Computational Complexity of Finite Asynchronous Cellular Automata

Alberto Dennunzio[a], Enrico Formenti[b], Luca Manzoni[a], Giancarlo Mauri[a], Antonio E. Porreca[a]

[a]*Dipartimento di Informatica, Sistemistica e Comunicazione*
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
[b]*Université Nice Sophia Antipolis, CNRS, I3S, UMR 7271*
06900 Sophia Antipolis, France

Abstract

Cellular Automata (CA) are a well-established bio-inspired model of computation that has been successfully applied in several domains. In the recent years the importance of modelling real systems more accurately has sparkled a new interest in the study of asynchronous CA (ACA). When using an ACA for modelling real systems, it is important to determine the fidelity of the model, in particular with regards to the existence (or absence) of certain dynamical behaviors.

This paper is concerned with two big classes of problems: reachability and preimage existence. For each class, both an existential and a universal version are considered. The following results are proved. Reachability is **PSPACE**-complete, its resource bounded version is **NP**-complete (existential form) or **coNP**-complete (universal form). The preimage problem is dimension sensitive in the sense that it is **NL**-complete (both existential and universal form) for one-dimensional ACA while it is **NP**-complete (existential version) or $\Pi_2^{\mathbf{P}}$-complete (universal version) for higher dimension.

Keywords: natural computing, asynchronous cellular automata, computational complexity

1. Introduction

Cellular Automata (CA) are a well-known and widely used formal model, consisting of a set of state automata arranged on a finite or infinite lattice. A configuration is a snapshot of all the states of the automata. Each automaton updates its state by using a local rule that depends only on the states of the automaton and of its neighbours. The local rule is uniform, i.e., the same for all the automata, and it is applied synchronously, that is, all the automata update their state at the same time. We refer the reader to [15, 9, 1, 8, 7, 2] for an up-to-date overview and recent results on classical CA.

The interest in studying asynchronous CA (ACA) has recently increased because of the necessity of modelling real-life systems. Indeed, in nature few processes are really synchronous, and asynchrony is the common situation. Inspired by the necessity of modelling such processes, many different variations of the classical CA have been defined. We recall, for example, α-asynchronous CA [12], in which a cell is updated at each step with a probability α, fully asynchronous CA [11, 17] (fully-ACA) in which only one cell is updated at each time step, and mACA, an "umbrella" model, that captures the behaviours of many other asynchronous CA models [6, 5].

Cellular automata have been widely used as formal models of complex systems because of their wide variety of dynamical behaviours. Most of these dynamical behaviours are undecidable in the general case [10, 16, 3]. However, in the case of finite configurations, they turn out to be decidable. It is therefore interesting to understand from a computationally complexity point of view how hard is the decision problem for a given dynamical behavior.

Email addresses: `dennunzio@disco.unimib.it` (Alberto Dennunzio), `formenti@unice.fr` (Enrico Formenti), `luca.manzoni@disco.unimib.it` (Luca Manzoni), `mauri@disco.unimib.it` (Giancarlo Mauri), `porreca@disco.unimib.it` (Antonio E. Porreca)

In [20], Sutner classifies the complexity of most of these behaviours. In particular, he shows that the reachability problem is **PSPACE**-complete and the existence of a preimage is **NL**-complete for one-dimensional CA while it is **NP**-complete for higher dimensions.

This paper follows the same trend for both the ACA and the fully ACA models. In any asynchronous setting, of course, one has to cope with the cell updating sequence of the automaton, which strongly interacts with the dynamical behaviour and might influence the computational complexity. Indeed, for any decision problem, two versions are considered: an universal form (i.e., the property is considered w.r.t. to all possible updates) and an existential one (i.e., the property is satisfied by at least one update sequence).

The main problem addressed is reachability, i.e., to establish if a configuration y is in the orbit of another configuration x. Similarly to what happens for classical CA, the reachability problem for ACA is **PSPACE**-complete. However, when a restriction is assumed on the number of time steps in the automaton evolution to reach a configuration y from a configuration x , there is a difference between the synchronous and asynchronous cases. In the former, the problem is **P**-complete, while in the latter it is either **NP** (existential form) or **coNP**-complete (universal form). These results highlight the fact that the order in which the cells are updated can be used to "extract" non-determinism.

Our results rely on a general simulation of non-deterministic Turing machine (TM) inspired by the simulation of a deterministic TM presented in [4]. While the idea of using asynchronous CA to simulate other systems – even other CA – is not new (see, for example [21]), here the asynchrony is not an additional challenge that needs to be solved, but an essential feature of the simulation that is used to perform non-deterministic choices.

The other main class of problems addressed in this paper is linked to the preimage existence. Remark that this problem has additional challenges for ACA and fully-ACA. Indeed, a preimage may exist for a certain choice of cells to update but not for another. The universal and the existential form of the preimage existence problem for one-dimensional fully ACA are in **L**, while the corresponding ones for fully ACA are **NL**-complete.

The same problems turn out to be intractable for ACA in dimension two or higher: the existence of an update sequence for which there exists a preimage is an **NP**-complete problem, while asking whether a preimage exists for all updates is $\mathbf{coNP}^{\mathbf{NP}}$-complete. The last case is particularly interesting since it shows an increase in complexity with respect to the synchronous case, which is **NP**-complete [20].

The paper is structured as follows: Section 2 recalls the necessary notions on CA, ACA, fully-ACA, and Turing machines. Section 3 contains a way to simulate a non-deterministic Turing machine with only a polynomial slowdown by means of ACA and fully-ACA. The complexity of the reachability and reachability in polynomial time problems are then studied and the obtained results are compared with the ones known for classical CA. The preimage problems are the core of Section 4. Finally, a brief summary of the paper and some directions for future research are given in Section 5.

2. Basic Notions

In this section we briefly recall the basic notions on CA, ACA, and fully ACA.

For $a, b \in \mathbb{N}$ with $a \leq b$, denote by $[a, b]$ the set of integers $\{a, a+1, \ldots, b\}$. Let A be a finite alphabet. A (finite) *CA configuration* of length $n \in \mathbb{N}$ is a function $c : [1, n] \to A$, i.e., $c \in A^n$. For any configuration c and any position (*cell*) $i \in [1, n]$ we denote by c_i the element $c(i)$ in that position.

Definition 1. A *one-dimensional CA* is a triple $\mathcal{C} = (A, r, \lambda)$, where A is a finite *alphabet*, $r \in \mathbb{N}$ is the *radius*, and $\lambda : A^{2r+1} \to A$ is the *local rule* of the CA.

A finite CA of size $n \in \mathbb{N}$ is a CA with configurations of length n. The local rule is applied synchronously to all positions of any configuration $c \in A^n$, giving rise to a *global function* $\Lambda : A^n \to A^n$ defined as follows:

$$\forall i \in [1, n], \qquad \Lambda(c)_i = \lambda(c_{i-r}, \ldots, c_i, \ldots, c_{i+r})$$

where *periodic boundary conditions* are used, i.e., $c_j = c_{(j \mod n)+1}$ for positions $j \notin [1, n]$. In this paper, we only deal with finite ACA and fully ACA with periodic boundary conditions. However, all the results also hold in the case of fixed boundary conditions.

The definition of CA can be extended to $d \in \mathbb{N}$ dimensions.

Definition 2. Let $d \in \mathbb{N}$. A d-dimensional CA is a triple $\mathcal{C} = (A, r, \lambda)$, where A is a finite alphabet, $r \in \mathbb{N}$ is the radius, and $\lambda \colon A^{(2r+1)^d} \to A$ is the local rule of the CA.

The definition of CA dynamics and global function immediately generalize to the any dimension.

We are now ready to define ACA and fully-ACA. A finite and instantiated *Asynchronous CA* (ACA) of size n (in any dimension) is a quadruple $(A, r, \lambda, \upsilon)$ where the first three elements define a CA and $\upsilon = (\upsilon_t)_{t \in \mathbb{N} \setminus \{0\}}$ with $\upsilon_t \subseteq [1, n]$ is called *updating sequence*. Every instantiated ACA induces a dynamical behavior described as follows. In dimension 1, the evolution of any configuration $c \in A^n$ is the sequence $\{\Lambda^t(c)\}_{t \in \mathbb{N}}$ where $\Lambda^0(c) = c$ and, for every $t \in \mathbb{N}$, $\Lambda^{t+1}(c)$ is defined as

$$\forall i \in [1, n], \qquad \Lambda^{t+1}(c)_i = \begin{cases} \lambda(\Lambda^t(c)_{i-r}, \dots, \Lambda^t(c)_{i+r}) & \text{if } i \in \upsilon_t \\ c_i & \text{otherwise.} \end{cases}$$

where periodic boundary conditions are used. In other words, at each time step $t > 0$, the local rule f is applied on the positions whose index is contained in υ_t, according to the updating sequence υ. For any two configurations x, y, we say that y is *reachable* from x iff $\Lambda^t(x) = y$ for some $t \in \mathbb{N}$; if y is reachable from x in exactly one step, we say that x is a *preimage* of y.

A fully-ACA is an ACA in which each element in the updating sequence is a singleton set. We address the reader to [17] for an introduction and a study of the dynamical behavior of fully-ACA acting on infinite configurations. All the previous notions easily extend to any dimensions.

When no updating sequence is specified but the triple (A, r, λ) is meant to be the core of an asynchronous automaton, we will refer to *uninstantiated ACA* (resp., *uninstantiated fully-ACA*), or in brief ACA (resp., fully-ACA).

We assume that all the problems presented in this paper use the following encoding for ACA and fully-ACA: The alphabet A is given only as its size (that is, we assume $A = [0, |A| - 1]$) and the local rule λ is given explicitly as a table of $|A|^{(2r+1)^d}$ entries (where d is the dimension of the ACA), each one consisting of an element of A. In this way, the total size of the encoded CA is $s = \Theta\big(\log |A| + |A|^{(2r+1)^d} \times \log |A|\big)$. This assures that $r = \Theta(\log s)$.

A non-deterministic Turing Machine (NDTM) is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_I, Q_F)$, where Q is the set of states, Σ is a finite alphabet of input symbols, $\Gamma \supseteq \Sigma$ is a finite alphabet of symbols that can be written on the machine tape, $\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{\triangleleft, \triangleright\})$ is the transition relation of the machine, and q_I and Q_F are the initial state and set of final states, respectively. With an abuse of notation we will write $\delta(q, \sigma) = (q', \sigma', m)$ if $(q, \sigma, q', \sigma', m) \in \delta$. A Turing machine is deterministic if δ a function. A space-bounded TM has a finite amount of available tape. It is endowed with an ending marker \$ denoting the end of the available space and it never moves its tape head beyond the marker. Without loss of generality, in this paper we assume that, for all TM, each accepting computation ends up into a configuration consisting of a blank tape with the tape head in the first cell and the machine in the unique accepting state.

We briefly recall the complexity classes used in this paper [18]: **L** (resp., **NL**) is the class of decision problems solvable by deterministic (resp., non-deterministic) TM working in logarithmic space, **P** is the class of decision problems solvable by deterministic TM in polynomial time, **NP** (resp., **coNP**) is the class of decision problems solvable by a NDTM in polynomial time and in which the answer is positive if a computation (resp., all computations) accepts. The class of $\mathbf{coNP^{NP}}$, also denoted as $\Pi_2^{\mathbf{P}}$, is the same as **coNP** except that the NDTM has access to a **NP** oracle. Finally, **PSPACE** is the class of decision problems solvable by a deterministic TM working in polynomial space; recall that this class coincides with **NPSPACE**, its nondeterministic counterpart, and with **coNPSPACE**, the class of problems solved by polynomial-space NDTM having positive answer when all their computations are accepting.

3. Simulation of NDTM

In this section we study the reachability problems for asynchronous cellular automata. To deal with them, we provide a way for ACA and fully-ACA to simulate a NDTM by exploiting the nondeterminism implicitly contained in the updating sequence. We show that the asynchronous counterpart of the reachability problem is **PSPACE**-complete, as in the classical CA setting. However, if a constraint on time is added, the complexity changes from **P**-completeness to **NP** or **coNP**-completeness depending on the specific version of the problem.

Definition 3. Let $n, k \in \mathbb{N}$. An infinite sequence $\upsilon = (\upsilon_0, \upsilon_1, \ldots)$ of natural numbers in $[1, n]$ is said to be (n,k)-*universal* if, for each $i \in \mathbb{N}$, its finite subsequence $(\upsilon_i, \ldots, \upsilon_{i+k-1})$ of length k contains each number in $[1, n]$ at least once.

Reachability problems. Given a finite d-dimensional ACA (resp., fully-ACA) $\mathcal{C}$ with n cells and radius r and two configurations $x, y \in A^n$, we consider the following asynchronous counterparts of the classical CA reachability problem:

- The $\exists$-*update reachability problem (existential reachability)* for finite ACA (resp., fully-ACA) consists of deciding whether there exists an updating sequence such that y is reachable from x in the corresponding instantiated ACA (resp., fully-ACA). The total input length is $\Theta\big(\log|A| + |A|^{(2r+1)^d}\log|A| + 2|A|^n\big)$ bits.
- The $\forall$-*update reachability problem (universal reachability)* for finite ACA (resp., fully-ACA) consists of deciding whether for all (n,k)-universal sequences, where k is given in input in unary, y is reachable from x in the corresponding instantiated ACA (resp., fully-ACA). The total input length for this problem is $\Theta\big(\log|A| + |A|^{(2r+1)^d}\log|A| + 2|A|^n + k\big)$ bits.

Given a finite d-dimensional ACA (resp., fully-ACA) $\mathcal{C}$ with n cells and radius r, two configurations $x, y \in A^n$, and a unary string 1^t with $t \in \mathbb{N}$, we define the following *polynomial-time* reachability problems:

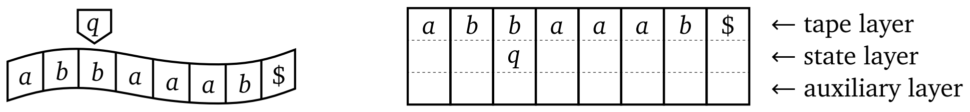
- The $\exists$-*update reachability in polynomial time problem* for finite ACA (resp., fully-ACA) consists of deciding whether there exists an updating sequence such that y is reachable from x *within t steps* in the corresponding instantiated ACA (resp., fully-ACA). The total input length is $\Theta\big(\log|A| + |A|^{(2r+1)^d}\log|A| + 2|A|^n + t\big)$ bits.
- The $\forall$-*update reachability in polynomial time problem* for finite ACA (resp., fully-ACA) consists of deciding whether for all (n,k)-universal sequences, where k is given in input in unary, y is reachable from x *within t steps* in the corresponding instantiated ACA (resp., fully-ACA). The total input length for this problem is $\Theta\big(\log|A| + |A|^{(2r+1)^d}\log|A| + 2|A|^n + k + t\big)$ bits.

Notice that in the universal versions of the problems the updating sequence is required to be (n,k)-universal. Otherwise, it would be easy to find updating sequences that never allow reachability (e.g., those which do not contain a certain cell position) or slow it down arbitrarily (e.g., those including a position only after an exponential, or worse, number of steps). The (n,k)-universality requirement actually allows us a certain degree of fairness in the sequences under investigation, since they force each cell to update at least once every k steps.

NDTM simulation by asynchronous cellular automata. Let $M = (Q, \Sigma, \Gamma, \delta, q_I, Q_F)$ be a NDTM working in space n. Without loss of generality, we assume that $|\delta(q,\sigma)| \le 2$ for all $q \in Q$ and $\sigma \in \Gamma$, i.e., nondeterministic choices are at most binary. Furthermore, when $|\delta(q,\sigma)| = 1$ we assume that the NDTM still performs a non-deterministic choice, but with two identical outcomes, i.e., the non-deterministic bit is ignored. This restriction is useful, as it allows us to simulate deterministic and non-deterministic steps uniformly.

We are now going to build the asynchronous automaton which simulates M. The alphabet of such ACA or fully-ACA is $A = \Gamma \times (Q \cup \{\epsilon\}) \times \{?, 0, 1, \triangleleft, \triangleright, \epsilon\}$. In this way, each element of A is a triple in which the first element represents the content of a tape cell of M, the second element is either the current state of M or empty (i.e., ϵ), and the third element is an auxiliary one that, depending on the current state of the simulation, can be either empty (ϵ), or requesting a non-deterministic bit (i.e., containing the symbol ?), or having a non-deterministic bit (0 or 1), or the movement that the tape head must perform ($\triangleleft$ or $\triangleright$).

The encoding of a configuration of M (which works in space n) is performed in the following way: the corresponding ACA has n cells and the first component of its i-th cell contains the symbol in the i-th position of the tape. The second component is empty (ϵ) except in the cell corresponding to the tape cell scanned by M, that contains the current state of the NDTM. The third component is empty (ϵ) in all the cells. This can be graphically pictured as follows:



The simulation of one step of M is performed by means of multiple steps in the ACA evolution. We describe only the steps involving the cell corresponding to the head position of M (cell i) and the two adjacent cells ($i-1$ and $i+1$). Any other updated cell simply maintains its current state. The different parts of the simulation can be summarized as follows:

1. When cell i is updated, the symbol ? appears in the auxiliary layer. This means that the next phase will be used to obtain a non-deterministic bit of information.
2. The non-deterministic bit is obtained by means of a "race condition" between cell i and the two adjacent cells. If i is updated before or at the same time as one of the adjacent cells, then the value 0 appears in the auxiliary layer. Otherwise, when cell i updates, it will be able to determine that one of the other cells has already updated and will write 1 in the auxiliary layer.
3. Before simulating the transition of M with the newly obtained non-deterministic bit, the cell i waits for cells $i-1$ and $i+1$ to delete the content of the auxiliary layer, thus reverting to the original state.
4. Now, when cell i updates again, it replaces its symbol and state according to the transition table of M and the non-deterministic bit and it writes in the auxiliary layer the movement that the tape head must perform ($\triangleleft$ or $\triangleright$).
5. When the cell toward which the tape head should be moved (i.e., $i+1$ for $\triangleright$, $i-1$ for $\triangleleft$) is updated, it copies to its own state layer the updated state of M from cell i.
6. Finally, when cell i updates again, it deletes the content of the state and auxiliary layer. The new configuration of the ACA now corresponds to one among the at most two possible next configurations of M.

The entire transition table of the local function can be obtained by observing Figure 1, in which the cells that update are highlighted in light gray. The local function on any non-highlighted cell does not modify the content of the cell. Furthermore, configurations marked with $*$ represent updates that can be performed either in parallel (as depicted) or sequentially (by updating the highlighted cells in any order). Finally, configurations marked with † represent updates that are only possible for ACA but not for fully-ACA, as they require multiple cells updating simultaneously; the simulation works even when these updates are unavailable.

Lemma 4. *Let M be a NDTM working in space n and let $\mathcal{C}$ be the ACA (resp., fully-ACA) obtained from M by means of the above construction. For any two configurations y, y' of M, there exists a computation of M from y to y' iff there exists a (n,k)-universal updating sequence υ, where k is polynomial in n, such that in the instantiated ACA (resp., fully-ACA) defined by $\mathcal{C}$ and υ, the encoding of y' is reachable from the encoding of y. Furthermore, the simulation of t steps of M by $\mathcal{C}$ and any (n,k)-universal sequence only requires $O(p(n,t))$ steps, for some polynomial p.*

Proof. Suppose that there exists a computation of M from y to y'. Choose any updating sequence υ' such that the instantiated ACA (resp., fully-ACA) defined by $\mathcal{C}$ and υ', updates every time step at least one (resp., exactly one) the cells highlighted in grey in Figure 1 at each time step, according to the non-deterministic choices that M has to perform. The construction of $\mathcal{C}$ assures that the simulation of one step of M needs 5 to 9 steps of the automaton. Now we need to choose υ' in order to have a (n,k)-universal sequence. Among all the updating sequences υ', consider a sequence υ such that, after having simulated the computation step of M, activate sequentially the cells in $[1,n]$ not already activated in the last $n+9$ steps. By choosing $k=n+9$ we obtain that υ is (n,k)-universal, where k is polynomial in n.

Notice that Figure 1 shows all the positions where an update causes a change of the configuration (by construction, every cell at distance two or more from the cell corresponding to the current tape head position of M does not change state when updated). The particular order in which the cells are updated is immaterial, except when a nondeterministic choice is simulated in the second step of Figure 1. Let υ be a (n,k)-universal updating sequence; then at least one of the positions needed to advance the simulation is activated every k steps (since *all* positions in $[1,n]$ are updated at least once every k time steps). Since the simulation of one step of the TM requires between 5 and 9 updates, it is guaranteed that after $9k$ steps at least one step of the TM has been simulated. If k is polynomial with respect to n, then the simulation can be carried out with a polynomial slowdown.

Vice versa, if the ACA reaches the encoding if y' starting from the encoding of y by means of a (k,n)-universal sequence υ, it is possible to construct a sequence of non-deterministic choices of M such that y' is reached from y.

Such a sequence can be built by looking at which path of Figure 1 is taken according to υ by looking only at the cell corresponding to the position of the head of M and its two neighbours. □

By using the above described NDTM simulation we are now able to prove the following two theorems that show that adding asynchrony to CA does not increase the complexity of the reachability problem, which is **PSPACE**-complete as in the synchronous case [20].

Theorem 5. *The $\exists$-update reachability problem for finite ACA (resp., fully-ACA) is* **PSPACE***-complete.*

Proof. The problem is in **PSPACE** since, if y is reachable from x, then it is reachable in a number of steps that is at most exponential in n, i.e., the number $|A|^n$ of distinct configurations. Therefore, we can count the number of visited configurations in polynomial space. Since **PSPACE** = **NPSPACE**, we can simulate the ACA (resp., fully-ACA) $\mathcal{C}$ with initial configuration x by means of a NDTM working in polynomial space that chooses non-deterministically at every time step a set of positions to update (or only one position if we are considering fully-ACA). If y has not been reached after at most $|A|^n$ time steps then the computation rejects and accepts otherwise. Thus, the problem is contained in **PSPACE**.

As to completeness, we have shown that it is possible to build an ACA (resp., fully-ACA) $\mathcal{C}$ that simulates a TM working in polynomial space. Let x be the configuration of $\mathcal{C}$ encoding the initial configuration of a TM M. We can assume, without loss of generality, that M has a single accepting configuration: this can be guaranteed by letting M erase its tape and moving its head to the leftmost tape position before entering the original accepting state; let y be the encoding of the final accepting configuration for M. By Lemma 4, there exists an updating sequence υ such that y is reachable from x in the instantiated ACA (resp., fully-ACA) if and only if there exists an accepting computation of M on the input encoded by x. Since deciding the latter property is **PSPACE**-hard, the statement of the theorem follows. □

Theorem 6. *The $\forall$-updates reachability problem for finite ACA (resp., fully-ACA) is* **PSPACE***-complete.*

Proof. Let $\mathcal{C}$ be a finite ACA (resp., fully-ACA) of size n. The automaton $\mathcal{C}$ can be simulated starting from any configuration x by a NDTM M which rejects only when it detects a (n,k)-universal updating sequence not leading to the target configuration y. At each time step, M non-deterministically chooses a subsets of cells to be updated (resp., a cell to be updated), while keeping track of the history of the last k choices. If $[1,n]$ is not contained inside this history, then the guessed sequence is not (n,k)-universal and M accepts.

The ACA (resp., fully-ACA) has $|A|^n$ possible configurations, and there are 2^n-1 (resp., n) possible updates, thus $(2^n-1)^k$ (resp., n^k) update sequences of length k. Hence, $\mathcal{C}$ can only perform $m=|A|^n\times(2^n-1)^k$ (resp., $m=|A|^n\times n^k$) computation steps before re-entering the same configuration z *after having performed the same updates in the last k steps*. We show that, if there is an updating sequence υ such that y is not reached within m steps in the instantiated ACA defined by $\mathcal{C}$ and υ, then there is a (n,k)-universal one such that y is never reached in the corresponding instantiated ACA. This means that the machine M can reject if y has not been reached after having simulated m steps of $\mathcal{C}$.

The pigeonhole principle assures that the above-mentioned updating sequence υ can be written as follows:

$$\upsilon=(\upsilon_1,\dots,\upsilon_i,\upsilon_{i+1},\dots,\upsilon_{i+k},\dots,\upsilon_j,\upsilon_{j+1},\dots,\upsilon_{j+k},\dots,\upsilon_{m+1},\dots)$$

for some distinct $i,j\in[1,m]$, with $(\upsilon_{i+1},\dots,\upsilon_{i+k})=(\upsilon_{j+1},\dots,\upsilon_{j+k})$ and $\Lambda_\upsilon^{i+k}(x)=\Lambda_\upsilon^{j+k}(x)=z$. Then, the updating sequence

$$\upsilon'=(\upsilon_1,\dots,\upsilon_i,\overline{\upsilon_{i+1},\dots,\upsilon_{i+k},\upsilon_{i+k+1},\dots,\upsilon_j}),$$

where the overlined part is periodically repeated in the sequence, ends up in the cycle

$$(z=\Lambda_\upsilon^{i+k}(x),\dots,\Lambda_\upsilon^{j+1}(x)=\Lambda_\upsilon^{i+1}(x),\dots,\Lambda_\upsilon^{j+k}(x)=\Lambda_\upsilon^{i+k}(x)=z),$$

never reaching y. Since M has not accepted while performing the computation corresponding to the updating subsequence $(\upsilon_1,\dots,\upsilon_j)$, such sequence has updated every position in $[1,n]$ at least once for each of its subsequences of length k. Therefore, υ' is (n,k)-universal since it consists of repeated subsequences of $(\upsilon_1,\dots,\upsilon_j)$. M can then reject if y has not been reached after m simulated steps; the machine can count from 0 to m in polynomial space

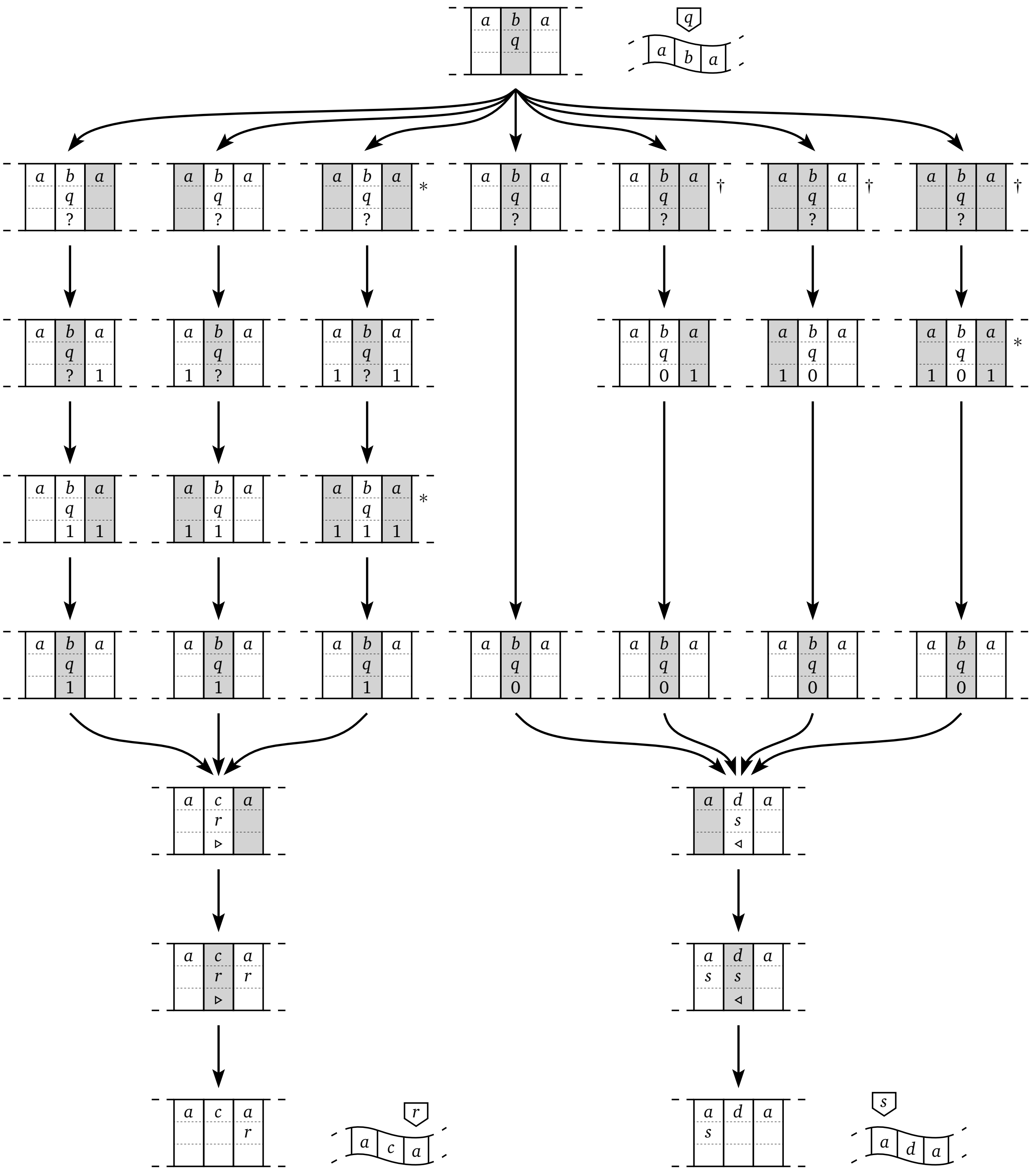


Figure 1: Simulation of one non-deterministic step of a NDTM in which $\delta(q, b) = \{(r, c, \triangleright), (s, d, \triangleleft)\}$.

as m consists of $O(n \log |A| + nk)$ (resp., $O(n \log |A| + k \log n)$) bits. If, on the other hand, M reaches y within m simulated steps, then it accepts.

Hence the problem is in **coNPSPACE** and thus in **PSPACE**; we now need to prove its hardness. Since *deterministic* TM working in polynomial space are a special case of NDTM with the same restriction, we can simulate them with ACA (resp., fully ACA) as described above. However, deterministic TM only have a single computation. By Lemma 4, *all* (n,k)-universal sequences will simulate that computation of the TM. Therefore, asking whether one (n,k)-universal sequence leads to the accepting configuration of a deterministic TM is equivalent to asking whether all such sequences lead to it (the only difference being the number of steps required to do it). This proves that the the same **PSPACE**-hardness construction presented in the proof of Theorem 5 can be used for the $\forall$-updates reachability problem. $\square$

When considering the reachability problems with the addition of the time constraint, we can prove that the complexity grows from **P**-completeness [20] to **NP**-completeness for the existential version and **coNP**-completeness for the universal one. The difference is the same as the one that appears when passing from simulating TM working in *space* n to TM working in *time* n [18].

Corollary 7. *The $\exists$-update reachability in polynomial time problem for finite ACA (or fully-ACA) is* **NP**-*complete. Furthermore, the $\forall$-updates reachability in polynomial time problem for finite ACA (or fully-ACA) is* **coNP**-*complete.*

Proof. The t computation steps of the ACA (or fully-ACA) $\mathcal{C}$ can be simulated in polynomial time by a NDTM M guessing a prefix of length t of an updating sequence υ. Only in the case of the universal version of the problem M machine accepts if the guess turns out not to be (n,k)-universal (since non-(n,k)-universal sequences do not influence the answer). Then, M actually performs the simulation of $\mathcal{C}$ according to υ for t steps starting from x, and accepts if and only if y is reached within that time limit (x and y are the configurations from the statement of the problem). Hence, M accepts at least once for the existential version of the problem, (resp., always for the universal one) iff for some updating sequence (resp., for all (n,k)-universal updating sequences) it holds that y is reached from x within t steps. This shows that the existential version of the problem is in **NP** while the universal one is in **coNP**.

The hardness of the problem in the existential (resp., universal) form can be shown by reduction from the problem of deciding whether a nondeterministic (resp., co-nondeterministic) Turing machine N accepts its input within u steps (in unary notation). We can build in polynomial time an ACA (or fully-ACA) simulating N as described above, and set $t = 9u$ for the existential form (simulating a step of the TM requires at most 9 steps in the most efficient updating sequence) and $t = 9ku$ for the universal form (simulating a step of the TM requires at most $9k$ steps in the least efficient (n,k)-universal updating sequence). This proves the **NP**-hardness of the existential form of the problem, and the **coNP**-hardness of the universal one. $\square$

4. Preimage Existence Problems

In this section we study the complexity of finding a preimage for ACA and fully-ACA. While in the second case the problems are all of low complexity, in the first one the complexity depends on the dimension, as in the synchronous case, with the problems in dimension 1 being **NL**-complete, and either **NP**-complete or $\mathbf{coNP^{NP}}$-complete in dimension 2 or higher.

Given a d-dimensional ACA (resp., fully-ACA) $\mathcal{C}$ of n cells and a configuration c, for a total input length of $\Theta\big(\log |A| + |A|^{(2r+1)^d} \log |A| + |A|^n\big)$ bits, we consider the following two problems:

- The $\exists$-*update preimage existence problem for finite ACA (resp., fully-ACA) problem* consists of deciding if there exists a non-empty set of positions (resp., a position) υ_1 such that c admits a preimage in any instantiated ACA (resp., fully-ACA) problem with updating sequence having υ_1 as first component.

- $\forall$-*update preimage existence problem for finite ACA (resp., fully-ACA) problem* consists of deciding if, for every set of positions (resp., position) υ_1, c admits a preimage in any instantiated ACA (resp., fully-ACA) problem with updating sequence having υ_1 as first component.

Notice that we have excluded the empty set in the first problem since every configuration is its preimage when no cell is updated.

Theorem 8. *The $\exists$-update (resp., $\forall$-updates) preimage existence problem for finite fully-ACA is in* **L***.*

Proof. Let $\mathcal{C}$ be the encoding of a fully-ACA (in any dimension) and c be a configuration. Let us consider the following algorithm which first, for each position i in c, examines the neighborhood of i, and replaces c_i by each state of A, one at a time; then, it checks if the image of the neighborhood obtained in this way coincides with c_i. Finally, it accepts if there exists one such position i (resp., all positions i have this property). This algorithm can be executed in logarithmic space, by storing the replacement of c_i on the working tape rather than changing the input. □

The following two theorems show that the preimage existence problem in both its existential and universal version is **NL**-complete, as in the synchronous case. The proof is, in fact, inspired by the one presented in [20].

Theorem 9. *The $\exists$-update preimage existence problem for one-dimensional finite ACA is* **NL***-complete.*

Proof. To show the inclusion in **NL** consider the following non-deterministic TM working in logarithmic space and having the description of an ACA $\mathcal{C} = (A, r, \lambda)$ and a target configuration $c = c_1 \cdots c_n$ on its input tape. First of all, the TM maintains a sliding window of size $2r + 1$ of symbols of A, a bit b (initially 0) to check if at the number of updated cells is non-null, and an index i ranging in $[1, n]$. At every time step the machine non-deterministically decides whether to apply the local function to the current $2r + 1$ symbols (and set b to 1) or to simply copy the central symbol (the i-th cell is not updated); in both cases, the machine checks if the obtained symbol is the i-th one in the target configuration. If so, the algorithm continues by increasing i by one, removing the leftmost symbol in the sliding window while moving all the others to the left and guessing another symbol for the rightmost position. Otherwise the machine rejects. When all the positions have been checked, i.e., there exists a preimage for the target configuration, the algorithm needs only to check if b is 0 (i.e., the preimage is the trivial one since no cell has been updated) or not. In the first case, the computation rejects and accepts otherwise. The periodic boundary conditions are enforced by storing the first r symbols guessed and using them when checking the last r positions. The algorithm runs in $\Theta((2r + 1)\log|A|)$ space. Since the encoding of the input requires $m = \Theta\big(\log|A| + |A|^{2r+1}\log|A| + |A|^n\big)$ bits (the description of the ACA plus the target configuration) we have $(2r + 1)\log|A| = O\big(\log\frac{m-|A|^n-\log|A|}{\log|A|}\big)$, that is $(2r + 1)\log|A| = O(\log m)$, thus showing that logarithmic space suffices. Notice that the previous algorithm can also be expressed in term of finding paths on a De Bruijn graph having multiple labels on each edge.

Let $G = (V, E)$ be a directed graph and $s, t \in V$. We reduce graph reachability in G to the problem of the theorem's statement. Let $\mathcal{C} = (A, 1, \lambda)$ be a ACA of length $n = |V| + 2$, where $A = E \cup \{(v, v) \mid v \in V\} \cup \{s, t, \heartsuit, \spadesuit\}$, and we ask whether the configuration $c = s\heartsuit^{n-2}t$ has a preimage. The local function λ is defined in the following way:

$$\lambda(x,y,z) = \begin{cases} s & \text{if } y = s \text{ and } z = (s,v) \text{ for some } v \in V. \\ t & \text{if } y = t,\ z = s, \text{ and } x = (u,t) \text{ for some } u \in V. \\ \heartsuit & \text{if } y = (u,v) \text{ and } z = (v,w) \text{ for some } u,v,w \in V \\ & \text{or if } y = (u,t) \text{ and } z = t \text{ for some } u \in V. \\ \spadesuit & \text{otherwise.} \end{cases}$$

We now show that c has a preimage iff there exists a path from s to t in G. The function λ forces each preimage to have the form $s(s, v_1)(v_1, v_2)\cdots(v_{n-2}, t)t$ where each state of the form (u, v) is either in V or it is a self-loop, where the self-loops exist only to assure that each path has length exactly $|V|$. Notice that the only updates that lead from such preimages to the target configuration c are those containing all positions in $[2, n-1]$ and, optionally, 1 or n for a total of four possible updates. Hence, it is immediate that an update leading from such preimages to c exists iff the path that it defines (possibly with self-loops removed) exists in G. Starting from G the ACA $\mathcal{C}$ can be built in deterministic logarithmic space. This can be done by looking at a constant number of edges at a time. Since determining the existence of a path in a directed graph is a **NL**-complete problem, the hardness of the theorem's problem follows. □

Theorem 10. *The* $\forall$*-updates preimage existence problem for one-dimensional finite ACA is* **NL***-complete.*

Proof. We first show that the algorithm in the proof of Theorem 9 can be adapted to prove the inclusion in $\mathbf{coNL}^{\mathbf{NL}}$. Since the logarithmic space hierarchy collapses to **NL** [14], the given problem is actually contained in **NL**. First of all, we redefine the algorithm to have an additional input of length n in binary representing the non-deterministic choices performed when deciding whether to update or not a cell. We will ask a query to an oracle for the problem solved by this modified algorithm. The non-deterministic **coNL** machine will simply copy its input (i.e., the description of the ACA $\mathcal{C}$ and the target configuration c) to the *query tape* and then, also on the query tape, it will write n non-deterministically chosen bits. The answer of the machine will be the same as the oracle unless the n guessed bits were all 0; in that case the machine will accept even if the oracle rejects. The machine has only accepting computations iff the target configuration c has a preimage for each possible choice of cells to update.

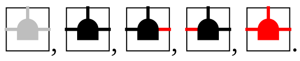
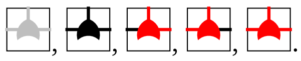
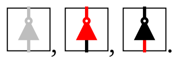
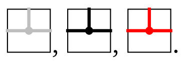
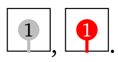
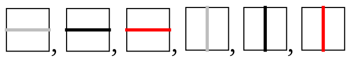
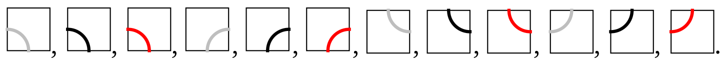
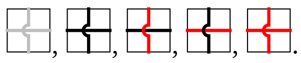
To show the **NL**-hardness of the problem, we modify the ACA used for the reduction in the proof of Theorem 9 in the following way: the alphabet and radius are the same, as it is the target configuration $c = s\heartsuit^{n-2}t$. The local rule λ is instead defined as follows:

$$\lambda(y,z) = \begin{cases} s & \text{if } y = s \text{ and } z = (s,v) \text{ for some } v \in V \text{ or } z = \heartsuit. \\ t & \text{if } y = t \text{ and } z = s. \\ \heartsuit & \text{if } y = (u,v) \text{ and } z = (v,w) \text{ for some } u,v,w \in V \text{ or} \\ & y = (u,t) \text{ and } z = t \text{ for some } u \in V \text{ or } z = \heartsuit. \\ \spadesuit & \text{otherwise.} \end{cases}$$

The only difference from the original local rule is that the presence of $\heartsuit$ in the right cell generates $\heartsuit$ instead of $\spadesuit$. As in the proof of Theorem 9, there exists a preimage c' for c when all cells update iff there exists a path from s to t in G. In that case c also has a preimage for each possible update. In particular, if a position $i \in [2, n-1]$ does not update we can simply take c' and replace the content of the i-th cell with $\heartsuit$. The local rule assure that such a state has c as image. Therefore, the problem is **NL**-hard. $\square$

As in synchronous CA, increasing the number of dimensions also increases the complexity of the problems. In particular, we show that it is possible to evaluate arbitrary circuits and, as a direct consequence, formulae. Furthermore, in one case we can also add universally quantified variables to the circuit or formula.

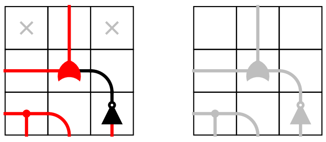
To encode a circuit in a two-dimensional CA, we will use an alphabet describing wires, logical gates, empty space and existentially and universally quantified variables. We will use the convention that black represents a *false* or 0 value, red a *true* or 1 value and light gray that the corresponding wire/port has no current value assigned. The entire alphabet is the following one:

- The *and* gate: 
- The *or* gate: 
- The *not* gate: 
- Duplication of the signal: 
- Circuit output set to true: 
- Horizontal and vertical wires: 
- Curved wires: 
- “Jumping wires”: 

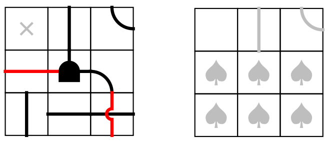
- Three existential and two universal quantifiers: ∃, ∃, ∃, ∀, ∀.
- White spaces: □, ☒.
- Error state: ♠.

Without going into details, it is possible to check that a circuit encoded in a CA is well-formed in a local way, in particular using a local rule having radius one. This means that it is necessary to verify that a black (resp., red) wire is always connected to a black (resp., red) wire or to a black (resp., red) input or output of a gate. If this is the case, the local rule "paint" all the wires and gates in gray.

Example 11. A correct circuit respects all the properties stated above and also all spaces unoccupied by wires and gates are crossed; this is a necessary technical restriction that will be used in one of the proofs. When a circuit is well-formed, all its wires and gates are colored in gray and the crosses are removed from the empty spaces by the local rule:



If the local rule detects an inconsistency of the circuit then an error state is produced as in the case below:



In this example there are two inconsistencies: a wire that is not connected to another wire or gate and a wire that, even if correctly connected, changes its value. The error state is produced in the cells that are able to detect the inconsistencies using radius one. Therefore, any configuration containing the error symbol cannot have a preimage that is a well-formed circuit.

We are now able to prove that deciding if a configuration admits a preimage for some non-empty set of positions to update is **NP**-complete, as the corresponding problem in synchronous CA [20].

Theorem 12. *The ∃-update preimage existence problem for two-dimensional finite ACA is* **NP***-complete.*

Proof. The problem is in **NP**, since, given a configuration c, it is possible to guess at the same time the preimage and the cells to update. Moreover, the verification that c has the guessed preimage can be performed in deterministic polynomial time.

We reduce the 3-SAT problem [13] to this problem. Given a Boolean formula $\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_m$ in CNF using the variables $V = \{x_1, \ldots, x_n\}$, it is possible to build a circuit representing φ using a two-dimensional CA encoding the circuit representing the formula having width $4m + 1$ and height $n + 4 + \lceil \log_2 m \rceil$ (see Figure 2). The circuit is built as follows: in the leftmost column for each variable there is an existentially quantified state, indicating that the variable appears under an existential quantifier (implicit in the usual formulation of 3-SAT). Notice that in this construction no universal quantifier is used. From each existential state a wire is connected to the gates representing the disjunctions in the clauses (possibly after being negated); the wire exiting from the disjunction are, in turn, connected to a binary tree of $m - 1$ gates having logarithmic depth and outputting one. This is a circuit representation of the formula φ, which requires only a polynomial amout of space with respect to the size of the formula. Notice that we do not need to name each variable since each one is defined by its connections to the different gates (i.e., the number of states of the CA is constant).

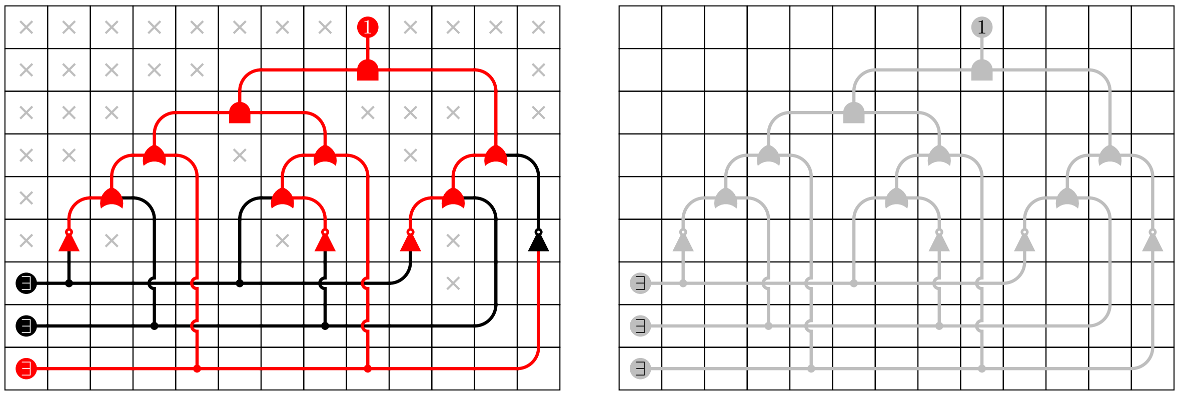


Figure 2: The formula $\varphi = (\neg x \lor y \lor z) \land (x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$ represented as a circuit encoded in a two-dimensional CA. On the left a satisfying assignment of φ and on the right its image when all the cells are updated.

Now consider a configuration c representing φ in which each gate and wire is colored in gray and each empty space is not crossed. We define the local function in such a way that it checks that the circuit is consistent, that the empty spaces are crossed and that there are no non-crossed empty spaces and no gray wires or gates. In that case an error state is produced. This forces all the cells to activate in order to obtain c as an image and, since the circuit must be consistent and with output one, there exists a preimage of c if and only if φ is satisfied. In fact, if one cell is not updated it must already contain the same value as in c, which is either a gray gate or wire or a non-crossed empty space; since at least one cell updates, an error would necessarily be produced. Since encoding φ as a circuit encoded in a CA can be performed in polynomial time, the **NP**-hardness of the problem follows. □

Theorem 13. *The ∀-updates preimage existence problem for two-dimensional finite ACA is* $\mathbf{coNP}^{\mathbf{NP}}$*-complete.*

Proof. First of all, we show that the problem is in $\mathbf{coNP}^{\mathbf{NP}}$. We use a polynomial time NDTM to guess a non-empty subset of cells to be updated and then we use a query to the **NP** oracle to verify the existence of a preimage with the chosen set of cells to be updated; the machine accepts if the answer is positive. If all computations of the NDTM accept, then for each choice of cells to update there exists a preimage, showing that the considered problem is contained in $\mathbf{coNP}^{\mathbf{NP}}$.

As to hardness, we reduce ∀∃3-SAT [19], which is complete for $\mathbf{coNP}^{\mathbf{NP}}$, to it. Let $\varphi = \forall X \exists Y \varphi(X, Y)$ be a quantified formula in 3-CNF over the variables $V = X \cup Y = \{x_1, \dots, x_n\}$. As in the proof of Theorem 12, we can build a circuit representation of φ as a CA, with the difference that the universally quantified variables are represented by a universally quantified state (see Figure 3). Out target configuration c is a circuit representing φ with each wire and gate colored in gray and in which each empty space is not crossed.

The local function checks the consistency of the circuit, but with the following modifications:

- The fact that a cell has in the neighborhood (excluding itself) a gray wire or gate (except the universally quantified ones) or a non-crossed empty space is not considered an error and the other consistency checks apply (i.e., that the wires are properly connected and the output of the gates is correct).
- Any wire coming out from a black universally quantified gate must be black (or gray), while any wire coming out from an universally quantified gray gate must be red (or gray).
- Gray gates and wire and empty non-crossed spaces if activated produce an error state.

The previous conditions assure that when all the cells update, except possibly the universally quantified gates, the value of the universally quantified variables only depend on whether the cells containing the universally quantified state update or not. Let P be the set of positions in which the universally quantified gates are located in the configuration. Each subset Q of P thus corresponds to an assignment of the variables in X. When all cells except those in $P - Q$ update *and* the next configuration is c, all the states of the wires starting from a universal gate are forced according to the specific assignment encoded by Q. The hypothesis that the image is c forces the state of the wires originating at an existential gate to encode a satisfying assignment to the variables in Y. Therefore, if there exists a preimage in which all cells update, except the ones in $P - Q$ for each possible subset Q of P, then the formula φ is valid. When not all cells outside P update we are actually considering a subset of one of the updates already examined. We can then build a preimage for that sub-update by simply taking the preimage c corresponding to the super-update and "greying-out" the gates and wires in the positions outside P that do not update. This shows that if there exists a preimage for each update then the formula is valid. On the other hand, if the formula φ is valid, then all the preimages corresponding to updates where all the cells excepts a subset of P update necessarily exist, since they are simply a circuit representation of a satisfying assignment. As before, all the other preimages can be obtained by "greying-out" part of these preimages.

Hence, there exists a preimage for c for each possible update if and only if φ is valid. Since encoding φ as a circuit encoded in a CA can be performed in polynomial time, the $\mathbf{coNP^{NP}}$-hardness of the problem follows. $\square$

It is possible to see an example of configuration and of the preimages satisfying it in Figure 3.

5. Conclusions

In this paper, we have introduced a way of simulating non-deterministic Turing Machines working in space n by means of ACA and fully-ACA of the same size. The resulting simulation can be performed in a time that is linear with respect to the time needed by the simulated Turing machine and the non-determinism is introduced by using different updating sequences. This construction has been used to show that the reachability problem, that can be formulated in two different ways for ACA and fully-ACA, is **PSPACE**-complete, as for the reachability problem in classical CA. When a restriction on the time needed to reach a configuration starting from another one is added, the asynchronous model behaves differently from the synchronous one, also in terms of complexity. The restricted reachability problem is **P**-complete for classical CA, but it is **NP**-complete or **coNP**-complete for both ACA and fully-ACA, in the existential or universal version, respectively. This shows that an asynchronous update can change the complexity of some decision problems for CA by adding non-determinism.

We have also proved that deciding whether a preimage of a given configuration exists is easy for fully-ACA (i.e., contained in **L**), while it depends on the dimension for ACA. For one dimensional ACA the preimage existence problem in both possible formulations is **NL**-complete, the same as in classical CA. However, for dimensions greater than one the complexity increases, depending on the formulation of the problem, and we either have a **NP**-complete problem (as in classical CA) or a $\mathbf{coNP^{NP}}$-complete problem, an increase with respect to the classical case. We have also illustrated a general and easy way to embed formulae represented as circuits in two-dimensional (synchronous or not) CA and how to use asynchrony to "simulate" universally quantified variables. This is particularly interesting since, unless $\mathbf{coNP^{NP}}$ collapses to **NP**, this is an action that cannot be performed with classical CA.

In the future we plan to study the complexity of others decision problems in the ACA and fully-ACA setting. The problems studied in [20] are, for example, an interesting starting point. A general trend that has emerged from the results of this paper is that ACA and, to a lesser extent, fully-ACA seems to behave – at least with respect to the computational complexity point of view – as classical CA enhanced with non-determinism. It would be interesting to determine if this is also true for other kinds of problem and to derive a general criteria to decide in which problems this behavior exists.

Acknowledgments

This work has been partially supported by the Italian MIUR PRIN 2010–2011 grant "Automata and Formal Languages: Mathematical and Applicative Aspects" H41J12000190001.

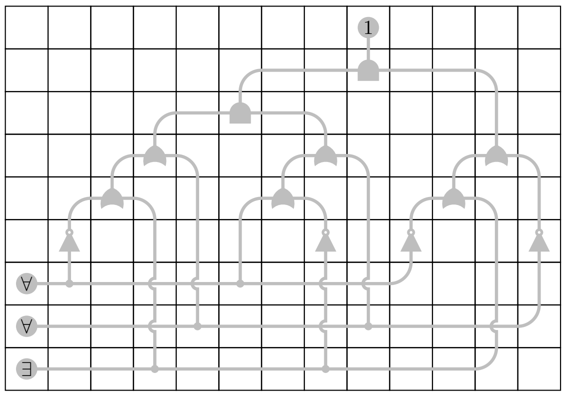
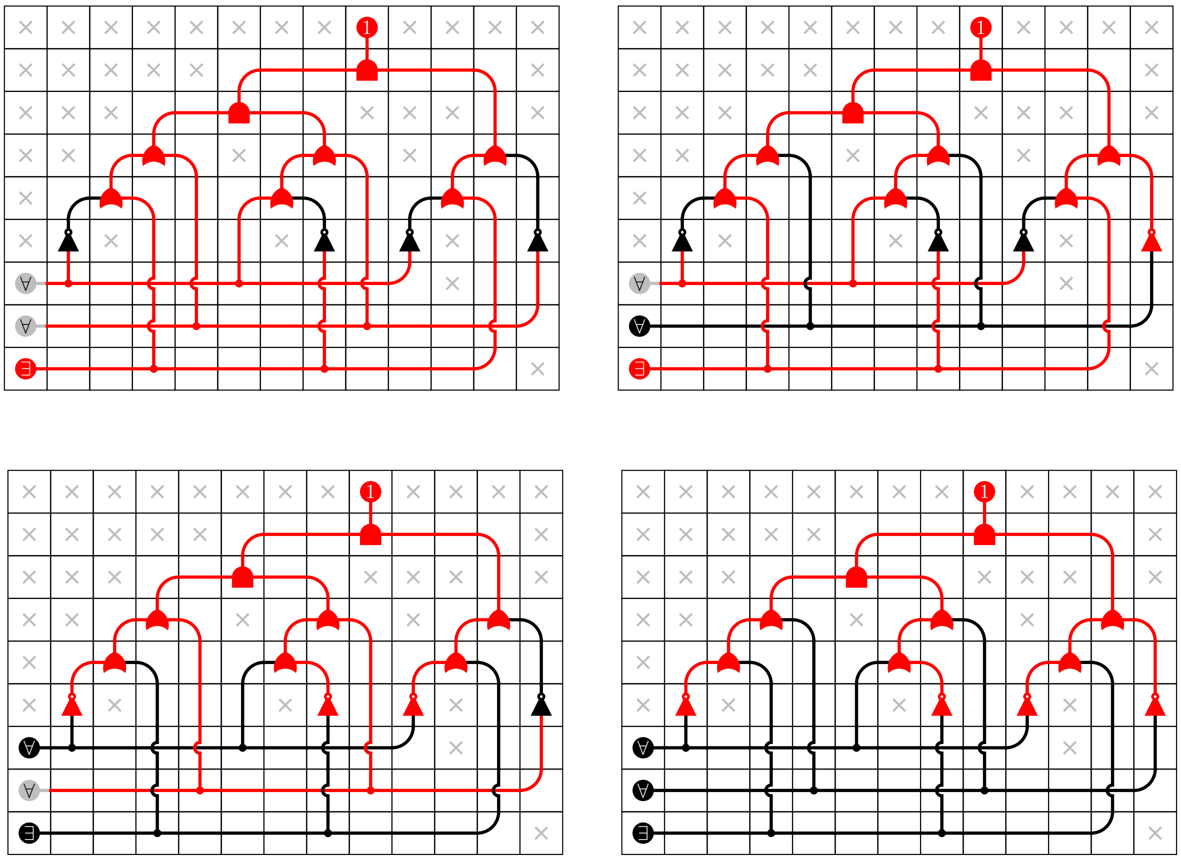



Figure 3: On the top, the target configuration c representing $\varphi = \forall x \forall y \exists z\big((\neg x \vee z \vee y) \wedge (x \vee \neg z \vee y) \wedge (\neg x \vee z \vee \neg y)\big)$. The other four configurations represent one preimage of c for each possible assignment of the universally quantified variables. The existence of such preimages shows that φ is valid.

Bibliography

[1] L. Acerbi, A. Dennunzio, E. Formenti, Conservation of some dynamical properties for operations on cellular automata, Theoretical Computer Science 410 (2009) 3685–3693. URL: `http://dx.doi.org/10.1016/j.tcs.2009.05.004`.

[2] L. Acerbi, A. Dennunzio, E. Formenti, Surjective multidimensional cellular automata are non-wandering: A combinatorial proof, Information Processing Letters 113 (2013) 156–159. URL: `http://dx.doi.org/10.1016/j.ipl.2012.12.009`.

[3] K. Culik, II, S. Yu, Undecidability of CA classification schemes, Complex Systems 2 (1988) 177–190. URL: `http://www.complex-systems.com/abstracts/v02_i02_a02.html`.

[4] A. Dennunzio, E. Formenti, L. Manzoni, Computing issues of asynchronous CA, Fundamenta Informaticae 120 (2012) 165–180. URL: `http://dx.doi.org/10.3233/FI-2012-755`.

[5] A. Dennunzio, E. Formenti, L. Manzoni, Limit properties of doubly quiescent m-asynchronous elementary cellular automata, Journal of Cellular Automata 9 (2014) 341–355. URL: `http://www.oldcitypublishing.com/journals/jca-home/jca-issue-contents/jca-volume-9-number-5-6-2014/jca-9-5-6-p-341-355/`.

[6] A. Dennunzio, E. Formenti, L. Manzoni, G. Mauri, m-Asynchronous cellular automata: From fairness to quasi-fairness, Natural Computing 12 (2013) 561–572. URL: `http://dx.doi.org/10.1007/s11047-013-9386-5`.

[7] A. Dennunzio, E. Formenti, J. Provillard, Non-uniform cellular automata: Classes, dynamics, and decidability, Information and Computation 215 (2012) 32–46. URL: `http://dx.doi.org/10.1016/j.ic.2012.02.008`.

[8] A. Dennunzio, E. Formenti, J. Provillard, Local rule distributions, language complexity and non-uniform cellular automata, Theoretical Computer Science 504 (2013) 38–51. URL: `http://dx.doi.org/10.1016/j.tcs.2012.05.013`.

[9] A. Dennunzio, E. Formenti, M. Weiss, Multidimensional cellular automata: closing property, quasi-expansivity, and (un)decidability issues, Theoretical Computer Science 516 (2014) 40–59. URL: `http://dx.doi.org/10.1016/j.tcs.2013.11.005`.

[10] B. Durand, E. Formenti, G. Varouchas, On undecidability of equicontinuity classification for cellular automata, in: M. Morvan, E. Rémila (Eds.), Discrete Models for Complex Systems, DMCS'03, volume AB of *DMTCS Proceedings*, 2003, pp. 117–128. URL: `http://www.dmtcs.org/dmtcs-ojs/index.php/proceedings/article/viewArticle/dmAB0110/1261`.

[11] N. Fatès, M. Morvan, N. Schabanel, E. Thierry, Fully asynchronous behaviour of double-quiescent elementary cellular automata, Theoretical Computer Science 362 (2006) 1–16. URL: `http://dx.doi.org/10.1016/j.tcs.2006.05.036`.

[12] N. Fatѐs, D. Regnault, N. Schabanel, E. Thierry, Asynchronous behavior of double-quiescent elementary cellular automata, in: J.R. Correa, A. Hevia, M. Kiwi (Eds.), LATIN 2006: Theoretical Informatics, volume 3887 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 455–466. URL: `http://dx.doi.org/10.1007/11682462_43`.

[13] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., 1979.

[14] N. Immerman, Nondeterministic space is closed under complementation, SIAM Journal on Computing 17 (1988) 935–938. URL: `http://dx.doi.org/10.1137/0217058`.

[15] J. Kari, Theory of cellular automata: A survey, Theoretical Computer Science 334 (2005) 3–33. URL: `http://dx.doi.org/10.1016/j.tcs.2004.11.021`.

[16] J. Kari, N. Ollinger, Periodicity and immortality in reversible computing, in: E. Ochmański, J. Tyszkiewicz (Eds.), Mathematical Foundations of Computer Science 2008, volume 5162 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 419–430. URL: `http://dx.doi.org/10.1007/978-3-540-85238-4_34`.

[17] L. Manzoni, Asynchronous cellular automata and dynamical properties, Natural Computing 11 (2012) 269–276. URL: `http://dx.doi.org/10.1007/s11047-012-9308-y`.

[18] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, 1993.

[19] L.J. Stockmeyer, The polynomial-time hierarchy, Theoretical Computer Science 3 (1976) 1–22. URL: `http://dx.doi.org/10.1016/0304-3975(76)90061-X`.

[20] K. Sutner, On the computational complexity of finite cellular automata, Journal of Computer and System Sciences 50 (1995) 87–97. URL: `http://dx.doi.org/10.1006/jcss.1995.1009`.

[21] T. Worsch, A note on (intrinsically?) universal asynchronous cellular automata, in: Proceedings of Automata 2010, 14-16 June 2010, Nancy (France), 2010, pp. 339–350. URL: `https://hal.inria.fr/inria-00549645/en`.