Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A sequential, multi-symbol cellular automaton that solves binary density classification, and its generalisation as a Turing machine

Pacôme Perrotin

pacome.perrotin@gmail.com

École Normale Supérieure de Lyon

Pedro Paulo Balbi

Universidade Presbiteriana Mackenzie

Eurico Ruivo

Universidade Presbiteriana Mackenzie

Research Article

Keywords:

Posted Date: May 26th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8997295/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

A sequential, multi-symbol cellular automaton that solves binary density classification, and its generalisation as a Turing machine

*Pacôme Perrotin[1] , Pedro Paulo Balbi[2] , and Eurico Ruivo[2]

[1]École Normale Supérieure de Lyon, France
[2]Univ. Presbiteriana Mackenzie - Faculdade de Computação e Informática
pacome.perrotin@gmail.com, pedrob@mackenzie.br,
eurico.ruivo@mackenzie.br

Abstract

First, we present a sequential cellular automaton of radius $\frac{1}{2}$ as a solution to the density classification task that makes use of an intermediate alphabet, and converges to a clean fixed point with no remaining auxiliary or intermediate information. We then extend this solution to arbitrary finite alphabets and to configurations in higher dimensions. Finally, we provide a similar construction, now with a cellular automaton with radius 1, that can simulate arbitrary Turing machines in bounded space.

1 Introduction

Cellular automata are computational models characterised by a local rule uniformly applied over a configuration. The straightforwardness of their definition makes them the subject of a wide array of applications to the modeling of complex physical phenomena [7, 14, 1, 20].

Among some of the theoretical questions studied on cellular automata, classification tasks [15] can be compared to decision problems defined on Turing machines. Such a task is defined by a function that assigns any initial configuration $x \in \mathbb{B}^n$ to a binary value b. A cellular automaton is a solution to a classification task if while executed on the initial configuration x it converges to the uniform configuration b^n, and this configuration is a fixed point. The two most studied classification tasks are the parity classification task, where b is 1 if and only if the initial configuration contains an odd number of 1s, and the density classification task, where b is the symbol in strict majority in the initial configuration.

*Corresponding author.

Various solutions to the parity classification task exist [19, 16, 12, 2, 5], including a pure solution in the form of a cellular automaton of radius 4 [3]. However, the density classification task has been proven to not have such a pure solution [10]. The current state of the art contains various solutions approaching the problem from different angles, including changing the local rule during the computation [13] or using multiple tracks in parallel to provide a solution [8]. Solutions have also been constructed by using a continuous space of values [4, 11], or by using a stochastic approach [6]. The present work differs from these solutions by being a deterministic solution that always converges to a uniform Boolean configuration.

This paper is structured by first detailing the specifications of the variant of the density classification task presently under investigation. We then offer a detailed and illustrated description of our novel solution, and offer generalisations to the solution to larger alphabets and configurations in more dimensions. Finally, we construct a variant of the solution that is able to simulate arbitrary Turing machines within the bounded space of the configuration.

In addition to formal proofs, we tested the one-dimensional binary solution to the density classification task on all initial configurations up to size 30, the source code for which can be freely accessed online [17], written in Rust.

2 Definitions

- *Configuration:*
  For Σ a finite alphabet and n a positive integer, we define a *configuration* of size n any $x \in \Sigma^{\{0,\dots,n-1\}}$. For any integer $0 \leq i < n$, we denote x_i the cell of index i in the configuration x. Indexes of configurations are always defined modulo n, i.e., $x_{-1} = x_{n-1}$, and $x_{n+1} = x_1$.

- *Cellular Automaton:*
  For r a positive integer, we refer to *local function* of radius r any function $f : \Sigma^{\{-r,\dots,0,\dots,r\}} \to \Sigma$. For f such a local function, and n a positive integer, the *Cellular Automaton* F defined by f is $F : \Sigma^{\{0,\dots,n-1\}} \to \Sigma^{\{0,\dots,n-1\}}$, where $F(x)_i = f(x_{i-r},\dots,x_i,\dots,x_{i+r})$. We define the particular case of a local function of radius $\frac{1}{2}$ as a local function $f : \Sigma^{\{-1,0\}} \to \Sigma$.

- *Sequential Update Schedule:*
  For x a configuration of size n, f a local function of radius r, and $i < n$ a non-negative integer, we say that the *update of i in x* is the configuration x', where $x'_k = f(x_{k-r},\dots,x_{k+r})$, if and only if $k = i$, and $x'_k = x_k$ otherwise. The *sequential* update of x under f is the configuration obtained by updating every cell in $\{0,\dots,n-1\}$ in order, and in sequence.

- *Fixed Point:*
  For any configurations x and y, x is said to converge to y if and only if $F^k(x) = y$ for some positive integer k. We call x a *fixed point* if

$F(x) = x$. Note that if x is a fixed point, then the sequential update of x is also x.

3 The problem and the general description of the solution

3.1 The problem

In this paper, we focus on a non-standard solution of the density classification task, as it relies on a variant cellular automaton and the use of an intermediate alphabet, that is, the full alphabet Σ of the solution is a super set of the input alphabet, $\mathbb{B}$.

The formulation of the problem is as follows: for any finite initial configuration over $\mathbb{B}$ that has a defined density (a symbol that outnumbers every other one), then the lattice of the cellular automaton must converge to the uniform configuration containing only that symbol, which must be a fixed point. If the initial solution does not have a defined density, then no expectation is set on the convergence.

Since there is no imposition on what would happen in between the initial configuration and the fixed point, a solution to the problem is free to use any number of symbols from its extended alphabet. Solutions relying on this freedom have been found in the past, namely [8]. This particular solution uses two tracks, one that contains the initial configuration and the second that converges to the desired fixed point, the former guiding the convergence of the latter. We consider the solution presented herein as an improvement because it always converges to a fixed point with no remaining auxiliary or intermediate information, whereas that two-tracks solution only ensures this clean result in one of its tracks.

3.2 The overall idea of the solution

Our solution is a cellular automaton of radius $\frac{1}{2}$ (the reference cell being the one on the right), updated sequentially, meaning that the local rule is applied using the value of the current cell, plus the value of the cell to its left, while the cellular automaton as a whole is updated from left to right. In essence, this cellular automaton simulates an operating head that moves along the configuration while updating an internal memory and transforming the configuration as it proceeds.

With its radius $\frac{1}{2}$, our solution can appear to belong in the family of one-way cellular automata [9]. However, it is updated sequentially and not in parallel, and operates on cyclic configurations.

In the following, a cycle is a complete loop of the operating head around the configuration. This cellular automaton tries to remove exactly one 0 and one 1 from the configuration per cycle, storing them in its internal memory. It then discards this pair when a cycle is complete. The detection of the end of a cycle is done thanks to an internal counter. At the end of a cycle, if only one symbol was found (say, a 0), it then means that there is no more of the other symbol in the configuration (so, no more 1s). Since symbols have

been removed in equal amounts, the cellular automaton can now converge to the uniform configuration containing the symbol it still holds in memory (here, it would converge to 0^n). All these extra pieces of information – the internal memory, the internal counter, and the marker that allows to know which symbols have been removed from the configuration – are encoded using the aforementioned intermediate alphabet.

So, alongside the Boolean alphabet $\mathbb{B} = \{0,1\}$, based upon which the initial configuration and the final fixed points are defined, a key aspect of the solution is that it relies on an intermediate alphabet composed of the following 16 triplets:
$\begin{pmatrix}\circ\\0\\\{0\}\end{pmatrix}$, $\begin{pmatrix}\circ\\0\\\mathbb{B}\end{pmatrix}$, $\begin{pmatrix}\circ\\1\\\{1\}\end{pmatrix}$, $\begin{pmatrix}\circ\\1\\\mathbb{B}\end{pmatrix}$, $\begin{pmatrix}\circ\\X\\\varnothing\end{pmatrix}$, $\begin{pmatrix}\circ\\X\\\{0\}\end{pmatrix}$, $\begin{pmatrix}\circ\\X\\\{1\}\end{pmatrix}$, $\begin{pmatrix}\circ\\X\\\mathbb{B}\end{pmatrix}$, $\begin{pmatrix}\bullet\\0\\\{0\}\end{pmatrix}$, $\begin{pmatrix}\bullet\\0\\\mathbb{B}\end{pmatrix}$, $\begin{pmatrix}\bullet\\1\\\{1\}\end{pmatrix}$, $\begin{pmatrix}\bullet\\1\\\mathbb{B}\end{pmatrix}$, $\begin{pmatrix}\bullet\\X\\\varnothing\end{pmatrix}$, $\begin{pmatrix}\bullet\\X\\\{0\}\end{pmatrix}$, $\begin{pmatrix}\bullet\\X\\\{1\}\end{pmatrix}$ and $\begin{pmatrix}\bullet\\X\\\mathbb{B}\end{pmatrix}$.

Therefore, the intermediate alphabet is actually composed of triplets of symbols, organised in three layers, one on top of the other.

At the top, the symbols $\{\circ,\bullet\}$ form an internal counter of the number of cycles. Each cycle has a uniform top layer, with either only $\circ$ symbols or only $\bullet$ symbols, and is always succeeded by a cycle with a uniform top layer using the opposite symbol. Alternating between these two symbols at each new cycle allows the solution to tell if any triplet and its left neighbour originate from the same cycle.

In the middle layer, the symbol in $\{0,1,X\}$ represents the value contained at any particular location of the configuration. The symbol X indicates the positions in the configuration where a pair of 0 and 1 have been removed.

As for the bottom layer, the symbols $\{\varnothing,\{0\},\{1\},\mathbb{B}\}$ are related to the memory of the moving head of the machine encoded in the cellular automaton. This memory is a set that can store one individual 0 and one individual 1, which are taken from the configuration. A memory of value $\varnothing$ means no symbol has yet been taken from the configuration in the current cycle, while a memory of $\{0\}$ (respectively, $\{1\}$) means that only a 0 symbol (respectively, a 1 symbol) has been taken in the current cycle, and a memory of $\mathbb{B}$ means both a 0 and a 1 have been taken in the current cycle. While a memory symbol appears at the bottom layer of every intermediate symbol of a configuration, not all of them are meaningful. The memory symbol considered to be the actual memory of the automaton (its *active memory*) is the one that has been added most recently in the configuration.

Based on the previous description of the layers, it is clear that there are indeed $2\times3\times4=24$ possible triplets; however, since the triplets $\begin{pmatrix}\circ\\0\\\varnothing\end{pmatrix}$, $\begin{pmatrix}\circ\\0\\\{1\}\end{pmatrix}$, $\begin{pmatrix}\circ\\1\\\varnothing\end{pmatrix}$, $\begin{pmatrix}\circ\\1\\\{0\}\end{pmatrix}$, $\begin{pmatrix}\bullet\\0\\\varnothing\end{pmatrix}$, $\begin{pmatrix}\bullet\\0\\\{1\}\end{pmatrix}$, $\begin{pmatrix}\bullet\\1\\\varnothing\end{pmatrix}$ and $\begin{pmatrix}\bullet\\1\\\{0\}\end{pmatrix}$ are never used, they are not considered part of the intermediate alphabet, bringing the total down from 24 to 16.

A warning should be made to the reader at this point: from now on we will refer to the triplets of the intermediate alphabet as its *symbols*, which should not be confused with the individual symbols they are made of, as already mentioned; so, in a sense we will be stretching the meaning of the word 'symbol' associated to the components of the intermediate alphabet, just to preserve the standard jargon associated to the notion of an alphabet, that is usually said to be composed of symbols.

3.3 Mechanics of the process

Since the constructed cellular automaton must have 0^n and 1^n as fixed points, this implies that the state transitions $00 \mapsto 0$ and $11 \mapsto 1$ must be part of the local rule. This represents an action embedded in the rule that we can refer to as *fixed point preservation*; other actions are represented in phases of the process which we refer to as *kickstart*, *propagation*, *swap* and *convergence*, as explained below.

- The *kickstart* phase is applied when 01 or 10 is detected on the initial configuration. These patterns are proof that the configuration is not uniform, and thus the cellular automaton kickstarts its computation of the density of the configuration by inserting a symbol from the intermediate alphabet. In particular, the neighbourhood 01 maps to $\binom{\circ}{\{1\}}$ and 10 maps to $\binom{\circ}{\{0\}}$. In the 01 case, the resulting symbol $\binom{\circ}{\{1\}}$ shows that the symbol 1 is no longer in the configuration at this location, has been replaced by X, and has been placed in the active memory $\{1\}$. Furthermore, we initialise the internal counter to the value $\circ$, which indicates an odd number of cycles. An equivalent logic applies to the case 10, where the 0 is removed and placed in memory instead.

- The *propagation* phase applies directly after the kickstart phase, but also in any case of the form ab where a is an intermediate symbol and b is either in $\mathbb{B}$ or is an intermediate symbol, but with a different internal counter from a. The propagation phase represents the majority of the updates once the process has started, and its function is to propagate the counter from the left to the right, and also to remove the current character of the configuration and to add it to the memory, if possible.

  For example, the pair $\binom{\circ}{\{1\}}$ 0 maps to the symbol $\binom{\circ}{\mathbb{B}}$. In essence, the current symbol 0 has been replaced by an intermediate symbol, in which the symbol 0 has been removed and placed in the memory, and the counter $\circ$ has been propagated from the left. The memory is now $\mathbb{B} = \{0,1\}$, which means a complete pair has been collected. Another typical case of the propagation phase is associated to the neighbourhood $\binom{\bullet}{\{0\}}\ \binom{\circ}{\mathbb{B}}$, which maps to $\binom{\bullet}{\{0\}}$; here, the new character is obtained by propagating the counter and the memory from the left. As the current character 0 is already present in the memory, it is not removed. Note that the previous memory $\mathbb{B}$ date from the previous cycle, as indicated by the outdated counter $\circ$, and can thus be deleted in the process of the propagation.

  To reiterate, the propagation phase serves two purposes: to propagate the current counter and the active memory from left to right, and to remove any character it can out of the configuration and place them in the active memory.

- The *swap* phase happens any time where the updated cell is at the right hand side of another cell with an identical counter, such as in $\binom{\circ}{0}{\mathbb{B}}\ \binom{\circ}{X}{\{0\}}$. What happens now depends on the state of the active memory. As a reminder, the active memory is the memory contained in the cell to the left of the cell which is to be updated. Since in this example, the active memory is $\mathbb{B}$, it is erased and the internal counter increased, therefore leading to the symbol $\binom{\bullet}{X}{\varnothing}$. This serves two purposes: the $\{0,1\}$ pair that was harvested from the configuration in the previous cycle is deleted, and the internal counter is updated, allowing for another propagation phase.

  Two remaining cases are possible: the active memory could be empty, or a singleton. If the memory is empty, this is the situation where the initial configuration had an equal amount of 0s and 1s. As discussed previously, the solution is not required to converge in this case and thus this is classified as an undefined behaviour. However, if the active memory is a singleton $\{b\}$, the automaton now has to converge to the uniform configuration b^n. This is done by simply updating to the symbol b. For example, the neighbourhood $\binom{\bullet}{1}{\{1\}}\ \binom{\bullet}{X}{\varnothing}$ would be updated to the symbol 1. The rest of the convergence is achieved through the next and final phase.

- The *convergence* phase is straightforward, and happens at any time where the left hand symbol is in $\mathbb{B}$ and the one at the right is an intermediate symbol, in which case the left hand symbol is simply propagated, as happens, for example, with the neighbourhood $0\ \binom{\circ}{X}{\{0\}}$ which maps to 0. This simple propagation method leads to a uniform configuration, in our example 0^n, which is a fixed point and concludes the execution of the process.

A more formal specification of the local rule is presented in Table 1. Executions of the solution on configurations of size 7 and 13 are represented in Figures 1 and 2, respectively.

4 Proofs

For what follows, we first introduce a few technical terms:

- F is the solution previously introduced in the paper.
- For any configuration x of length n, $F(x)$ denotes the update of x through the solution.
- For any integer $k \in \mathbb{N}$, $F^k(x)$ is the repetition of the application of F on x, k times.
- Since our solution works in the sequential update schedule, we will use a notation to denote parts of an update. For any integer $i \in \{1,\dots,n\}$, $F^{k,i}(x)$ denotes the configuration obtained after k applications of function F, followed by the application of F's local rule on cells 1 to i, in

Pattern	Case Identification	Example
$00 \mapsto 0,\ 11 \mapsto 1$	Fixed point behaviour	$00 \mapsto 0$
$0\,1 \mapsto \begin{pmatrix} \circ \\ X \\ \{1\} \end{pmatrix},\ 1\,0 \mapsto \begin{pmatrix} \circ \\ X \\ \{0\} \end{pmatrix}$	Kickstart	$0\,1 \mapsto \begin{pmatrix} \circ \\ X \\ \{1\} \end{pmatrix}$
$\forall M \subseteq \mathbb{B},\ b \in \mathbb{B} \setminus M,\ c \in \{\bullet, \circ\},\ \begin{pmatrix} c \\ ? \\ M \end{pmatrix}\ b \mapsto \begin{pmatrix} c \\ X \\ M \cup \{b\} \end{pmatrix}$	First propagation phase and removal of a character	$\begin{pmatrix} \circ \\ 0 \\ \{0\} \end{pmatrix}\ 1 \mapsto \begin{pmatrix} \circ \\ X \\ \mathbb{B} \end{pmatrix}$
$\forall M \subseteq \mathbb{B},\ b \in M,\ c \in \{\bullet, \circ\},\ \begin{pmatrix} c \\ ? \\ M \end{pmatrix}\ b \mapsto \begin{pmatrix} c \\ b \\ M \end{pmatrix}$	First propagation phase without removing a character	$\begin{pmatrix} \bullet \\ 1 \\ \{1\} \end{pmatrix}\ 1 \mapsto \begin{pmatrix} \bullet \\ 1 \\ \{1\} \end{pmatrix}$
$\forall c, c' \in \{\bullet, \circ\},\ c \neq c',\ \begin{pmatrix} c \\ ? \\ \mathbb{B} \end{pmatrix} \begin{pmatrix} c \\ ? \\ ? \end{pmatrix} \mapsto \begin{pmatrix} c' \\ X \\ \varnothing \end{pmatrix}$	Swap phase and reset of the memory	$\begin{pmatrix} \bullet \\ X \\ \mathbb{B} \end{pmatrix} \begin{pmatrix} \bullet \\ X \\ \{0\} \end{pmatrix} \mapsto \begin{pmatrix} \circ \\ X \\ \varnothing \end{pmatrix}$
$\forall M \subseteq \mathbb{B},\ b \in \mathbb{B} \setminus M,\ c, c' \in \{\bullet, \circ\},\ c \neq c',\ \begin{pmatrix} c \\ ? \\ M \end{pmatrix} \begin{pmatrix} c' \\ b \\ ? \end{pmatrix} \mapsto \begin{pmatrix} c \\ X \\ M \cup \{b\} \end{pmatrix}$	Propagation phase and removal of a character	$\begin{pmatrix} \bullet \\ X \\ \varnothing \end{pmatrix} \begin{pmatrix} \circ \\ 1 \\ \mathbb{B} \end{pmatrix} \mapsto \begin{pmatrix} \bullet \\ X \\ \{1\} \end{pmatrix}$
$\forall M \subseteq \mathbb{B},\ b \in M,\ c, c' \in \{\bullet, \circ\},\ c \neq c',\ \begin{pmatrix} c \\ ? \\ M \end{pmatrix} \begin{pmatrix} c' \\ b \\ ? \end{pmatrix} \mapsto \begin{pmatrix} c \\ b \\ M \end{pmatrix}$	Propagation phase without removal of a character	$\begin{pmatrix} \circ \\ X \\ \mathbb{B} \end{pmatrix} \begin{pmatrix} \bullet \\ X \\ \{0\} \end{pmatrix} \mapsto \begin{pmatrix} \circ \\ X \\ \mathbb{B} \end{pmatrix}$
$\forall b \in \mathbb{B},\ c \in \{\bullet, \circ\},\ \begin{pmatrix} c \\ ? \\ \{b\} \end{pmatrix} \begin{pmatrix} c \\ ? \\ ? \end{pmatrix} \mapsto b$	Swap phase and starting the convergence	$\begin{pmatrix} \circ \\ 0 \\ \{0\} \end{pmatrix} \begin{pmatrix} \circ \\ X \\ \varnothing \end{pmatrix} \mapsto 0$
$\forall b \in \mathbb{B},\ b \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \mapsto b$	Convergence phase	$1 \begin{pmatrix} \circ \\ 1 \\ \mathbb{B} \end{pmatrix} \mapsto 1$

Table 1: The different cases of the proposed solution to the density classification task using an intermediate alphabet. The propagation phase is described by 4 different sets of cases, and the swap phase by 2. The symbol '?' in patterns represents a wildcard.

order. In effect, $F^{k,n}(x)$ is equivalent to $F^{k+1}(x)$. We denote $|x|_0$ and $|x|_1$ the number of 0 and 1 symbols in the configuration x.

We call the *active cell* the cell to the left of the next cell to be updated. If we consider some configuration $F^{k,i}(x)$ taken at some point of an execution, the active cell is the cell number i. If the symbol contained in that cell is from the intermediate alphabet, its memory part is called the *active memory* of the automata.

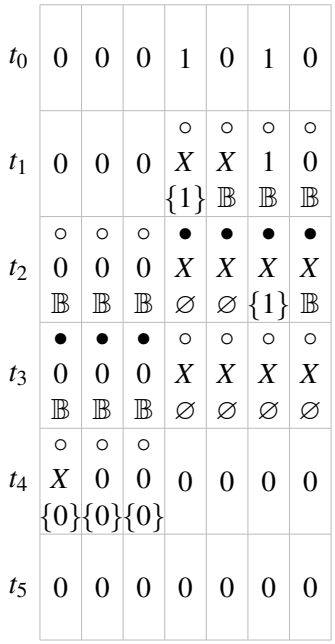


Figure 1: An execution of the proposed solution to a configuration of size 7, in the form of a table with steps labelled from t_0 to t_5. The first row contains the initial configuration, and the execution operates from top to bottom. The first propagation phase starts at line 2, column 4. This propagation phase is followed by a second one with internal counter $\bullet$, and together they remove two symbols 0 and two symbols 1 from the configuration, leaving only 0 symbols. The third propagation phase ends with the active memory $\{0\}$, which leads to the convergence phase starting at line 5.

For the lemma that follows, we specify that a propagation phase starts when the first symbol with an updated internal counter is inserted, and lasts until the configuration has a uniform internal counter. The convergence phase at the end of an execution is not considered as a propagation phase. We refer to as a rule case as an application of the local rule. For example, by propagation rule case we mean the application of the local rule in the propagation phase.

Lemma 1. *Throughout any propagation phase, the count of* 0 *and* 1 *symbols in the configuration plus the symbols contained in the active memory are preserved.*

Proof. At the start of the propagation phase, the active memory is $\varnothing$. Thus, the count of 0 and 1 symbols at this point is exactly what is contained in the configuration. Starting from there and at each step, the local rule attempts to remove the local symbol in the configuration and to place it in memory, if possible. If this is the case, the removed symbol would then be contained in the active memory, preserving the count. If no symbol is taken, the ac-

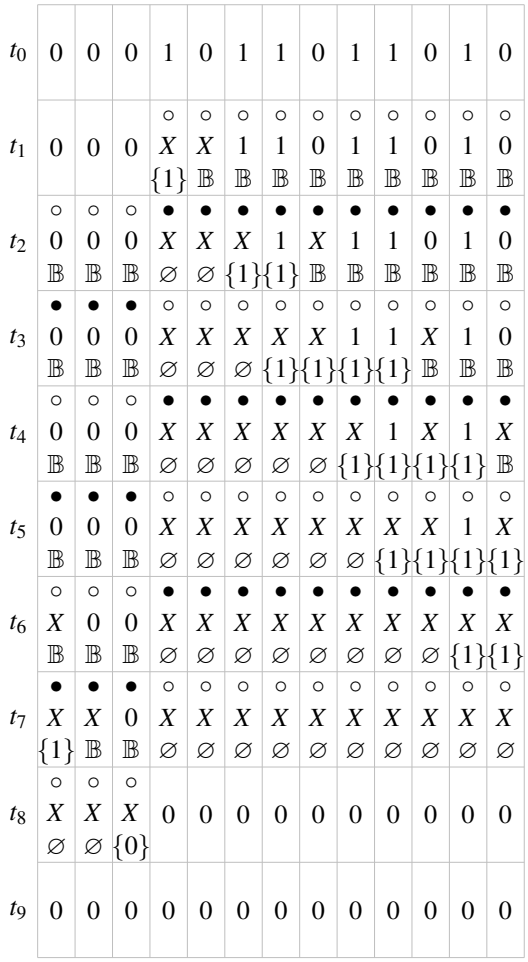


Figure 2: An execution of the proposed solution to a configuration of size 13, in the form of a table, with steps labelled from t_0 to t_9. The first row contains the initial configuration, and the execution operates from top to bottom. This execution contains a total of 7 propagation phases, which is also the count of 1 symbols in the starting configuration, plus one.

tive memory is simply copied over and the configuration is not altered, thus conserving the count. □

Lemma 2. *Let x be a configuration and k and i be integers such that $F^{k,i}(x)$ is a configuration which has completed p propagation phases and has not started a convergence phase. The count of* 0 *and* 1 *symbols in the configuration and the active memory of $F^{k,i}(x)$ are $|x|_0 - p$ and $|x|_1 - p$.*

Proof. We observe that at the end of a propagation phase, if the active memory is $\mathbb{B}$, a swap is applied that starts a new propagation phase and resets the active memory to $\varnothing$. Thus, if at the end of a propagation phase the count of

the 0 and 1 symbols shared between the configuration and the active memory was $(|F^{k,i}(x)|_0, |F^{k,i}(x)|_1)$, the count starting in the new propagation phase has to be $(|F^{k,i}(x)|_0 - 1, |F^{k,i}(x)|_1) - 1$. Lemma 1 then tells us that this count is preserved until the start of the next propagation phase.

Assuming that $|x|_0$ and $|x|_1$ are both higher than p, a sequence of p propagation phases is observed, which implies a final count of $|x|_0 - p$ and $|x|_1 - p$. □

Theorem 1. *The sequential cellular automaton F is a solution to the density classification task using an intermediate alphabet.*

Proof. Let $x \in \mathbb{B}^n$ be an initial configuration. If $|x|_0 = |x|_1$, then no special behaviour is expected. We now assume that $|x_0| \neq |x_1|$. If x is uniform in value, then it is a fixed point of F, which is a required behaviour. We now assume that x is not uniform.

This implies that a pattern 01 or 10 appears somewhere in the configuration, which implies that the kickstart rule case applies at some point, which in turns implies a number of propagation phases. We now argue that exactly $min(|x|_0, |x|_1) + 1$ propagation phases happen in the execution, followed by a convergence phase.

As shown in Lemma 2, the succession of p propagation phases removes p symbols 0 and 1 from the configuration and the active memory. We argue that as long as the configuration and active memory contains a symbol 0 and 1 at the beginning of a propagation phase, then the active memory will be $\mathbb{B}$ at the end of a propagation phase. This can be seen because all propagation rule cases try to capture symbols from the configuration into the memory at any occasions, and all symbols of the configurations are considered this way during a propagation phase. Thus, as long as the configuration and active memory contains both a 0 and 1, another propagation phase will follow the current one. It follows that a sequence of $min(|x|_0, |x|_1)$ propagation phases happen in a sequence, after which there will be no more 0 or 1 left in the configuration and active memory, which leads to an extra propagation phase which ends with an active memory of the form $\{b\}$, which then leads to a convergence phase.

When this is the case, we argue that the remaining symbol (either 0 and 1) had a greater count in the initial configuration than the other one, which implies that it was the majority symbol in the initial configuration. Furthermore, at the end of the last configuration phase, the active memory $\{b\}$ contains the symbol in majority, and the following convergence phase turns the configuration into the fixed point b^n, which proves that F is a sequential solution to the density classification task, using an intermediate alphabet. □

5 Generalisations

This solution is versatile enough to be generalised to support any finite alphabet to compute a density over, and also to support configurations in higher dimensions. In this section we present the necessary changes to realise these generalisations, along with proofs of their correctness.

5.1 Larger alphabets

Let us consider S some finite alphabet. The generalisation of the density classification task over that alphabet is the task of converging to the uniform configuration s^n if the symbol $s \in S$ is in strict majority over the input configuration $x \in S^n$ of size n. Such a uniform configuration must be a fixed point.

We denote F_S the generalised solution, working on the extended alphabet $\Sigma = S \cup (\{\circ, \bullet\} \times (S \cup \{X\}) \times 2^S)$, where 2^S stands for the power set of S, the set of all the subsets of S. This intermediate alphabet grows exponentially in the size of the input alphabet S. Similarly to the binary case, F_S is a sequential cellular automaton of radius $\frac{1}{2}$.

The rule cases are similar to the ones in the binary solution. For any symbol $s \in S$, the rule $s\ s \mapsto s$ ensures that the uniform configurations are fixed points. Any mismatch $s, s' \in S$ leads to the kickstart rule case $s\ s' \mapsto \begin{pmatrix} \circ \\ X \\ \{s'\} \end{pmatrix}$. The propagation phases operate as normal, moving symbols from the configuration into the active memory when possible. The swap cases lead to another propagation phase as long as the active memory contains more than one symbol, and to a convergence phase if the memory contains exactly one symbol. An empty active memory at the end of a propagation phase leads to undefined behaviour.

Theorem 2. *The sequential cellular automaton F_S is a solution to the density classification task over the set S using an intermediate alphabet.*

Proof. We follow the steps of the proof in the binary case, the main difference being that we count the number of propagation phases in a different way. In each propagation phase, the active memory accumulates one of each symbol of S still in the configuration, and the propagation phases stop when the active memory is a singleton at the end of propagation. Such an event happens when each symbol that is not in majority in the initial configuration has been taken out. This time is exactly the maximum of the counts of the non-majority symbols, or the count of the second most present symbol in the initial configuration. After this exact amount of propagation phases, the configuration only contains X and the majority symbols, and an extra propagation phase leads to a convergence phase. □

5.2 Higher dimensions

For $d \geq 1$ an integer, we generalise the density classification task over an alphabet S to any number d of dimensions. Instead of considering cyclic configurations, we here consider cuboids of sides $(n_1, n_2, \ldots, n_d)$ in d dimensions, which associate a symbol in S to any coordinate $(x_1, x_2, \ldots, x_d)$. The density task in higher dimensions then asks to replace all symbols of this cuboids by the symbol which is in strict majority in the initial configuration.

The generalisation of our solution on d-dimensional configurations, denoted $F_{S,d}$, is still a sequential rule, which explores the space of coordinates in an interative order. We denote it $F_{S,d}$. For example, for $d = 2$ and $n_1 = n_2 = 3$, the update sequence is $((0,0),(1,0),(2,0),(0,1),(1,1),(2,1),(0,2),(1,2),(2,2))$.

The neighbourhood of our solution is extended to contain one neighbour per dimension, plus the cell itself. For $(x_1, x_2, \ldots, x_d)$ the position of a given cell, the neighbours of this cell are those with coordinates $(x_1 - 1, x_2, \ldots, x_d)$, $(x_1 - 1, x_2 - 1, x_3, \ldots, x_d)$, $(x_1 - 1, x_2 - 1, x_3 - 1, x_4, \ldots, x_d)$, ..., $(x_1 - 1, x_2 - 1, \ldots, x_d - 1)$, plus $(x_1, x_2, \ldots, x_d)$, the cell itself. This neighbourhood is a subset of the Moore neighbourhood in d dimensions, as it always contains corners.

The local rule of $F_{S,d}$ is defined as the following generalisation of the one-dimensional solution over S. Firstly, the fixed point behaviour is applied when the neighbourhood is of a uniform value in S. Secondly, the kickstart rule case is applied if the neighbourhood only contains symbols in S but is not uniform.

Thirdly, the propagation phase must be applied by discerning which neighbour was the cell to be last updated in the sequence; or more precisely, it must determine what the active memory is. This is done by considering all the neighbours which are in the intermediate alphabet, and taking the larger memory, by comparing them by set inclusion. We then apply the one-dimensional local rule as if this active memory was in the cell to the left, and the present cell the cell to the right.

If both the present cells and all its neighbours are intermediate symbols with a uniform internal counter, we apply the swap rule case. The active memory is computed as before, by taking the maximum between the memories of the neighbours.

Finally, if the present cell is in the intermediate alphabet and has at least one neighbour with a value in S, we propagate that value to converge to a fixed point.

An execution of our generalised solution over a two dimensional configuration of size 9 over the alphabet $\{0, 1, 2\}$ is shown in Figure 3.

Theorem 3. *The sequential cellular automaton $F_{S,d}$ is a solution to the density classification task over the set S in d dimensions using an intermediate alphabet.*

Proof. This proof works by showing that $F_{S,d}$ boils down to the application of F_S to the d-dimensional configuration as if it were one-dimensional, by reconstructing the active memory from its neighborhood. We will show that this is the case on every phase of the solution.

First, observe that if the configuration is uniform over S, then it is a fixed point of $F_{S,d}$. However, if the configuration is non-uniform over S, then there exist two locations that differ. We claim that if two locations differ, then a kickstart rule case happens in the first update of the configuration. This is true because a path can be made from any two locations of the configuration only using jumps in the local neighbourhood of the cells. If the starting and ending points of such a path have different values, then this path is bound to have two consecutive values that differ, and those two values would be the value of a cell and one of its neighbours, which leads to a kickstart rule case when this cell is updated.

Let us assume that the intermediate symbol introduced in the kickstart rule case is in the cell of coordinates $(x_1, x_2, \ldots x_d)$. Let us show that a prop-

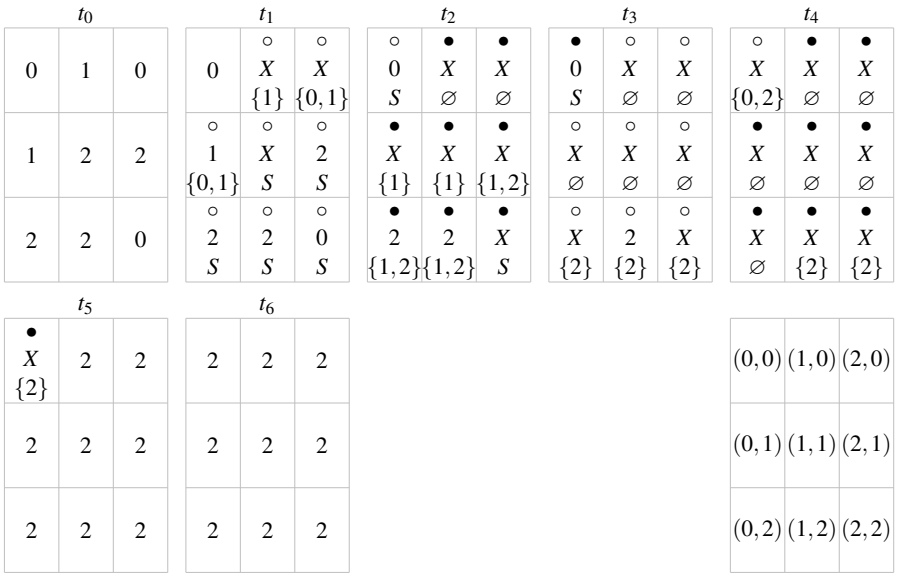


Figure 3: An execution of $F_{\{0,1,2\},2}$ on a 3 by 3 configuration, with the set S of the input alphabet being $\{0,1,2\}$. This execution lasts for 7 configurations, which are illustrated from left to right, top to bottom, and labelled from t_0 to t_6. Within each configuration, the sequential order is from left to right, and from top to bottom. The element at the bottom right of the figure indicates the coordinates of the cells; for instance, the neighbours of $(0,0)$ are the cells $(2,0)$, $(2,2)$ and $(0,0)$ itself. As all these cells have the value 0 in the initial configuration, the cell $(0,0)$ does not change value during the first update.

agation phase now starts and replaces the entire configuration with intermediate symbols.

If we assume that $x_1 < n_1 - 1$, then the next coordinate to be updated is $(x_1+1, x_2, \ldots, x_d)$. This new cell has not been updated yet since the kickstart, and thus contains a symbol in S. The first cell in its neighbourhood is the active cell, and thus the propagation rule case is applied.

Let us now assume that $x_1 = n_1 - 1$, but that $x_2 < n_2 - 1$. This means that the next cell to be updated is of coordinates $(x_1+1, x_2+1, x_3, \ldots, x_d)$. Similarly to before, this cell has not yet been updated and contains a symbol over S. Moreover, its neighbour at coordinate $(x_1, x_2+1, x_3, \ldots, x_d)$ has not been updated either, and the neighbour at coordinate $(x_1, x_2, \ldots, x_d)$ is the active cell, and contains an intermediate symbol. This means that we apply a propagation rule case; to do so, the local rule is defined to apply a propagation phase as in F_S by taking the active memory as the largest memory within its neighbourhood. As the memory only accumulates symbols throughout a propagation phase, taking the largest set (in the sense of set inclusion) is always a defined operation that computes the correct active memory. Note that the configuration symbol of the active cell (in $S \cup X$) is actually never relevant to apply the local rule, and it is not necessary to know its value.

It follows that one step of propagation is applied as it would in the one-dimensional solution F_S.

This logic works for any number of dimensions; when the sequential order jumps in the direction of the k-th dimension, the $k-1$ first neighbours of the new cell have not yet been updated in the current propagation phase, but the k-th neighbour is the active cell, the last cell to be updated. This means that the propagation phase operates as the one dimensional local rule would, up to the point where the entire configuration has been converted to the intermediate alphabet with a uniform internal counter. The new cell to be updated has the active cell within its neighbourhood, which means that it can accurately compute the active memory by taking the largest memory in its neighbourhood. If this active memory is a singleton, this means that the previous propagation phases removed all symbols except one, and the solution can converge. If the active memory is empty, then the density is not defined over the initial configuration and the behaviour is said to be undefined. If the active memory has a size of at least 2, then the solution starts a new propagation phase by emptying the memory and updating the internal counter.

Overall, this solution successfully explores a d-dimensional configuration to apply the local rule as if it were one-dimensional, which leads to the result. □

6 Simulating arbitrary bounded Turing machines

The methods presented so far in this paper are much more general than the resolution of the density classification task. To show this formally, we create in this section a variation of our solution that can be used to execute any Turing machine over the bounded space of the configuration. Such machines do not have infinite space to work with and thus are not universal. To compensate, we will simulate a two-tape Turing machine. As we will show, these machines are able to solve problems such as the density classification task or the parity problem. While the previous construction in this paper used a radius of $\frac{1}{2}$, this new construction is a cellular automaton with radius 1.

Formally, our target Turing machine is a tuple $M = (Q, \Gamma, b, \Sigma = \mathbb{B}, q_0, F, \delta)$ where Q is the set of states, Γ is the tape alphabet, $b \in \Gamma$ is the blank tape symbol, $\Sigma \subseteq \Gamma$ is the input alphabet, q_0 is the initial state while F is the set of final states, and $\delta : Q \times \Gamma \times \Gamma \to Q \times \Gamma \times \Gamma \times \{L, R\}$ is the transition function.

Our simulating cellular automaton is sequential, and its cells are updated from left to right. The different cases of the local function of this simulation are detailed in Table 2. It works as follows.

The initial configuration is binary, so only the rule cases 'Fixed point behaviour' and 'Kickstart' apply. If the initial configuration is uniform, it will stay as such; otherwise, the 'Kickstart' rule cases will eventually apply and insert the initial state of the Turing machine: q_0.

Alongside q_0, two binary values are inserted: the first is the value of the configuration at that point, and the second is part of the extra tape and is initialised at 0. Then, these triplets of values will propagate throughout the configuration thanks to the 'Propagation' rule case. In this phase we do not

insert a state, but instead a • (dot), meaning that no Turing machine head is located here.

The result of this is a configuration made of intermediary symbols, with exactly one state q_0, where the kickstart happened, and the symbol • everywhere else.

Next, the simulation of the Turing machine takes place. When the cellular automaton finds the Turing machine head (which has a state different from •), it will update one step of the Turing machine. This has different behaviours if the machine's state indicate that it should go left or right.

Different rule cases handle these two scenarios. The 'Left move' rule cases start when the automaton finds the Turing machine head to its right. In this case, the result of the move of the Turing machine is written at the current location of the cellular automaton. This means that there are two Turing machine heads in the configuration simultaneously. The second family of 'Left move' cases removes the first Turing machine head and completes the move.

The 'Right move' cases are slightly more complicated. When the cellular automaton arrives at the location of a Turing machine state q that will go right, it indicates this fact using a special $\vec{q}$ symbol. The second family of 'Right move' cases recognises that this special $\vec{q}$ is on its left, and writes the new state of the Turing machine in its current location. If the automaton finds this symbol at the centre or the right of its neighbourhood instead, it is treated as a •, and ignored.

Finally when the Turing machine arrives at either state f_0 or state f_1, the cellular automaton should converge to a uniform Boolean configuration. It does this by first placing a 0 if the state was f_0, and a 1 if the state was f_1. Then, the 'Convergence' rule case simply propagates that Boolean value to the rest of the configuration.

As an example, the execution of our simulation is depicted in Figure 4, which simulates the Turing machine represented in Figure 5. This particular machine is defined on the alphabets $\Sigma = \Gamma = \mathbb{B}$, but that does not have to be the case in general.

This example solves the parity problem, which means that if the initial configuration has an odd number of symbols 1, it converges to a uniform configuration with 1s (a fixed point), or to a uniform of 0s otherwise (another fixed point). We have chosen to solve the parity problem, instead of DCT itself, so that the corresponding Turing machine could be made compact and its execution easier to follow.

7 Closing words

The presented solution pushes the boundary of the classification tasks known to be possible using cellular automata. It proves that using an intermediate alphabet together with a sequential update schedule is enough to solve the density classification task.

Following what the extensive literature available suggests, especially [18], we are of the opinion that a solution of the problem using the synchronous update schedule does not exist. Proving this conjecture – i.e., does there exist a synchronous, multi-symbol cellular automaton that always converges to

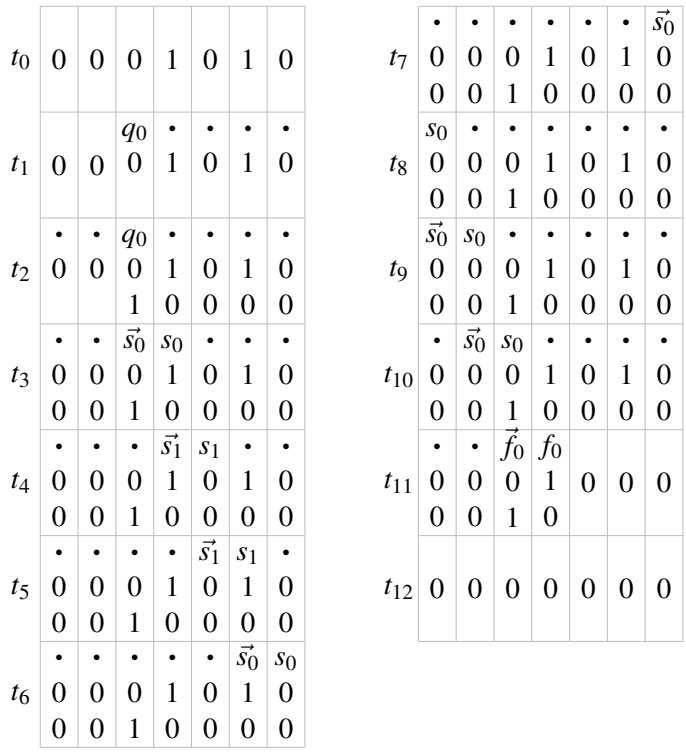


Figure 4: Execution of a Turing machine through our simulation, in 13 steps labelled from t_0 to t_{12}. This Turing machine is detailed in Figure 5. It only goes right, and solves the parity problem.

a binary density classification for any initial configuration, no matter their size? – would be an appealing continuation of the present work, and would confirm that the present solution is in fact one example of some minimal requirements that are sufficient to solve the density classification task.

However, our simulation of a Turing machine tends to indicate that the present space of solutions – sequential, multi-symbol cellular automaton – is in fact very expressive, and it seems surprising to suggest that removing one of these features (replacing sequential for synchronous) would be enough to prevent useful computation.

Acknowledgements

P.P.B. thanks the Brazilian agencies CNPq for the research grant PQ 303356/2022-7, and CAPES for Mackenzie-PrInt research grant no. 88887.310281/2018-00. P.P.B. and E.R. jointly thank CAPES for the research grant STIC-AmSud no. 88881. 694458/2022-01, and P.P. thanks CAPES for the postdoc grant no. 88887. 833212/2023-00, with which most of this work was done, while he was at Universidade Presbiteriana Mackenzie, Brazil.

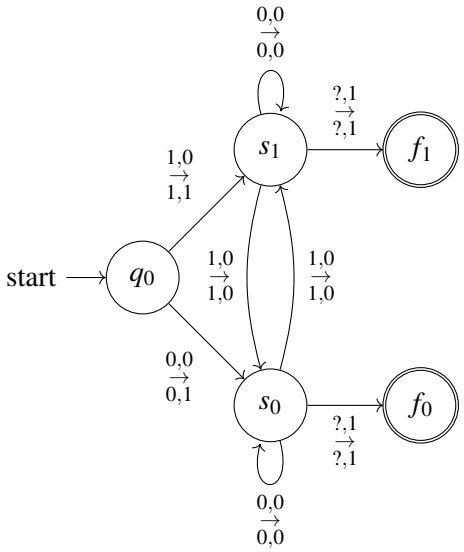


Figure 5: The transition graph of a Turing machine $(Q = \{q_0, s_0, s_1, f_0, f_1\}, \Gamma = \mathbb{B}, b, \Sigma = \mathbb{B}, q_0, F = \{f_0, f_1\}, \delta)$ that solves the parity problem. A transition labelled $\underset{z,w}{\overset{x,y}{\rightarrow}}$ implies that when x is on the input tape and y is on the extra tape, the Turing machine takes this transition, writes z to the input tape and w to the extra tape, and moves to the right.

References

[1] M. M. Aburas, Y. M. Ho, M. F. Ramli, and Z. H. Ash'aari. The simulation and prediction of spatio-temporal urban growth trends using cellular automata models: A review. *International Journal of Applied Earth Observation and Geoinformation*, 52:380–389, 2016.

[2] P. P. Balbi, E. Ruivo, and F. Faria. Synchronous solution of the parity problem on cyclic configurations, with elementary cellular automaton rule 150, over a family of directed, non-circulant, regular graphs. *Information Sciences*, 615:578–603, 2022.

[3] H. Betel, P. P. B. de Oliveira, and P. Flocchini. Solving the parity problem in one-dimensional cellular automata. *Natural Computing*, 12:323–337, 2013.

[4] R. Briceno, P.M. de Espanés, A. Osses, and I. Rapaport. Solving the density classification problem with a large diffusion and small amplification cellular automaton. *Physica D: Nonlinear Phenomena*, 261:70–80, 2013.

[5] Fernando Faria, Eurico Ruivo, and Pedro Paulo Balbi. Generalisation of a synchronous solution of the parity problem on cyclic configurations over a non-circulant graph. *Information Sciences*, 666:120387, 2024.

[6] N.A. Fatès. Stochastic cellular automata solve the density classification problem with an arbitrary precision. In *Symposium on Theoretical Aspects of Computer Science-STACS2011*, volume 9, pages 284–295. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011.

[7] A. G. Hoekstra, J. Kroc, and P. M. A. Sloot. Introduction to modeling of complex systems using cellular automata. In *Simulating Complex Systems by Cellular Automata*, pages 1–16. Springer, 2010.

[8] J. Kari and B. Le Gloannec. Modified traffic cellular automaton for the density classification task. *Fundamenta Informaticae*, 116(1-4):141–156, 2012.

[9] M. Kutrib. Complexity of one-way cellular automata. In *Cellular Automata and Discrete Complex Systems*, pages 3–18, Cham, 2015. Springer, Springer International Publishing.

[10] M. Land and R. K. Belew. No perfect two-state cellular automata for density classification exists. *Physical review letters*, 74(25):5148, 1995.

[11] L. Leal, P. Montealegre, A. Osses, and I. Rapaport. A large diffusion and small amplification dynamics for density classification on graphs. *International Journal of Modern Physics C*, 34(05):2350056, 2023.

[12] K.M. Lee, H. Xu, and H.F. Chau. Parity problem with a cellular automaton solution. *Physical Review E*, 64(2):026702, 2001.

[13] C.L.M. Martins and P. P. B. de Oliveira. Evolving sequential combinations of elementary cellular automata rules. In *European Conference on Artificial Life*, pages 461–470. Springer, 2005.

[14] A. R. Mikler, S. Venkatachalam, and K. Abbas. Modeling infectious diseases using global stochastic cellular automata. *Journal of Biological Systems*, 13(04):421–439, 2005.

[15] M. Montalva-Medel, P. P. B. de Oliveira, and E. Goles. A portfolio of classification problems by one-dimensional cellular automata, over cyclic binary configurations and parallel update. *Natural Computing*, 17:663–671, 2018.

[16] S. Ninagawa. Solving the parity problem with elementary cellular automaton rule 60. *Proc. of AUTOMATA and JAC*, 2012, 2012.

[17] P. Perrotin. Density solution checker. https://github.com/Demonagon/density_checker, August 2024.

[18] E. Ruivo, K. Perrot, PP. Balbi, and P. Perrotin. Negative results on density determination with one-dimensional cellular automata with block-sequential asynchronous updates. *Journal of Computational Science*, 82:102430, 2024.

[19] E. L. P. Ruivo and P. P. B. de Oliveira. A perfect solution to the parity problem with elementary cellular automaton 150 under asynchronous update. *Information Sciences*, 493:138–151, 2019.

[20] D. A. Wolf-Gladrow. *Lattice-gas cellular automata and lattice Boltzmann models: an introduction*. Springer, 2004.

Pattern	**Case Identification**
$000 \mapsto 0,\ 111 \mapsto 1$	Fixed point behaviour
$0\,0\,1 \mapsto \begin{pmatrix} q_0 \\ 0 \\ b \end{pmatrix},\ 0\,1\,0 \mapsto \begin{pmatrix} q_0 \\ 1 \\ b \end{pmatrix},$ $0\,1\,1 \mapsto \begin{pmatrix} q_0 \\ 1 \\ b \end{pmatrix},\ 1\,0\,0 \mapsto \begin{pmatrix} q_0 \\ 0 \\ b \end{pmatrix},$ $1\,0\,1 \mapsto \begin{pmatrix} q_0 \\ 0 \\ b \end{pmatrix},\ 1\,1\,0 \mapsto \begin{pmatrix} q_0 \\ 1 \\ b \end{pmatrix}$	Kickstart
$\forall s \in Q,\ x \in \mathbb{B}\ \begin{pmatrix} s \\ ? \\ ? \end{pmatrix} x\ ? \mapsto \begin{pmatrix} \cdot \\ x \\ b \end{pmatrix}$	Propagation
$\forall s \in Q,\ x,y,z,w \in \Gamma,\ \delta(s,x,y)=(s',?,?,L)\ \begin{pmatrix} \cdot \\ ? \\ ? \end{pmatrix} \begin{pmatrix} ? \\ z \\ w \end{pmatrix} \begin{pmatrix} s \\ x \\ y \end{pmatrix} \mapsto \begin{pmatrix} s' \\ z \\ w \end{pmatrix}$	Left move (first half)
$\forall s,s' \in Q,\ x,x',y,y' \in \Gamma,\ \delta(s,x,y)=(s',x',y',L)\ \begin{pmatrix} s' \\ ? \\ ? \end{pmatrix} \begin{pmatrix} s \\ x \\ y \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \mapsto \begin{pmatrix} \cdot \\ x' \\ y' \end{pmatrix}$	Left move (second half)
$\forall s \in Q,\ x,x',y,y' \in \Gamma,\ \delta(s,x,y)=(s',x',y',R)\ \begin{pmatrix} \cdot \\ ? \\ ? \end{pmatrix} \begin{pmatrix} s \\ x \\ y \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \mapsto \begin{pmatrix} \vec{s'} \\ x' \\ y' \end{pmatrix}$	Right move (first half)
$\forall s \in Q,\ x,y \in \Gamma\ \begin{pmatrix} \vec{s} \\ ? \\ ? \end{pmatrix} \begin{pmatrix} ? \\ x \\ y \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \mapsto \begin{pmatrix} s \\ x \\ y \end{pmatrix}$	Right move (second half)
$\forall x,y \in \Gamma\ \begin{pmatrix} \cdot \\ ? \\ ? \end{pmatrix} \begin{pmatrix} \cdot \\ x \\ y \end{pmatrix} \begin{pmatrix} \cdot \\ ? \\ ? \end{pmatrix} \mapsto \begin{pmatrix} \cdot \\ x \\ y \end{pmatrix}$	Skip
$\forall x,y \in \Gamma\ \begin{pmatrix} \cdot \\ ? \\ ? \end{pmatrix} \begin{pmatrix} \cdot \\ x \\ y \end{pmatrix} \begin{pmatrix} \vec{?} \\ ? \\ ? \end{pmatrix} \mapsto \begin{pmatrix} \cdot \\ x \\ y \end{pmatrix}$	Skip
$\forall x,y \in \Gamma\ \begin{pmatrix} \cdot \\ ? \\ ? \end{pmatrix} \begin{pmatrix} \vec{?} \\ x \\ y \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \mapsto \begin{pmatrix} \cdot \\ x \\ y \end{pmatrix}$	Skip and cleanup
$\begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \begin{pmatrix} f_0 \\ ? \\ ? \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \mapsto 0$	Start converging to 0
$\begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \begin{pmatrix} f_1 \\ ? \\ ? \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \mapsto 1$	Start converging to 1
$\forall x \in \mathbb{B},\ x \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \mapsto x$	Convergence phase

Table 2: The different cases of the proposed simulation of a Turing machine with an intermediate alphabet. The symbol '?' in patterns represents a wildcard, and b is the blank symbol.