Complexity-Theoretic Aspects of Expanding Cellular Automata

Augusto Modanese

Complexity-Theoretic Aspects of Expanding Cellular Automata ⋆

Augusto Modanese(✉)

Institute of Theoretical Informatics (ITI), Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany
`modanese@kit.edu`

Abstract The expanding cellular automata (XCA) variant of cellular automata is investigated and characterized from a complexity-theoretical standpoint. The respective polynomial-time complexity class is shown to coincide with $\leq_{tt}^{p}(\mathbf{NP})$, that is, the class of decision problems polynomial-time truth-table reducible to problems in $\mathbf{NP}$. Corollaries on select XCA variants are proven: XCAs with multiple accept and reject states are shown to be polynomial-time equivalent to the original XCA model. Meanwhile, XCAs with diverse acceptance behavior are classified in terms of $\leq_{tt}^{p}(\mathbf{NP})$ and the Turing machine polynomial-time class $\mathbf{P}$.

1 Introduction

Traditionally, cellular automata (CAs) are defined as a rigid and immutable lattice of cells; their behavior is dictated exclusively by a local transition function operating on homogeneous local configurations. This can be generalized, for instance, by mutable neighborhoods [17] or by endowing CAs with the ability to *shrink*, that is, delete their cells [18]. When shrinking, the automaton's structure and dimension are preserved by "gluing" the severed parts and reconnecting their delimiting cells as neighbors. When employed as language recognizers, shrinking CAs (SCAs) can be more efficient than standard CAs [18, 10].

Other variants of CAs with dynamically reconfigurable neighborhoods have emerged throughout the years. In the case of two-dimensional CAs, there is the structurally dynamical CA (SDCA) [7], in which the connections between neighbors are created and dropped depending on the local configuration. In the one-dimensional case, further variants in this sense are considered in the work of Dubacq [5], where one finds, in particular, CAs whose neighborhoods vary over time. Dubacq also proposes the dynamically reconfigurable CA (DRCA), a CA whose cells are able to exchange their neighbors for neighbors of their neighbors. Dantchev [4] later points out a drawback in the definition of DRCAs and proposes an alternative dubbed the dynamic neighborhood CA (DNCA).

By relaxing the arrangement of cells as a lattice, CAs may be generalized to graphs. Graph automata [21] are related to CAs in that each vertex in the graph

⋆ Parts of this paper have been submitted [13] in partial fulfillment of the requirements for the degree of Master of Science at the Karlsruhe Institute of Technology (KIT).

can be identified as a cell; thus, graphs whose vertices have finite degrees provide a natural generalization of CAs. In [21], the authors also define a rule based on topological refinements of graphs, which may be used as a model for biological cell division. An additional example of cell division in this sense is found in the "inflating grid" described in [1].

Modeling cell division and growth, in fact, was one of the driving motivations towards the investigation of the *expanding CA* (XCA) in [14]. An XCA is, in a way, the opposite of an SCA; instead of cells vanishing, new cells can emerge between existing ones. This operation is topologically similar to the cell division of graph automata; as in the SCA model, however, it maintains the overall arrangement and connectivity of the automaton's cells as similar as possible to that of standard CAs (i.e., a bi-infinite, one-dimensional array of cells).

We mention a few aspects in which XCAs differ from the aforementioned variants. Contrary to SDCAs [7] or CAs with dynamic neighborhoods such as DRCAs [5] and DNCAs [4], XCAs enable the creation of new cells, not simply new links between existing ones. In addition, the XCA model does not focus as much on the reconfiguration of cells; in it, the neighborhoods are homogeneous and predominantly immutable. Furthermore, in contrast to the far more general graph automata [21], XCAs are still one-dimensional CAs; this ensures basic CA techniques (e.g., synchronization) function the same as they do in standard CAs.

Finally, shrinking and expanding are not mutually exclusive. Combining them yields the shrinking and expanding CA (SXCA). The polynomial-time class of SXCA language deciders was shown in [14, 15] to coincide with **PSPACE**.

In [14], the polynomial-time class **XCAP** of XCA language deciders is shown to contain both **NP** and **coNP** while being contained in **PSPACE**. A precise characterization of **XCAP**, however, remained outstanding. Such was the topic of the author's master's thesis [13], the results of which are summarized in this paper. The main result is **XCAP** being equal to the class of decision problems which are polynomial-time truth-table reducible to **NP**, denoted $\leq_{tt}^{p}(\mathbf{NP})$.

The rest of this paper is organized as follows: Section 2 covers the fundamental definitions and results needed for the subsequent discussions. Following that, Section 3 recalls the main result of [14] concerning **XCAP** and presents the aforementioned characterization of **XCAP**. Section 4 covers some immediate corollaries, in particular by considering an XCA variant with multiple accept and reject states as well as two other variants with diverse acceptance conditions. Finally, Section 5 concludes.

2 Basic Definitions

This section recalls basic concepts and results needed for the proofs and discussions in the later sections and is broken down in two parts. The first is concerned with basic topics regarding formal languages, Turing machines, and Boolean formulas. The second part covers the definition of expanding CAs.

2.1 Formal Languages and Turing Machines

It is assumed the reader is familiar with the concepts of $\omega\omega$-words and their homomorphisms as well as deterministic and non-deterministic Turing machines (TMs and NTMs, respectively) and the fundamental classes **P**, **NP**, **coNP**, and **PSPACE**. In this paper, it is assumed all words have length at least one. The notion of a *complete* language is employed strictly in the sense of polynomial-time many-one (i.e., Karp) reductions by Turing machines.

Boolean Formulas Let V be a language of *variables* over an alphabet Σ which, without loss of generality, is disjoint from $\{T, F, \neg, \wedge, \vee, (,)\}$. BOOL_V denotes the language of Boolean formulas over the variables of V. An *interpretation* of V is a map $I\colon V \to \{T, F\}$. Each interpretation I gives rise to an *evaluation* $E_I\colon \mathrm{BOOL}_V \to \{T, F\}$ which, given a formula $f \in \mathrm{BOOL}_V$, substitutes each variable $x \in V$ with the truth value $I(x)$ and reduces the resulting formula using standard propositional logic. A formula f is *satisfiable* if there is an interpretation I such that $E_I(f) = T$; f is a *tautology* if this holds for all interpretations.

In order to define the languages SAT of satisfiable formulas and TAUT of tautologies, a language V of variables must be agreed on. In this paper, variables are encoded as binary strings prefixed by a special symbol x, that is, $V = \{x\} \cdot \{0, 1\}^+$. The language SAT contains exactly the satisfiable formulas of BOOL_V. Similarly, TAUT contains exactly the tautologies of BOOL_V. The following is a well-known result concerning SAT and TAUT [3]:

Theorem 1. SAT *is* **NP***-complete, and* TAUT *is* **coNP***-complete.*

Truth-Table Reductions The theory of truth-table reductions was established in [12, 11]; later it was shown the class of decision problems polynomial-time truth-table (i.e., Boolean circuit) reducible to **NP**, denoted $\leq^p_{tt}(\mathbf{NP})$, is equivalent to that of those polynomial-time Boolean formula reducible to **NP** [22]. We refer to [2] for a series of alternative characterizations of $\leq^p_{tt}(\mathbf{NP})$. As a cursory remark, we state the inclusions $\mathbf{NP} \cup \mathbf{coNP} \subseteq \leq^p_{tt}(\mathbf{NP})$ and $\leq^p_{tt}(\mathbf{NP}) \subseteq \mathbf{PSPACE}$ hold.

For the results of this paper, a formal treatment of the class $\leq^p_{tt}(\mathbf{NP})$ is not necessary; it suffices to note $\leq^p_{tt}(\mathbf{NP})$ has complete languages. In particular, we are interested in Boolean formulas with **NP** and **coNP** predicates. To this end, we employ SAT and TAUT to define membership predicates of the form $f \sqsubseteq_L$, where f is a Boolean formula, $L \in \{\mathrm{SAT}, \mathrm{TAUT}\}$, and "$\sqsubseteq_L$" is a purely syntactic construct which stands for the statement "$f \in L$".

Definition 1 ($\mathbf{SAT}^{\wedge}$-$\mathbf{TAUT}^{\vee}$). *Let* $V = \{x\} \cdot \{0, 1\}^+$ *and* $V_L = \mathrm{BOOL}_V \cdot \{\sqsubseteq_L\}$ *for* $L \in \{\mathrm{SAT}, \mathrm{TAUT}\}$*. The language* $\mathrm{BOOL}^{\wedge\vee}_{\mathrm{SAT},\mathrm{TAUT}} \subseteq \mathrm{BOOL}_{V_{\mathrm{SAT}} \cup V_{\mathrm{TAUT}}}$ *is defined recursively as follows:*

1. $V_{\mathrm{SAT}}, V_{\mathrm{TAUT}} \subseteq \mathrm{BOOL}^{\wedge\vee}_{\mathrm{SAT},\mathrm{TAUT}}$
2. *If* $v \in V_{\mathrm{SAT}}$ *and* $f \in \mathrm{BOOL}^{\wedge\vee}_{\mathrm{SAT},\mathrm{TAUT}}$*, then* $\wedge(v, f) \in \mathrm{BOOL}^{\wedge\vee}_{\mathrm{SAT},\mathrm{TAUT}}$
3. *If* $v \in V_{\mathrm{TAUT}}$ *and* $f \in \mathrm{BOOL}^{\wedge\vee}_{\mathrm{SAT},\mathrm{TAUT}}$*, then* $\vee(v, f) \in \mathrm{BOOL}^{\wedge\vee}_{\mathrm{SAT},\mathrm{TAUT}}$

The language $\mathrm{SAT}^{\wedge}\text{-}\mathrm{TAUT}^{\vee} \subseteq \mathrm{BOOL}^{\wedge\vee}_{\mathrm{SAT},\mathrm{TAUT}}$ *contains all formulas which are true under the interpretation mapping* $f \in_L$ *to the truth value of the statement "*$f \in L$*".*

The following follows from the results of Buss and Hay [2]:

Theorem 2. $\mathrm{SAT}^{\wedge}\text{-}\mathrm{TAUT}^{\vee}$ *is* $\leq^p_{tt}(\mathbf{NP})$*-complete.*

2.2 Cellular Automata

In this paper, we are strictly interested in one-dimensional cellular automata (CAs) with the standard neighborhood and employed as language deciders. CA deciders possess a *quiescent state* q; cells which are not in this state are said to be *active* and may not become quiescent. The input for a CA decider is provided in its initial configuration surrounded by quiescent cells. As deciders, CAs are Turing complete, and, more importantly, CAs can simulate TMs in real-time [19]. Conversely, it is known a TM can simulate a CA with time complexity t in time at most t^2. A corollary is that the CA polynomial-time class equals $\mathbf{P}$.

Expanding Cellular Automata First considered in [14], the expanding CA (XCA) is similar to the shrinking CA (SCA) in that it is dynamically reconfigurable; instead of cells being deleted, however, in an XCA new cells emerge between existing ones. This does not alter the underlying topology, which remains one-dimensional and biinfinite.

For modeling purposes, the new cells are seen as *hidden* between the original (i.e., *visible*) ones, with one hidden cell placed between any two neighboring visible cells. These latter cells serve as the hidden cell's left and right neighbors and are referred to as its *parents*. In each CA step, a hidden cell observes the states of its parents and either assumes a non-hidden state, thus becoming visible, or remains hidden. In the former case, the cell assumes the position between its parents and becomes an ordinary cell (i.e., visible), and the parents are reconnected so as to adopt the new cell as a neighbor. Visible cells may not become hidden.

Definition 2 (XCA). *Let* $N = \{-1, 0, 1\}$ *be the standard neighborhood. An* expanding CA *(*XCA*) is a CA* A *with state set* Q *and local transition function* $\delta\colon Q^N \to Q$ *and which possesses a distinguished* hidden state $\odot \in Q$*. For any local configuration* $\ell\colon N \to Q$*,* $\delta(\ell) = \odot$ *is allowed only if* $\ell(0) = \odot$*.*

Let $c\colon \mathbb{Z} \to Q$ *be a global configuration and* $z \in \mathbb{Z}$*, and let* $h_c\colon \mathbb{Z} \to Q^N$ *be such that* $h_c(z)(-1) = c(z)$*,* $h_c(z)(0) = \odot$*, and* $h_c(z)(1) = c(z+1)$*. Define* $\alpha\colon Q^{\mathbb{Z}} \to Q^{\mathbb{Z}}$ *as follows, where* Δ *is the standard CA global transition function:*

$$\alpha(c)(z) = \begin{cases} \Delta(c)(\frac{z}{2}), & z \text{ even} \\ \delta(h_c(\frac{z-1}{2})), & \text{otherwise} \end{cases}$$

Finally, let Φ *be the* $\omega\omega$*-word homomorphism induced by the mapping* $Q \to Q^*$ *which maps any state to itself except for* $\odot$*, which is mapped to* ε *(i.e., the empty word). Then the global transition function of an XCA is* $\Delta^X = \Delta^X_\delta := \Phi \circ \alpha$*.*

Figure 1 illustrates an XCA A and its operation for input 001010 as an example. The local transition function δ of A is as follows:

$$\delta(q_{-1}, q_0, q_1) = \begin{cases} q_{-1} \oplus q_1, & q_{-1}, q_1 \in \{0, 1\} \\ q_0, & \text{otherwise} \end{cases}$$

(where $\oplus$ denotes the bitwise XOR operation, that is, addition modulo 2). In the initial configuration c, the hidden cells are all in the state $\odot$. Using h_c as the hidden cells' local configurations, α applies δ to each local configuration and promotes all hidden cells to ordinary (i.e., visible) ones. Finally, Φ eliminates hidden cells which conserved the state $\odot$ (as these are present only implicitly in the global configuration).

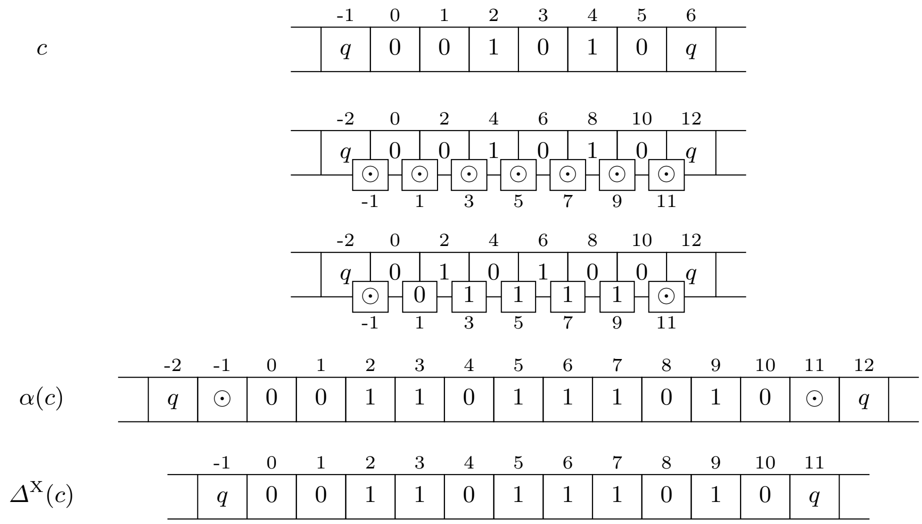

Figure 1. Illustration of a step of the XCA A. The number next to each cell indicates its index in the respective configuration. In this particular example, we use the convention that the $\omega\omega$-word homomorphism Φ contracts deleted symbols towards index zero. Adapted from [14].

The supply of hidden cells is never depleted; if a hidden cell becomes an ordinary cell, new ones appear between it and its neighbors. Thus, the number of active cells in an XCA may increase exponentially [14]:

Lemma 1. *Let A be an XCA. For an input of size n, A has at most $(n+3)2^t - 3$ active cells after $t \in \mathbb{N}_0$ steps. This upper bound is sharp.*

We have postponed defining the acceptance behavior of an XCA until now. Usually, a CA possesses a distinguished cell, often cell 0, which dictates the automaton's accept or reject response [9]. In the case of XCAs, however, under a reasonable complexity-theoretical assumption (i.e., $\mathbf{P} \neq \leq_{tt}^{p}(\mathbf{NP})$) such an acceptance behavior results in XCAs not making full use of the efficient cell growth indicated in Lemma 1 (see Section 4.3). This phenomenon does not occur if the acceptance behavior is defined based on *unanimity*, that is, in order for an XCA to accept (or reject), *all* its cells must assume the accept (or reject) state simultaneously. This acceptance condition is by no means novel [6, 20, 16, 8]. As an aside, note all (reasonable) CA time complexity classes (including, in particular, linear- and polynomial-time) remain invariant when using this acceptance condition instead of the standard one.

Definition 3 (Acceptance behavior, time complexity). *Each XCA has a (unique)* accept state *a and a (unique)* reject state *r. An XCA A halts if all active (and non-hidden) cells are either all in state a, in which case the XCA* accepts*, or they are all in state r, in which case it* rejects*; if neither is the case, the computation continues. $L(A)$ denotes the set of words accepted by A.*

The time complexity *of an XCA is the number of elapsed steps until it halts. An* XCA decider *is an XCA which halts on every input. A language L is in* **XCAP** *if there is an XCA decider A' with polynomial time complexity (in the length of its input) and such that $L = L(A')$.*

In summary, the decision result of an XCA decider is the one indicated by the first configuration in which its active cells are either all in the accept or all in the reject state. This agrees with our aforementioned notion of a *unanimous* decision.

3 Characterizing the XCA Polynomial-Time Class

In [14], the following first result regarding the class **XCAP** is proven:

Theorem 3. $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{XCAP}$.

We give a brief outline of the proof and refer the interested reader to [14] for the details. Since many-one reductions by TMs can be simulated by (X)CAs in real-time, it suffices to show **XCAP** contains **NP**- and **coNP**-complete problems. We construct XCAs for SAT and TAUT which run in polynomial time and apply Theorem 1. The two constructions are very similar; in fact, one obtains one from the other simply by swapping the accept and reject states. The following describes the XCA A for TAUT.

XCAs allow for efficient creation of new cells between existing ones. This may be used in order to efficiently create copies of the original formula and in each copy set a given variable to a possible truth value (i.e., "true" or "false"). A iterates over the input formula's variables and, at each iteration step, creates two copies of each formula: one in which the respective variable is set to "true" and one in which it is set to "false". The copies are synchronized with each other, and the process continues in parallel as separate computation branches. The synchronization ensures that, since all copies have equal length, all variables will be exhausted by each branch at the same time.

When this is the case, parallel evaluations of the resulting formulas are carried out, and their results combining using a subtlety of the accepting behavior of XCAs: Reject states are conserved while accept states yield to reject in the next step. If the result of an evaluation is "true", then all respective cells are synchronized and enter the accept state; otherwise, that is, if the result is "false", then all respective cells enter the reject state (after synchronization). As a result, if no reject states are present, then A immediately accepts; otherwise, in the next step, any existing accepting cells become rejecting cells, and A rejects.

Note the steps described above are all carried out in polynomial time since the whole process amounts to replacing variables with truth values and then evaluating the resulting formulas.

3.1 A First Characterization

This section covers the following (main) result of [13]:

Theorem 4. $\mathbf{XCAP} = \leq_{tt}^{p}(\mathbf{NP})$.

The equality in Theorem 4 is proven by considering the two inclusions.

Proposition 1. $\leq_{tt}^{p}(\mathbf{NP}) \subseteq \mathbf{XCAP}$.

Proof. We construct an XCA A which decides $\text{SAT}^{\wedge}\text{-TAUT}^{\vee}$ (see Definition 1 and Theorem 2) in polynomial time. The actual inclusion follows from the fact that CAs can simulate polynomial-time many-one reductions by TMs in real-time.

Given a problem instance f, A evaluates f recursively. Without loss of generality, $f = \wedge(f_1 \in_{\text{SAT}}, \vee(f_2 \in_{\text{TAUT}}, f'))$, where f' is some other problem instance; other instances of $\text{SAT}^{\wedge}\text{-TAUT}^{\vee}$ are obtained by replacing f_1, f_2, or f' with a trivial formula (e.g., a trivial tautology).

To evaluate $f_1 \in_{\text{SAT}}$, A emulates the behavior of the XCA for SAT (see Theorem 3); however, special care must be taken to ensure A does not halt prematurely. All computation branches retain a copy of f. Whenever a branch obtains a "true" result, the respective cells do not directly accept (as in the original construction); instead, they proceed with evaluating the formula's next connective. Conversely, if the result is false, the respective cells simply enter the reject state. The behavior for $f_2 \in_{\text{TAUT}}$ is analogous, with A emulating the XCA for TAUT instead (and with exchanged accept and reject states, accordingly). Additionally, the accept and reject states are such that cells transition between them back and forth[1], and we (arbitrarily) enforce accept states only exist in even- and reject states in odd-numbered steps[2].

If $f_1 \notin \text{SAT}$, all branches of A transition into the reject state, and A rejects. Otherwise, f_1 is satisfiable; thus, at least one branch obtains a "true" result, and A continues to evaluate f until the (aforementioned) base case is reached.

[1] That is, $\delta(\ell) = a$ for $\ell(0) = r$ and vice-versa, where δ and ℓ are as in Definition 2.

[2] This may be accomplished, for example, by using a bit counter in the cells' states, and having cells wait for a step before transitioning to an accept or reject state if needed.

An analogous argument applies for f_2. Note the synchronicity of the branches guarantee they operate exactly the same and terminate at the same time. The repeated transition between accept and reject states guarantee the only cells relevant for the final decision of A are those in the branches which are still "active" (in the sense they are still evaluating f).

It is concluded that A accepts f if and only if it evaluates to true and rejects it otherwise. A runs in polynomial time (in $|f|$) since f has at most $|f|$ predicates and since evaluating a predicate requires polynomial time in $|f|$. □

For the converse, we express an XCA computation as a $\mathrm{SAT}^{\wedge}$-$\mathrm{TAUT}^{\vee}$ instance. The main effort here lies on defining the appropriate "variables":

Definition 4 ($\mathbf{STATE}_\forall$). *Let A be an XCA, and let V_A be the set of triples (w, t, z), w being an input for A, $t \in \{0,1\}^+$ a (standard) binary representation of $\tau \in \mathbb{N}_0$, and z a state of A. $\mathrm{STATE}_\forall(A) \subseteq V_A$ is the subset of triples such that, if A is given w as input, then after τ steps all active cells are in state z.*

Lemma 2. *If A has polynomial time complexity,* $\mathrm{STATE}_\forall(A) \in \mathbf{coNP}$.

Proof. Fix an efficiently computable polynomial $p\colon \mathbb{N} \to \mathbb{N}_0$ such that A always terminates after at most $p(n)$ steps for an input of size n. Consider an NTM T which non-deterministically picks an active cell in step τ of A for input w, computes its state z' in polynomial time, and accepts if and only if $z' = z$. Additionally, suppose that, by doing so, T covers all active cells of A in step τ. Furthermore, to ensure T only simulates A for a polynomial number of steps (in $|w|$), T determines $p(|w|)$ and rejects in case $\tau > p(|w|)$; this is not a restriction because of the choice of p. The claim follows immediately from the existence of such a T: If all computation branches of T accept, then in step τ all cells of A are in state z; otherwise, there is a cell in a state which is not z, and T rejects. The rest of the proof is concerned with the construction of T as well as showing that its branches cover all active cells of A in its final configuration for the input w.

To compute the state of an active cell in step τ, T calculates a series of *subconfigurations* $c_0, \dots, c_\tau$ of A, that is, contiguous excerpts of the global configuration of A. As the number of cells in an XCA may increase exponentially in the number of computation steps, bounding c_i is essential to ensure T runs in polynomial time; in particular, T maintains $|c_i| = 1 + 2(\tau - i)$, thus ensuring the lengths of the c_i are linear in τ (which, in turn, is polynomial in $|w|$). This choice of length for the c_i ensures each of the subconfigurations correspond to a cell of A surrounded by $\tau - i$ cells on either side.[3] The non-determinism of T is used precisely in picking the cells from c_i which are to be included in the next subconfiguration c_{i+1}.

The initial subconfiguration c_0 is set to be $q^{2\tau} w q^{2\tau}$, thus containing the input word as well as (as shall be proven) a sufficiently large number of surrounding quiescent cells. To obtain c_{i+1} from c_i, T applies the global transition function of A on c_i to obtain a new temporary subconfiguration c'_{i+1}. The next state of

[3] That is to say, each c_i corresponds to the so-called extended $(\tau - i)$-neighborhood of a cell of A.

the two "boundary" cells (i.e, those belonging to indices 0 and $|c_i| - 1$) cannot be determined since the state of their neighbors is unknown; thus, they are excluded from c'_{i+1}. The same applies to any hidden cell which remains so. c'_{i+1} contains, as a result, $|c_i| - 2$ active cells from the previous configuration c_i in addition to a maximum of $|c_i| - 1$ previously hidden cells. To maintain $|c_i| = 1 + 2(\tau - i)$, T non-deterministically chooses a contiguous subset s of c'_{i+1} containing $1 + 2(\tau - i)$ cells and sets c_{i+1} to s; when doing so, T ignores subsets containing solely quiescent cells. That there are enough cells to choose from is, again, ensured by the fact that c'_{i+1} contains at least $|c_i| - 2$ active cells from the previous configuration c_i.

The process of selecting a next subconfiguration c_{i+1} from c_i is depicted in figure 2. In the figure, $|c_i|$ has been replaced with n for legibility. T at first applies the global transition function of A to obtain an intermediate subconfiguration c'_{i+1} with $m = |c'_{i+1}|$ cells. Because of hidden cells, c'_{i+1} may consist of $n - 2 \leq m \leq 2n - 3$ cells. Non-determinism is used to select a contiguous subconfiguration of $n - 2$ cells, thus giving rise to c_{i+1}.

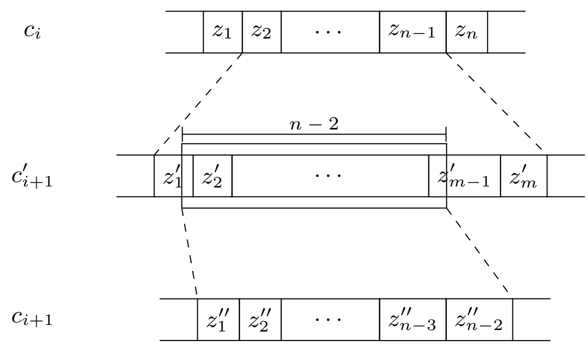

Figure 2. Illustration of how T obtains the next subconfiguration c_{i+1} from c_i.

This concludes the construction of T. Note it runs in polynomial time since the invariant $|c_i| = 1 + 2(\tau - i)$ guarantees the number of states T computes in each step is bounded by a multiple of τ, which, as previously discussed, is bounded by $p(|w|)$. Only $|w|$ has to be taken into account when estimating the time complexity of T since the encoding of z is constant with respect to $|w|$ while that of t is logarithmic with respect to $p(|w|)$; expressing the problem instance (w, t, z) requires, as a result, asymptotically $|w|$ space.

To show all active cells of A in step τ are covered by T, it suffices to prove the following by induction: Let $i \in \{0, \ldots, \tau\}$, and let $z_1, \ldots, z_m$ be the active cells of A in step i; then T covers all subconfigurations of $q^{2(\tau-i)} z_1 \cdots z_m q^{2(\tau-i)}$ of size $1 + 2(\tau - i)$. Note this corresponds to T covering all subconfigurations of A in step i which contain at least one active cell; thus, when T reaches step τ, it covers all subconfigurations of $z_1 \cdots z_m$ of size 1, that is, all active cells.

The induction basis follows from $c_0 = q^{2\tau} w q^{2\tau}$. For the induction step, fix a step $0 < i \leq \tau$ and assume the claim holds for all steps prior to i. To each subconfiguration of $q^{2(\tau-i)}z_1 \cdots z_m q^{2(\tau-i)}$ having size $1+2(\tau-i)$ corresponds a cell w which is located in its center; such subconfiguration is denoted by $s_i(w)$. Now let $s_i(w)$ be given, in which case there are three cases to be considered: w was active in step $i-1$; w was a hidden cell which became active in the transition to step i; or w was a quiescent cell in step $i-1$ and, by $|s_i(w)| = 1+2(\tau-i)$, is at most $\tau - i$ cells away from z_1 or z_m.

In the first case, by the induction hypothesis, there is a value of c_{i-1} corresponding to $s_{i-1}(w)$; since only the two boundary cells are present in c_{i-1} but not in c'_i, choosing c_i from c'_i with w as its middle cell yields $s_i(w)$. In the second, for any of the two parents p_1 and p_2 of w, there are, by the induction hypothesis, values of c_{i-1} which equal $s_{i-1}(p_1)$ and $s_{i-1}(p_2)$; in either case, choosing c_i from c'_i with w as its middle cell again yields $s_i(w)$.

Finally, if w was a quiescent cell, then, without loss of generality, consider the case in which w was located to the left of the active cells in step $i-1$. By the induction hypothesis, for each cell w' up to $\tau - i + 1$ cells away from the leftmost active cell z_1 there is a value of c_{i-1} corresponding to $s_{i-1}(w')$, and the first case applies; the only exception is if c_i would then contain only quiescent cells, in which case w would be located strictly more than $\tau - i$ cells away from z_1, thus contradicting our previous assumption. The claim follows. □

The following proposition completes our argument by reduction:

Proposition 2. $\mathbf{XCAP} \subseteq \leq^p_{tt}(\mathbf{NP})$.

Proof. Let $L \in \mathbf{XCAP}$, and let A be an XCA for L whose time complexity is bounded by a polynomial $p\colon \mathbb{N} \to \mathbb{N}_0$. Additionally, let w be an input for A, V_A be as in Definition 4, and let $V = V_A \cdot \{\Subset_{\mathrm{STATE}_\forall(A)}\}$, where $\Subset_{\mathrm{STATE}_\forall(A)}$ is a syntactic symbol standing for membership in $\mathrm{STATE}_\forall(A)$ (cf. Definition 1). Define $f_0(w), \dots, f_{p(n)}(w) \in \mathrm{BOOL}_V$ recursively as follows:

$$f_i(w) := \begin{cases} \vee(\,(w,i,a) \Subset_{\mathrm{STATE}_\forall(A)}, \wedge(\,\neg(\,(w,i,r) \Subset_{\mathrm{STATE}_\forall(A)}\,), f_{i+1}\,)\,), & i \neq p(n) \\ \vee(\,(w,p(n),a) \Subset_{\mathrm{STATE}_\forall(A)}, \neg(\,(w,p(n),r) \Subset_{\mathrm{STATE}_\forall(A)}\,)\,), & i = p(n) \end{cases}$$

Lemma 2 together with the **coNP**-completeness of TAUT (see Theorem 1) ensures each subformula of the form $(w,i,a) \Subset_{\mathrm{STATE}_\forall(A)}$ is polynomial-time many-one reducible to an equivalent[4] $\mathrm{SAT}^\wedge$-$\mathrm{TAUT}^\vee$ formula $g \Subset_{\mathrm{TAUT}}$, g being a TAUT instance. Similarly, each subformula $\neg((w,i,a) \Subset_{\mathrm{STATE}_\forall(A)})$ is reducible to an equivalent formula $h \Subset_{\mathrm{SAT}}$. Since each of the $f_i(w)$ may contain only polynomially (respective to $|w|$) many connectives, each is polynomial-time (many-one) reducible to an equivalent $\mathrm{SAT}^\wedge$-$\mathrm{TAUT}^\vee$ instance $f'_i(w)$.

By the definition of XCA (i.e., Definitions 2 and 3) and our choice of p, $f'(w) := f'_0(w)$ is true if and only if A accepts w. Since $f'(w)$ is such that $|f'(w)|$ is polynomial in $|w|$, this provides a polynomial-time (many-one) reduction of L to a problem instance of $\mathrm{SAT}^\wedge$-$\mathrm{TAUT}^\vee \in \leq^p_{tt}(\mathbf{NP})$. The claim follows. □

[4] In the sense of evaluating to the same truth value under the respective interpretations (see Definition 1).

4 Immediate Implications

This section covers some immediate corollaries of Theorem 4 regarding XCA variants. In particular, we address XCAs with multiple accept and reject states, followed by XCAs with acceptance conditions differing from that from Definition 3, in particular the two other classical acceptance conditions for CAs [16].

4.1 XCAs with Multiple Accept and Reject States

Recall the definition of an XCA specifies a single accept and a single reject state (see Section 2.2). Consider XCAs with multiple accept and reject states. As shall be proven, the respective polynomial-time class (**MAR-XCAP**) remains equal to **XCAP**. In the case of TMs, the equivalent result (i.e., TMs with a single accept and a single reject state are as efficient as standard TMs) is trivial, but such is not the case for XCAs. Recall the acceptance condition of an XCA requires orchestrating the states of multiple, possibly exponentially many cells. In addition, an XCA with multiple accept states may, for instance, attempt to accept whilst saving its current state (i.e., a cell in state z may assume an accept state a_z while simultaneously saving state z). Such is not the case for standard XCAs (i.e., as specified in Definition 3), in which all accepting cells have necessarily the same state.

Definition 5 (MAR-XCA, MAR-XCAP). *A* multiple accept-reject XCA *(*MAR-XCA*) D is an XCA with state set Q and which admits subsets $A, R \subseteq Q$ of accept and reject states, respectively. D accepts (rejects) if its active cells all have states in A (R), and it halts upon accepting or rejecting. In addition, D is required to either accept or reject its input after a finite number of steps.* **MAR-XCAP** *denotes the MAR-XCA analogue of* **XCAP**.

The following generalizes $\text{STATE}_\forall$ (see Definition 4 and Lemma 2) to the case of MAR-XCAs:

Definition 6 ($\mathbf{STATE}_\forall^{\mathbf{MAR}}$). *Let A be a MAR-XCA with state set Q, and let V_A be the set of triples (w, t, Z), w being an input for A, $t \in \{0,1\}^+$ a binary encoding of $\tau \in \mathbb{N}_0$, and $Z \subseteq Q$. $\text{STATE}_\forall^{MAR}(A) \subseteq V_A$ is the subset of triples such that, if A is given w as input, after t steps all active cells have states in Z.*

Lemma 3. *If A has polynomial time complexity,* $\text{STATE}_\forall^{MAR}(A) \in \mathbf{coNP}$.

Proof. Simply adapt the NTM from the proof of Lemma 2 so as to accept if and only if the final state is contained in Z. □

Proceeding as in the proof of Proposition 2 (simply using $\text{STATE}_\forall^{\text{MAR}}$ instead of $\text{STATE}_\forall$) yields:

Theorem 5. **MAR-XCAP** = **XCAP**.

Proof. Define formulas $f_i(w)$ as in the proof of Proposition 2 while replacing $\text{STATE}_\forall$ with $\text{STATE}_\forall^{\text{MAR}}$, the accept state a with the set A, and the reject state r with the set R. Lemma 3 guarantees the reductions to $\text{SAT}^\wedge\text{-TAUT}^\vee$ are all efficient. This implies $\mathbf{MAR\text{-}XCAP} \subseteq \leq_{tt}^p(\mathbf{NP}) = \mathbf{XCAP}$. Since MAR-XCAs are a generalization of XCAs, the converse inclusion is trivial. □

4.2 Existential XCA

The remainder of this section is concerned with XCAs variants which use the two other acceptance conditions from [16, 6, 20, 8]. The first is that of a single final state being present in the CA's configuration sufficing for termination.

Definition 7 (EXCA, EXCAP). *An* existential[5] XCA *(EXCA) is an XCA with the following acceptance condition: If at least one of its cells is in the accept (reject) state a (r), then the EXCA accepts (rejects). The coexistence of accept and reject states in the same global configuration is disallowed (and any machine contradicting this requirement is, by definition,* not *an EXCA).* **EXCAP** *denotes the EXCA analogue of* **XCAP**.

Disallowing the coexistence of accept and reject states in the global configuration of an EXCA is necessary to ensure a consistent accepting behavior. An alternative would be to establish a priority relation between the two (e.g., an accept state overrules a reject one); nevertheless, this behavior can be emulated by our chosen variant with only constant delay. This is accomplished by introducing binary counters to delay state transitions and assure, for instance, that accept and reject states exist only in even- and odd-numbered steps, respectively.

Despite the diverse accepting behavior, the following holds for EXCAs:

Theorem 6. $\mathbf{EXCAP} = \mathbf{XCAP} = \leq_{tt}^{p}(\mathbf{NP})$.

Note this is an equivalence between two disparately complex acceptance behaviors: As by Definition 3, in an XCA all cells must agree on the final decision; on the other hand, in an EXCA, a single, arbitrary cell suffices. We ascribe this phenomenon to $\mathbf{XCAP} = \leq_{tt}^{p}(\mathbf{NP})$ being equal to its complementary class.

As for the proof of Theorem 6, first note that Proposition 1 may easily be restated in the context of EXCAs:

Proposition 3. $\leq_{tt}^{p}(\mathbf{NP}) \subseteq \mathbf{EXCAP}$.

Proof. Consider the XCA A for $\text{SAT}^{\wedge}\text{-TAUT}^{\vee}$ from the proof of Proposition 1; by adapting it, we obtain an EXCA B which decides $\text{TAUT}^{\wedge}\text{-SAT}^{\vee}$ in polynomial time. $\text{TAUT}^{\wedge}\text{-SAT}^{\vee}$ is the problem analogous to $\text{SAT}^{\wedge}\text{-TAUT}^{\vee}$ and which is obtained by exchanging "TAUT" and "SAT" in Definition 1. As $\text{SAT}^{\wedge}\text{-TAUT}^{\vee}$, $\text{TAUT}^{\wedge}\text{-SAT}^{\vee}$ is $\leq_{tt}^{p}(\mathbf{NP})$-complete (see Theorem 2).

To evaluate a predicate of the form $f \in_{\text{TAUT}}$, B proceeds as A and emulates the behavior of the XCA deciding TAUT (see Theorem 3); however, unlike A, the computation branches of B which evaluate to false reject immediately while it is those that evaluate to true that continue evaluating the input formula. As a result, if $f \in \text{TAUT}$, all branches of B evaluate to true and continue evaluating the input in a synchronous manner; otherwise, there is a branch evaluating to false, and, since a single rejecting cell suffices for it to reject, B rejects immediately. The evaluation of $f \in_{\text{SAT}}$ is carried out analogously.

The modifications to A to obtain B do not impact its time complexity whatsoever; thus, B also has polynomial time complexity. □

[5] An allusion to the existential states of alternating Turing machines (ATMs)

For the converse inclusion, consider the following **NP** analogue of the $\text{STATE}_\forall$ language (cf. Definition 4 and Lemma 2):

Definition 8 ($\mathbf{STATE}_\exists$). *Let A be an XCA, and let V be the set of triples (w, t, z) as in definition 4.* $\text{STATE}_\exists \subseteq V$ *is the subset of triples such that, for the input w, after t steps* at least one *of the active cells of A is in state z.*

Lemma 4. *If A has polynomial time complexity,* $\text{STATE}_\exists(A) \in \mathbf{NP}$.

Proof. Consider the NTM T from Lemma 2 and notice that, if any of the active cells of A in step τ have state z, then T will have at least one accepting branch; otherwise, none of the active cells of A in step τ have state z; thus, all branches of T are rejecting. □

Using Lemma 4 to proceed as in Proposition 2 yields the following, from which Theorem 6 follows:

Proposition 4. $\mathbf{EXCAP} \subseteq \leq_{tt}^{p}(\mathbf{NP})$.

4.3 One-Cell-Decision XCA

We turn to the discussion of XCAs whose acceptance condition is defined in terms of a distinguished cell which directs the automaton's decision, considered the standard acceptance condition for CAs [9]. This behavior is similar to the existential variant in the sense that a single cell suffices to trigger the automaton's termination; the difference lies in the position of this single cell being immutable.

We consider only the case in which the decision cell is the leftmost active cell in the initial configuration (i.e., cell 0). By a *one-cell-decision XCA* (*1XCA*) we refer to an XCA which accepts if and only if 0 is in the accept state and rejects if and only if cell zero is in the reject state. Let **1XCAP** denote the polynomial-time class of 1XCAs.

The position of the decision cell is fixed; with a polynomial-time restriction in place, it can only communicate with cells which are a polynomial (in the length of the input) number of steps apart. As a result, despite a 1XCA being able to efficiently increase its number of active cells exponentially (see Lemma 1), any cells impacting its decision behavior must be at most a polynomial number of cells away from the decision cell. Thus:

Theorem 7. $\mathbf{1XCAP} = \mathbf{P}$.

Proof. The inclusion $\mathbf{1XCAP} \supseteq \mathbf{P}$ is trivial. For the converse, recall the construction of the NTM T in Lemma 2. T can be modified such that it works deterministically and always chooses the next configuration c_{i+1} from c_i by selecting cell zero as the middle cell. If cell zero is accepting, then T accepts immediately; if it is rejecting, then T also rejects immediately. This yields a simulation of a 1XCA by a (deterministic) TM which is only polynomially slower, thus implying $\mathbf{1XCAP} \subseteq \mathbf{P}$. □

5 Conclusion

This paper summarized the results of [13], which, in turn, expanded on the complexity-theoretic aspects of XCAs from [14]. The main result was the characterization $\mathbf{XCAP} = \leq_{tt}^{p}(\mathbf{NP})$ in Section 3.1. In Section 4, XCAs with multiple accept and reject states were shown to be equivalent to the original model. Also in Section 4, two other variants based on varying varying acceptance conditions were considered: the existential (EXCA), in which a single, though arbitrary cell may direct the automaton's response; and the one-cell-decision XCA (1XCA), in which a fixed cell does so. In the first case, it was shown that the polynomial-time class **EXCAP** equals **XCAP**; in the latter, it was shown that the polynomial-time class **1XCAP** of 1XCAs equals **P**.

This paper has covered some XCA variants with diverse acceptance conditions. A topic for future work might be considering further variations in this sense (e.g., XCAs whose acceptance condition is based on *majority* instead of *unanimity*). Another avenue of research lies in restricting the capabilities of XCAs and analyzing the effects thereof (e.g., restricting 1XCAs or SXCAs to a polynomial number of cells). A final open question is determining what polynomial speedups, if any, 1XCAs provide with respect to 1CAs.

Acknowledgments

The author would like to thank Thomas Worsch for his mentoring, encouragement, and support during the writing of this paper. The author would also like to thank Dennis Hofheinz for pointing out a crucial mistake in a preliminary version of this paper as well as the anonymous referees for their valuable remarks and suggestions.

References

[1] Pablo Arrighi and Gilles Dowek. "Causal graph dynamics." In: *Inf. Comput.* 223 (2013), pp. 78–93.

[2] Samuel R. Buss and Louise Hay. "On Truth-Table Reducibility to SAT." In: *Inf. Comput.* 91.1 (1991), pp. 86–102.

[3] Stephen A. Cook. "The Complexity of Theorem-proving Procedures." In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing.* STOC '71. Shaker Heights, Ohio, USA: ACM, 1971, pp. 151–158.

[4] Stefan S. Dantchev. "Dynamic Neighbourhood Cellular Automata." In: *Visions of Computer Science - BCS International Academic Conference, Imperial College, London, UK, 22-24 September 2008.* 2008, pp. 60–68.

[5] Jean-Christophe Dubacq. "Different Kinds of Neighborhood-Varying Cellular Automata." Maîtrise/honour bachelor degree. École normale superiéure de Lyon, 1994.

[6] Oscar H. Ibarra et al. "Fast Parallel Language Recognition by Cellular Automata." In: *Theor. Comput. Sci.* 41 (1985), pp. 231–246.

[7] Andrew Ilachinski and Paul Halpern. “Structurally Dynamic Cellular Automata.” In: *Complex Systems* 1.3 (1987), pp. 503–527.
[8] Sam Kim and Robert McCloskey. “A Characterization of Constant-Time Cellular Automata Computation.” In: *Phys. D* 45.1-3 (Oct. 1990), pp. 404–419.
[9] Martin Kutrib. “Cellular Automata and Language Theory.” In: *Encyclopedia of Complexity and Systems Science*. 2009, pp. 800–823.
[10] Martin Kutrib et al. “Shrinking One-Way Cellular Automata.” In: *Cellular Automata and Discrete Complex Systems - 21st IFIP WG 1.5 International Workshop, AUTOMATA 2015, Turku, Finland, June 8-10, 2015. Proceedings*. 2015, pp. 141–154.
[11] Richard E. Ladner and Nancy A. Lynch. “Relativization of Questions About Log Space Computability.” In: *Mathematical Systems Theory* 10 (1976), pp. 19–32.
[12] Richard E. Ladner et al. “A Comparison of Polynomial Time Reducibilities.” In: *Theor. Comput. Sci.* 1.2 (1975), pp. 103–123.
[13] Augusto Modanese. “Complexity-Theoretical Aspects of Expanding Cellular Automata.” Master’s thesis. Karlsruhe Institute of Technology, 2018.
[14] Augusto Modanese. “Shrinking and Expanding One-Dimensional Cellular Automata.” Bachelor’s thesis. Karlsruhe Institute of Technology, 2016.
[15] Augusto Modanese and Thomas Worsch. “Shrinking and Expanding Cellular Automata.” In: *Cellular Automata and Discrete Complex Systems - 22nd IFIP WG 1.5 International Workshop, AUTOMATA 2016, Zurich, Switzerland, June 15-17, 2016, Proceedings*. 2016, pp. 159–169.
[16] Azriel Rosenfeld. *Picture languages*. New York: Academic Press, 1979.
[17] Azriel Rosenfeld and Angela Y. Wu. “Reconfigurable Cellular Computers.” In: *Information and Control* 50.1 (1981), pp. 60–84.
[18] Azriel Rosenfeld et al. “Fast language acceptance by shrinking cellular automata.” In: *Inf. Sci.* 30.1 (1983), pp. 47–53.
[19] Alvy Ray Smith III. “Simple Computation-Universal Cellular Spaces.” In: *J. ACM* 18.2 (1971), pp. 339–353.
[20] Rudolph Sommerhalder and S. Christian van Westrhenen. “Parallel Language Recognition in Constant Time by Cellular Automata.” In: *Acta Inf.* 19 (1983), pp. 397–407.
[21] Kohji Tomita et al. “Graph automata: natural expression of self-reproduction.” In: *Physica D: Nonlinear Phenomena* 171.4 (2002), pp. 197–210.
[22] Klaus W. Wagner. “Bounded Query Classes.” In: *SIAM J. Comput.* 19.5 (1990), pp. 833–846.