolated pairs of 0s occurring simultaneously (because of the *Grouping* rule).

$\left(\overleftarrow{G}_1^{0\,N} R_{2,1}^{0\,N} \right)^N$

Amount of 1s		Amount of
Initial	Final	Configurations
0	0	1
1	13	13
2	11	78
3	9	78
	12	208
4	7	52
	10	130
	13	533
5	5	13
	8	13
	11	1261
6	9	585
	12	1131
7	7	78
	10	455
	13	1183
8	8	91
	11	1196
9	9	234
	12	481
10	10	156
	13	130
11	11	78
12	12	13
13	13	1
Total		8192

$\left(R_{2,1}^{0\,N} \overleftarrow{G}_1^{0\,N} \right)^N$

Amount of 1s		Amount of
Initial	Final	Configurations
0	0	1
1	13	13
2	11	78
3	9	104
	12	182
4	7	65
	13	650
5	5	13
	11	1274
6	9	650
	12	1066
7	7	78
	13	1638
8	8	91
	11	1196
9	9	234
	12	481
10	10	156
	13	130
11	11	78
12	12	13
13	13	1
Total		8192

$\left(\overleftarrow{M}_{2,1}^{0\,N} \right)^N$

Amount of 1s		Amount of
Initial	Final	Configurations
0	0	1
1	13	13
2	11	78
3	9	78
	12	208
4	7	52
	10	130
	13	533
5	5	13
	8	13
	11	1261
6	9	533
	12	1183
7	7	78
	10	650
	13	988
8	8	91
	11	1196
9	9	234
	12	481
10	10	156
	13	130
11	11	78
12	12	13
13	13	1
Total		8192

Figure 3: Summary after the application to all possible initial configurations of a given size (N=13) of the different rule sequences.

So, if the desired task is exactly the latter (lattice with at most two consecutive 0s, and without isolated 0s and pairs of 0s simultaneously), the goal has been achieved, demonstrating the equivalence of these alternatives.

We should remember that the solution to the MOD3 alternates the sequence of rules that operate on the 0s with the sequence of rules that operate on the 1s, alternating the size of the bit strings with only 0s and 1s, until achieving the required condition for elementary rule 254 to finalise the solution.

Therefore, applying rule M_3^0 instead of R_3^0 and G_1^0, and rule M_3^1 instead of R_3^1 and G_1^1, the solution S_3 is simplified to Ss_3, using only three rules, instead of five, as originally (according to [1]):

$$S_3 = E_{254}^{\lfloor \frac{N}{2} \rfloor}\left(G_1^{1\lfloor \frac{N}{2} \rfloor} R_3^{1\lfloor \frac{N}{3} \rfloor} G_1^{0\lfloor \frac{N}{2} \rfloor} R_3^{0\lfloor \frac{N}{3} \rfloor} \right)^{\lceil \frac{N}{3} \rceil} \sigma \quad (2)$$

$$Ss_3 = E_{254}^{\lfloor \frac{N}{2} \rfloor}\left(M_3^{1N} M_3^{0N} \right)^{\lfloor \frac{N}{3} \rfloor} \sigma. \quad (3)$$

Disregarding the action of elementary rule 254 in the previous expressions, and letting S_3' and Ss_3' denote the remaining (partial) and simplified (partial) solutions, respectively, Figure 4 compares them, displaying a summary of all final configurations left after applying these two solutions to all possible initial configurations with N=16. The final configuration 1100000000000000 and all its equivalent configurations due to rotational symmetry are considered the same.

Original Sequence S_3'

Amount of 1s	Final Configuration with some Necklaces	No. Of Configs.
0	0000000000000000	21845
2	0000000000000011	19232
	0000000100000001	2608
8	0011001100110011	4
	0101010101010101	2
16	1111111111111111	21845
TOTAL		65536

Simplified Sequence Ss_3'

Amount of 1s	Final Configuration with some Necklaces	No. Of Configs.
0	0000000000000000	21845
2	0000000000000011	14576
	0000000100000001	656
8	0011001100110011	4
	0101010101010101	2
16	1111111111111111	21845
TOTAL		65536

Amount of 1s	Final Configuration with some Partial Necklaces	No. Of Configs.
2	0000000000000101	4768
	0000000000100001	1568
5	0000000101010101	144
	0000100001010101	64
	0000101000010101	64

Figure 4: Summary after applying both partial solutions (original and simplified) to the MOD3 problem on all possible initial configurations of size 16.

The simplified partial solution transforms some initial configurations into *partial necklaces* (i.e, configurations where only a part of it is a *necklace*), further to the *necklaces* that already had been transformed by the original partial solution.

Necklaces and *partial necklaces* appear only when MODn(N) ≠ MODn($|\sigma|_1$); otherwise, the lattice converges to 0^N or 1^N.

For the MOD3 problem, *necklaces* have the forms $(01)^8$ or $(0^21^2)^4$, and *partial necklaces* belong to $((0^3)^+(01)^+)^+$; they are trapped in these forms because after M_3^0 has transformed the three 0s into three 1s, only isolated 0s remain, that move synchronously, keeping the form $((1^3)^+(10)^+)^+$, until M_3^1 is applied, thus reversing the configuration to $((0^3)^+(01)^+)^+$.

For both alternatives, original and simplified partial solutions, any initial configurations where MODn($|\sigma|_1$) = 0 converges to 0^N and the problem is already solved, as

defined. However, in order for all the other configurations with MOD$n(|\sigma|_1) \neq 0$ to end up in 1^N, elementary rule 254 is used, with no effect on the configuration 0^N.

4.3 The MOD4 Case and the Problem of Synchronised Displacements

Following the same rationale for merging rules in the optimised MOD3 solution (i.e., merging *Replacement* and *Grouping* rules that work on the same bit), in order to optimise the solution to MOD4 we have to merge one *Replacement* rule with two *Grouping* rules. For the 0-bit, M_4^0 results from merging rules R_4^0, G_1^0 and G_2^0, while for the 1-bit, M_4^1 results from merging rules R_4^1, G_1^1 and G_2^1. Rules R_4 and G_2 should have radius 3, at least. Therefore, the resulting M_4 rules should also have radius 3. We employ the '+' symbol for the joining of active transitions of the rules.

The problem in this merging is related to the *Grouping* rules. For instance, applying rules G_1^0 and G_2^0, separately or joined, without editing some active transitions, does not lead to the same outcome because of possible synchronised movements of isolated 0s and isolated pairs of 0s.

Figure 5 shows the joining of rules $R_{2,2}^0$, $\overleftarrow{G}_1^0$ and $\overleftarrow{G}_2^0$, and also shows the temporal evolution of an initial configuration where the problem occurs.

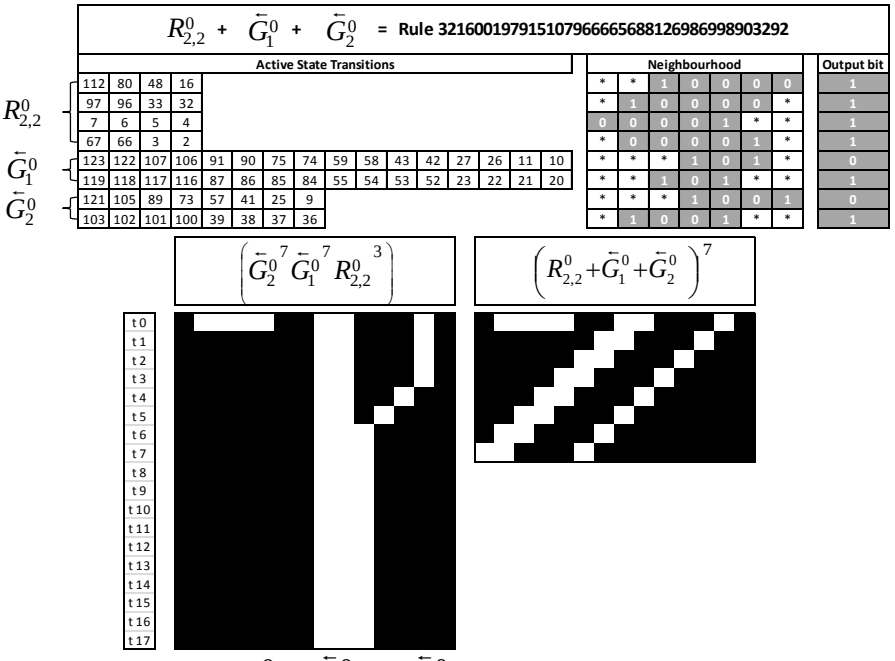

Figure 5: Joining of rules $R_{2,2}^0$, $\overleftarrow{G}_1^0$ and $\overleftarrow{G}_2^0$, and the temporal evolution of the configuration 10000110011101 comparing two processing alternatives (joined or separate rules).

4.4 Removing and Inserting Active Transitions to Change the Displacement Step of the Rule

Hence, for the efficient merging of these three rules, we have to fix these synchronised movements caused by the *Grouping* rules. A first idea would be to exchange one of them by a shift to the right, compensating for the current shift to the left. However, looking ahead to the context of the MOD5 problem, in order to optimise its solution with 3 *Grouping* rules, this method would no longer be sufficient.

Since we are handling larger radius than in the MOD3 problem, a more robust solution is to change the displacement step of rule $\overleftarrow{G}_1^0$: instead of moving the isolated 0 just one position to the left, we had better move it two positions, whenever possible (which is not always the case). The new rule with this feature is denoted $\overleftarrow{G}_1^{0,2}$. Figure 6 shows its 8 active transitions (125, 109, 93, 77, 61, 45, 29 and 13), but no longer the 8 others (123, 122, 91, 90, 59, 58, 27 and 26) that the previous rule had.

$\overleftarrow{G}_1^{0,2}$ = Rule 297668273963817264613187722719825810176

Active State Transitions																Neighbourhood							Output bit
119	118	117	116	87	86	85	84	55	54	53	52	23	22	21	20	*	*	1	0	1	*	*	1
		107	106			75	74			43	42			11	10	*	*	0	1	0	1	*	0
125	109			93	77			61	45			29	13			*	*	*	1	1	0	1	0

Figure 6: Rule $\overleftarrow{G}_1^{0,2}$ represented by its active transitions.

4.5 Removing Active State Transitions to Eliminate Remaining Synchronised Displacements

Finally, there is a further problem yet to be solved: when the isolated 0 cannot be moved 2 positions because of an isolated pair of 0s close to its left (as happens in the string *100101*), a synchronised displacement will continue to occur. Therefore, instead of transforming the string 100101 into 001011 (as would happen due to the modification just proposed), we had rather transform it into 000111.

To accomplish this some active state transitions should be turned off, as shown in Figure 7, indicated by two arrows. For clarity purposes of this process, notice that the string **0101* in Figure 6 has been instantiated into strings 000101*, 010101*, 100101* and 110101*, and that the string *1001** in Figure 5 gave rise to *100100, *100101, *100110 and *100111.

The active transitions that should be deactivated are 75 and 74 from rule $\overleftarrow{G}_1^{0,2}$, and 101 and 37 from rule $\overleftarrow{G}_2^0$. Rules $\overleftarrow{G}_1^{0,2}$ and $\overleftarrow{G}_2^0$ become $\overleftarrow{G}_{1*}^{0,2}$ and $\overleftarrow{G}_{2*}^0$, respectively, after some of their original active transitions have been deactivated (see Figure 7). Rule $\overleftarrow{M}_{2,2}^0$ is derived from joining $R_{2,2}^0$, $\overleftarrow{G}_{1*}^{0,2}$ and $\overleftarrow{G}_{2*}^0$.

The ideal merging of active state transitions to solve these conflicts is presented in Figure 8.

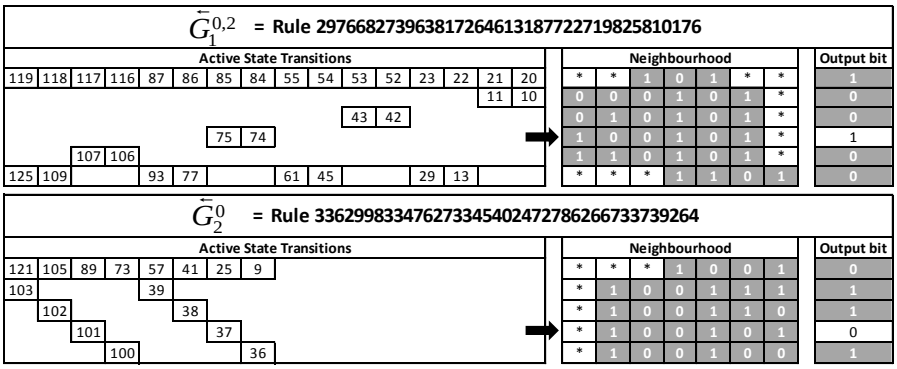

Figure 7: Rules $\overleftarrow{G}_1^{0,2}$ and $\overleftarrow{G}_2^{0}$ represented by their active state transitions and pointing out those which have to be disabled (not highlighted output bit).

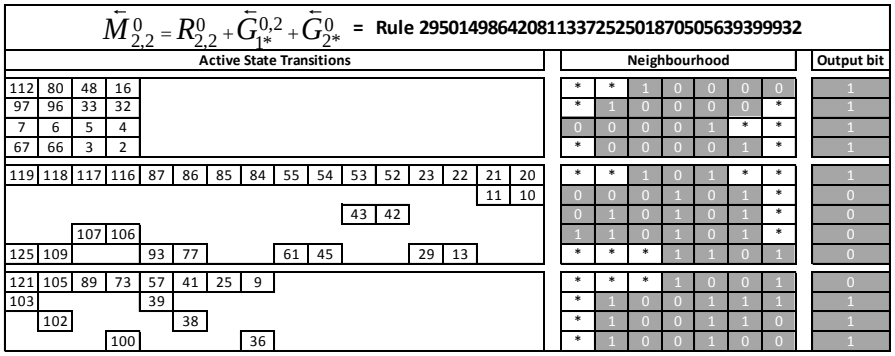

Figure 8: Rule $\overleftarrow{M}_{2,2}^{0}$ represented by its active state transitions.

The same rationale lead us to rule $\overleftarrow{M}_{2,2}^{1}$, composed by $R_{2,2}^{1}$, $\overleftarrow{G}_{1*}^{1,2}$ and $\overleftarrow{G}_{2*}^{1}$, whose Wolfram number is 255816358659918533639648036274123862084.

So, the solution S_4 is simplified to Ss_4, using only 3 rules instead of the 7 original rules:

$$S_4 = E_{254}^{\left\lfloor \frac{N}{2} \right\rfloor} \left(G_2^{1\left\lfloor \frac{N}{2} \right\rfloor} G_1^{1\left\lfloor \frac{N}{2} \right\rfloor} R_4^{1\left\lfloor \frac{N}{4} \right\rfloor} G_2^{0\left\lfloor \frac{N}{2} \right\rfloor} G_1^{0\left\lfloor \frac{N}{2} \right\rfloor} R_4^{0\left\lfloor \frac{N}{4} \right\rfloor} \right)^{\left\lceil \frac{N}{4} \right\rceil} \sigma \quad (4)$$

$$Ss_4 = E_{254}^{\left\lfloor \frac{N}{2} \right\rfloor} \left(M_4^{1N} M_4^{0N} \right)^{\left\lfloor \frac{N}{4} \right\rfloor} \sigma. \quad (5)$$

Testing the same procedure for MOD5 and obtaining similar results, we conclude that the simplified solution that can solve the MOD*n* problem can have only 3 rules, one elementary (radius 1), and two others with radius *n*-1, as long as the size *N* of the binary string σ is not multiple of *n* neither multiple of any factor of *n*. This simplified solution then becomes:

$$Ss_n = E_{254}^{\left\lfloor \frac{N}{2} \right\rfloor} \left(M_n^{1N} M_n^{0N} \right)^{\left\lfloor \frac{N}{n} \right\rfloor} \sigma. \quad (6)$$

5 Concluding Remarks

We demonstrated how a simple action, locally coordinated, represented here by one-dimensional cellular automata rules, can perform a solution to a complex global computation such as the MOD*n* problem.

The design of rules to compute some task may start from analysing these tasks, and maybe dividing them into smaller instances. For each minor task, we can select a single or a group of active state transitions to execute it.

Different tasks performed by different CA rules, can be merged in a single rule, as long as there is no conflict among the individual tasks; this was the case of the task performed by the *Replacement* rules and the one performed by the *Grouping* rules in the MOD3 problem.

Much study is still necessary for the understanding and the generalisation of these mergings. However, in our work we could realise some evidence which might help further studies.

Rules classified with fixed point or null dynamical regimes according to [7] yield a predictable behaviour and render themselves, in some cases, to performing specific tasks. These rules are good candidates to have their active transitions joined or even separated, generating other rules that perform the same task. In contrast, rules with complex, chaotic or periodic dynamical behaviours are not candidates for these compositions or decompositions of their active transitions.

Even for the candidate rules, it is not always possible to merge their tasks into a single rule, especially when there is an overlap among the active transitions, or when two tasks performed simultaneously do not perform as required; this is what occurred, for instance, in our attempt to join the two *Grouping* rules, where one of them grouped isolated bits and the other grouped isolated pairs of bits in the MOD4 problem.

In order to merge rules for performing equivalent operation, we modified tasks and solved conflicts, following a standard that allowed us to generalise the MOD*n* solution. As *n* increases, in order to solve the MOD5 problem, for instance, the number of synchronised displacements caused by merging *Grouping* rules also increases. So, in order to merge satisfactorily rules $G^{0,3}_{1*}$, $G^{0,2}_{2*}$ and G^{0}_{3*}, or rules $G^{1,3}_{1*}$, $G^{1,2}_{2*}$ and G^{1}_{3*}, we have to disable active transitions in the rule pairs involved in synchronised displacements of strings, as shown below (where G^{-} stands for either G^0 or G^1):

- 10001001 (or 01110110): G^{-}_{3*} and $G^{-,2}_{2*}$ (a single step for each rule);
- 1000101 (or 0111010): G^{-}_{3*} and $G^{-,3}_{1*}$ (a single step for each rule);
- 11001101 (or 00110010): $G^{-,2}_{2*}$ and $G^{-,3}_{1*}$ (2 steps for each rule);
- 0100101 (or 1011010): $G^{-,2}_{2*}$ and $G^{-,3}_{1*}$ (a single step for each rule); and
- 1100101 (or 0011010): $G^{-,2}_{2*}$ and $G^{-,3}_{1*}$ (2 steps for $G^{-,2}_{2*}$ and 1 for $G^{-,3}_{1*}$).

For the suitability of M_5^0 and M_5^1, we then have to disable 5 pairs of groups of active transitions. In general, this number depends on *n*, as exemplified in Figure 9.

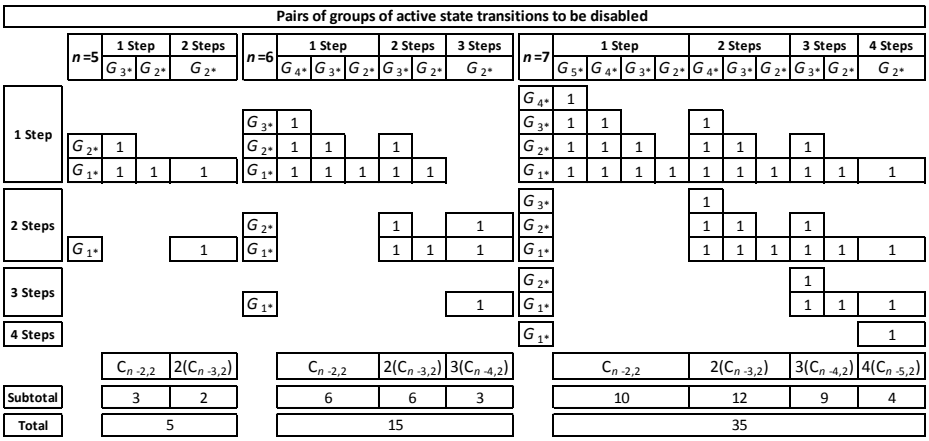

Figure 9: Number of groups of active state transitions to be disabled according to *n*.

The number of pairs of groups of active transitions to be disabled for each merging rule is $C_{n-2,2} + 2C_{n-3,2} + 3C_{n-4,2} + \ldots + (n-3)C_{2,2}$ which can be calculated through the 4th degree polynomial $((n-3)^4 + 6(n-3)^3 + 11(n-3)^2 + 6(n-3))/24$. All these conflicts can be eliminated, and the merging process can always be carried out.

Acknowledgements: We are grateful to CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico, and IPM – Instituto Presbiteriano Mackenzie.

References

1. C.L.M. Martins and P.P.B. de Oliveira. "Computing Modulo-*n* by Composing Cellular Automata Rules". Under submission to *Fundamenta Informaticae*, 2015.
2. C.L.M. Martins and P.P.B. de Oliveira. "Improvement of a result on sequencing elementary cellular automata rules for solving the parity problem". *Electronic Notes Theoretical Computer Science*, 252:103–119, 2009.
3. H. Xu, K.M. Lee and H.F. Chau. "Modulo three problem with a cellular automaton solution". *International Journal of Modern Physics C*, 14(03):249-256, 2003.
4. K.M. Lee, H. Xu and H.F. Chau. "Parity problem with a cellular automaton solution". *Physical Review E*, 64:026702/1-026702/4, 2001.
5. H. Betel, P.P.B. de Oliveira and P. Flocchini. "Solving the parity problem in one-dimensional cellular automata". *Natural Computing*, 12(3):323-337, 2013.
6. S. Wolfram. *A New Kind of Science*, Wolfram Media, 2002.
7. W. Li. "Parameterizations of Cellular Automata Rule Space". *SFI Technical Report: Preprints, Santa Fe, NM, USA,* 1991.