Complex Systems **1** (1987) 453–474

NP-Complete Problems in Cellular Automata

Frederic Green
Department of Mathematics and Computer Science, Clark University, Worcester, MA 01610, USA

Abstract. An example of a cellular automaton (CA) is given in which the following problems are NP-complete: (i) determining if a given subconfiguration s can be generated after $|s|$ time steps, (ii) determining if a given subconfiguration s will recur after $|s|$ time steps, (iii) determining if a given temporal sequence of states s can be generated in $|s|$ time steps. It is also found that the CA constructed has an NP-hard limit language.

1. Introduction

Cellular automata (CAs) are examples of homogeneous systems of components between which only local communication exists. There has recently been a renewal of interest in CAs as paradigms for discrete dynamical systems as well as parallel computation (see [1]; for an extensive collection of reprints and an annotated bibliography, see [2]). In reference [3], a number of open problems having to do with cellular automata were posed. One of them is the question of how common NP-complete problems are in cellular automata. The answer may have a significant bearing on our practical ability to predict the outcome of various chaotic phenomena based on computation. In addition, some of the practical difficulties of efficiently programming parallel computers might be revealed [1]. To date, however, there are no proven results in the literature on any NP-complete problems in this area, let alone on how common they are. In this paper, we present an example of a particular CA which has NP-complete problems associated with it.

Most NP-complete problems [4] apparently deal with constructs consisting of a large set of components having long-range connections between them. For example, an instance of Hamilton path consists of a graph, the nodes of which can be arbitrarily connected. Equally important is the lack of regularity in the structure; in the graphical case, any constraints on a problem which make the graph symmetric or homogeneous in some way often tend to make the associated problem easier. Therefore, because of their uniform construction, there is some intrinsic interest in examining the nature of NP-complete problems associated with CAs.

Informally, a CA (for our purposes) is a one-dimensional, two-way infinite array of cells. Each cell can be in one of a number of states. The array is evolved in time by synchronously updating the state of each cell according to a rule which depends on the value of a cell and its two nearest neighbors. An infinite array of states at a given time is called a configuration, and a finite array of contiguous states is called a sub-configuration. We will see that there is a CA for which each of the following problems are NP-complete:

1. CA Preimage: Given a subconfiguration of length K, is there a configuration that could have led to it in K time steps?
2. CA Subconfiguration Recurrence: Given a subconfiguration of length K, will it be the same K time steps later?
3. Temporal Sequence Preimage: Given a temporal sequence of K states in a given cell, is there a subconfiguration that could have led to it in K time steps?

Generalizations of problems 1 and 3 have previously been conjectured as being NP-complete [3,5,6]. The generalizations simply ask the same questions, but with the number of time steps being independent of the size of the input subconfiguration. As we shall see, it is not clear that these generalizations are in NP. However, the results of this paper do prove that these generalized versions are NP-hard. Both 1 and 3 have a direct bearing on the problem of computing entropies [3]. Problem 2 has not been previously mentioned, and its NP-completeness is established as an interesting by-product of the proof technique used for 1. The difficulty of problem 2 places limits on our ability to predict the outcome of a CA computation on the basis of incomplete information, even when the active part of the configuration is finite.

Another question raised in [3] is: what limit sets can CAs produce? The limit set is essentially the set of configurations generable in infinite time. Hurd [7] has constructed examples of CAs with strictly non-regular, non-context-free, and non-recursively enumerable limit sets. The question as to what classes of languages can be generated when they are classified according to their computational complexity naturally arises. It is quite possible that any set can be generated in the limit. Finding examples is, however, nontrivial. It is found that the CA constructed in this paper has an NP-hard limit set. In this case, this appears to be a direct consequence of the NP-hardness of the preimage problem.

Previous work on computational complexity of CAs for the most part deals with language recognition. There are many ways to define a language recognized by a CA. One way is to specify a state, or set of states, to which an input string must evolve in order to be included in the language. Such characterizations are given in, for example, [8,9,10]. It is noted in [8] that the set of languages recognized by bounded (i.e., finite) *non*deterministic CAs is the set of context-sensitive languages. This is true because it is possible to simulate any linear bounded automaton with a bounded CA. In fact, there are fixed deterministic linear bounded automata for which the language acceptance problem is PSPACE-complete [4]. It follows immediately that there exists a particular deterministic bounded CA for which the language recognition problem is PSPACE-complete.

In contrast, it is not at all clear how to prove results such as the NP-completeness of the preimage problem for a particular CA. An NP-completeness proof for a problem must provide either a generic reduction of any language in NP to the problem of interest, or a polynomial time reduction from a previously known NP-complete problem. To implement the former strategy, it seems we must simulate any polynomial time computation of any nondeterministic Turing machine with (for example) an instance of CA preimage. This would require encoding the computation of a nondeterministic Turing machine in the substring (as in the standard proof of Cook's theorem), the CA rules being such that the substring has a preimage if and only if the Turing machine accepts in polynomial time. This turns out to be prohibitively complicated, and leads to an enormous number of states in the CA.

We are therefore faced with the simpler alternative of reducing from a known NP-complete problem. However, here we also have a problem. When an instance of an NP-complete problem is encoded in the substring, once again the CA rules must be such that (for example) the substring has a preimage if and only if the problem is a "yes"-instance of the problem (using the terminology of [4]). The difficulty is that widely separated parts of the substring must communicate with each other, and this has to be effected in the inherently local CA rules. This communication problem is solved conceptually by breaking the CA up into two coupled CAs, where states in one of the CAs can move, while states in the other one are "stationary." This is similar to separating any computing device into its control part (e.g., a Turing machine's finite control) and its output part (e.g., the tape of a Turing machine). The difference with Turing machines is that we will take advantage of the parallelism of CAs. Actually, a similar difficulty would exist if one were trying to prove NP-completeness of a preimage problem for Turing machines. Unfortunately, the NP-completeness of any preimage problem for Turing machines is apparently not known, so it is necessary to reduce from some other NP-complete problem. The known NP-complete problem we use is 3SAT. The fact that there are three literals per clause actually does not impose any restrictions on any of the problems, but it makes the encoding process simpler.

In section 2, the basic definitions and notational conventions used throughout the paper are set down. Section 3 contains the proof of three of the main results: NP-completeness of the preimage and substring recurrence problems and NP-hardness of the limit language. In section 4, the NP-completeness of the temporal sequence preimage problem is proved. Section 5 discusses the implications of these results.

2. Preliminaries

A *cellular automaton* (CA) is a pair (Q, δ), where Q is a finite set of states, and $\delta : Q \times Q \times Q \to Q$ is a *transition function*. (The definition of a CA often includes a distinguished $q_0 \in Q$ called the quiescent state. Here, there is no special reason to separate q_0 from the other states, so it is not included in the definition.) A *configuration* of a CA at time $t \in Z, C^{(t)} : Z \to Q$, is an assignment of a unique state to each "cell", where a cell is represented by an integer. $C^{(t)}(i), i \in Z$, is the "state of cell i at time t." A configuration at time t uniquely determines a configuration at time $t+1$ as follows:

$$C^{(t+1)} = \{(i,q) | q = \delta(C^{(t)}(i-1), C^{(t)}(i), C^{(t)}(i+1)), i \in Z\}$$

The definition of δ can be extended so that we may write the above more compactly as $C^{(t+1)} = \delta(C^{(t)})$. We will generally refer to $C^{(0)}$ as an *initial* or *starting* configuration. A *computation* of a CA of length n is a sequence of configurations $C^{(0)}, C^{(1)}, \ldots C^{(n-1)}$. If $C^{(i)}$ and $C^{(j)}, i \geq j$, are two configurations in a computation, then we say that $C^{(i)}$ computes $C^{(j)}$, and write $C^{(i)} \vdash^* C^{(j)}$. To specify that a configuration C_a computes a configuration C_b in t time steps (i.e., $C_b = \delta^t(C_a)$), we write $C_a \vdash^t C_b$.

A *string* associated with a CA (Q, δ) is an element of Q^*. We write $s = s_1 s_2 s_3 \ldots s_k$ where $|s| \equiv k$ is the length of s, $s_i \in Q$ for all $i \in \{1 \ldots k\}$, and s_i is referred to as the i^{th} element of s. A *configuration substring* or *subconfiguration* is a string $s = s_1 s_2 \ldots s_k$ where $s_i = C^{(t)}(i-j)$ for some j, t and each $i \in \{1 \ldots k\}$. Such a string s is said to be a substring of $C^{(t)}$ occupying cells $1-j$ through $k-j$. All strings we will encounter are configuration substrings, so unless there is danger of confusion, "string" or "substring" will henceforth mean "configuration substring."

Let $s^{(t)}$ be a substring of $C^{(t)}$ occupying cells l through k and $s^{(t+p)}$ be a substring of $C^{(t+p)}$ occupying the same cells. As with configurations, we say that $s^{(t)}$ computes $s^{(t+p)}$ in p steps and write $s^{(t)} \vdash^p s^{(t+p)}$, if $C^{(t)} \vdash^p C^{(t+p)}$.

If $X \vdash^K Y$, where X and Y are both substrings or both configurations, then X is said to be a *K^{th}-preimage* of Y. If X and Y are strings, and X is a substring of configuration C, then we will also refer to C as a K^{th}-preimage of Y, and write $C \vdash^K Y$.

A *finite* CA is the same as a CA, but with a configuration $C : [1..N] \to Q$ defined on a finite array $[1..N]$ of cells. Finite CAs will be taken to have periodic boundary conditions, i.e., $C^{(t+1)}(N) = \delta(C^{(t)}(N-1), C^{(t)}(N), C^{(t)}(1))$ and $C^{(t+1)}(1) = \delta(C^{(t)}(N), C^{(t)}(1), C^{(t)}(2))$.

For conceptual reasons, it is advantageous to introduce the notion of a *coupled pair of CAs* (CPCA). A CPCA is a quadruple $(Q_1 Q_2, \delta_1, \delta_2)$ where Q_1 and Q_2 are finite sets of states, and

$$\delta_1 : (Q_1 \times Q_2) \times (Q_1 \times Q_2) \times (Q_1 \times Q_2) \to Q_1$$

$$\delta_2 : (Q_1 \times Q_2) \times (Q_1 \times Q_2) \times (Q_1 \times Q_2) \to Q_2$$

are transition functions. A CPCA $(Q_1, Q_2, \delta_1, \delta_2)$ is nothing more than a CA (Q, δ) with $Q = Q_1 \times Q_2$ and $\delta = \delta_1 \times \delta_2$. However, we will find it helpful to think of a CPCA intuitively as two CAs, $\mathrm{CA}_1(Q_1, \delta_1)$ and $\mathrm{CA}_2(Q_2, \delta_2)$ which interact via the transition functions δ_1 and δ_2. Configurations can be defined in like manner for CA_1 and CA_2 in a CPCA. We will denote configurations in CA_1 by $C_1^{(t)} : Z \rightarrow Q_1$ and in CA_2 by $C_2^{(t)} : Z \rightarrow Q_2$. Computations, strings, and configuration substrings of either $C_1^{(t)}$ or $C_2^{(t)}$ are also defined in analogy to the CA case. In particular, we will find it necessary to refer to computations of strings in CA_1 or CA_2. Thus, if $s_i^{(t)}$ is a substring of $C_i^{(t)}$ and $s_i^{(t+p)}$ is a substring of $C_i^{(t+p)}$, $(i = 1$ or $2)$, then we write $s_i^{(t)} \vdash^p s_i^{(t+p)}$ if $C_i^{(t)} \vdash^p C_i^{(t+p)}$. We also need some notation for adjacent strings in CA_1 and CA_2, that is, strings occupying corresponding sets of cells. Substrings in CA_1 will be written above substrings in CA_2. For example,

$$\begin{array}{l} s_1s_2s_3\ldots s_k \\ s'_1s'_2s'_3\ldots s'_k \\ \uparrow \\ i \end{array}$$

means that $C_1^{(t)}(i) = s_1, C_1^{(t)}(i+1) = s_2, \ldots C_1^{(t)}(i+k-1) = s_k$, and $C_2^{(t)}(i) = s'_1, C_2^{(t)}(i+1) = s'_2, \ldots C_2^{(t)}(i+k-1) = s'_k$. If two states are vertically aligned as above (e.g., s_2 and s'_2, s_3 and s'_3, etc.) they will always be understood to be in corresponding cells of the CPCA. Hence, the position of the first cell of a string (i in the above example) will be omitted if it is irrelevant. Similar notation is introduced for overlapping strings, e.g.,

$$\begin{array}{lll} & s_1 & s_2\ldots s_k \\ s'_1s'_2 \;.. & s'_{k'-1} & s'_{k'} \end{array}$$

indicates that s_1 and $s'_{k'-1}$, and s_2 and $s'_{k'}$, respectively occupy corresponding cells. Cells not given values are irrelevant to the discussion. Neighboring strings in CA_1 and CA_2 are denoted by

$$\begin{array}{r|l} & s \\ s' & \end{array} \quad \text{or} \quad \begin{array}{r|l} s & \\ & s' \end{array}$$

the former indicating that the rightmost element of s' is in the cell immediately to the left of the cell containing the leftmost element of s, and the latter vice versa. There is one final notation introduced for CPCAs. In the encodings that follow, there will be a "quiescent state" which we will denote by an underscore, _. Strings of quiescent states, when there is no ambiguity, will be denoted by a continuous line. Thus, for example,

$$\begin{array}{r|l} s_1s_2\ldots s_k & \\ & \overline{s'_1s'_2\ldots s'_k} \end{array}$$

means that s_k (in $C_1^{(t)}$) is in the cell immediately to the left of the cell containing s'_1 (in $C_2^{(t)}$), and all the cells (in $C_1^{(t)}$) corresponding to the s'_i (in $C_2^{(t)}$), $i = 1, \ldots k$, contain the quiescent state "_".

Polynomial-time reductions will be denoted by $\propto$. Thus, if A and B are languages over some alphabet Σ, $A \propto B$ means that there is a function $f : \Sigma^* \to \Sigma^*$ such that for any $a \in \Sigma^*, f(a)$ is computable in a time polynomial in $|a|$, and $a \in A$ if and only if $f(a) \in B$. Regarding A and B as decision problems, the definition of $A \propto B$ is the same, except to replace "$a \in A$" by "a is a yes-instance of A" and "$f(a) \in B$" by "$f(a)$ is a yes-instance of B."

The reduction will be from 3SAT, for which we introduce our conventions now. A *literal* is a symbol in one of two finite sets, $L = \{l_1, l_2, \ldots l_m\}$ or $\bar{L} = \{\bar{l}_1, \bar{l}_2, \ldots \bar{l}_m\}$. The overbar denotes negation. A *truth assignment* τ is an m-tuple $(\lambda_1, \lambda_2, \ldots \lambda_m)$ where $\lambda_i \in \{l_i, \bar{l}_i\}$ for each $i \in \{1 \ldots m\}$. A *clause* c is a finite set of literals. In the case of 3SAT, $|c| = 3$ for any clause c. A *conjunctive normal form* (CNF) *formula* γ is a finite set of n clauses. A CNF formula γ is said to be satisfiable if there is a truth assignment τ such that for every clause $c \in \gamma$ there is at least one literal $\lambda_i \in c$ such that we also have $\lambda_i \in \tau$.

Henceforth, m will denote the number of literals and n will denote the number of clauses in a CNF formula.

3. The preimage problems and limit languages

The CA Preimage (CAP) problem is stated formally as follows:

CAP:
Given: A fixed CA (Q, δ) and a configuration substring s.
Question: Is there a configuration substring $s^{(0)}$ such that $s^{(0)} \vdash^K s$ where $K = |s|$?

A more general version of this problem has been mentioned in [3,5,6] and will be referred to as "Generalized CA Preimage" (GCAP). As input to GCAP, we are also given a number T, and the question is altered in that we ask if there is an $s^{(0)}$ such that $s^{(0)} \vdash^T s$. The only difference is that the number of time steps T is independent of the size of the string $K = |s|$. It is argued in [3] that GCAP $\in$ NP, using the following line of reasoning: Only the T cells to the right of $s^{(0)}$ and T cells to the left of $s^{(0)}$ can affect s, because of the finite rate of propagation. Therefore, if we are given a subconfiguration (of length $K + 2T$) including those extra $2T$ cells in the "environment" of $s^{(0)}$, we can verify that it leads to $s^{(T)} \equiv s$ by running our CA for T time steps (see figure 1 and the proof below). Unfortunately, this is not a polynomial time algorithm. Any reasonable encoding [4] for the input T is $O(\log(T))$, so the run time for the verification algorithm ($O(T^2)$ on a sequential machine) is exponential in this input. Of course, if T had a unary encoding, the algorithm would be polynomial, and is therefore pseu-

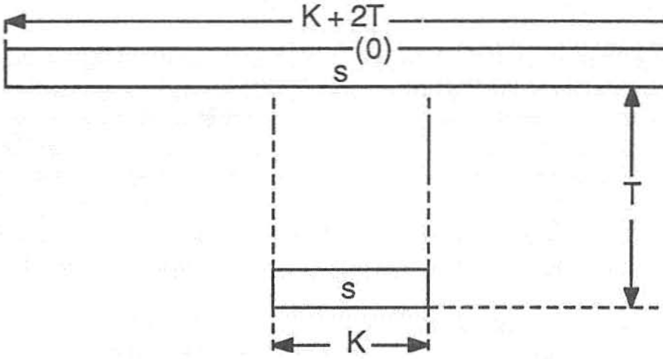

Figure 1: Illustration of the fact that only the string $s^{(0)}$, of length $K + 2T$, can affect s.

dopolynomial. Hence, it is not at all clear that GCAP is in NP. However, if we restrict T to be K, there is then no numerical input to the problem, and the verification turns out to be polynomial ($O(K^2)$) in the length of the input. Therefore, we have

Lemma 3.1. *CAP* $\in$ *NP.*

Proof: We must show that if we are given an $s^{(0)}$ such that $s^{(0)} \vdash^K s$, where $K = |s|$, we could verify that $s^{(0)} \vdash^K s$ in a time polynomial in K, which is the size of the input s. Let us write, as usual, $C^{(0)}$ for the configuration containing $s^{(0)}$, and $C^{(K)}$ for the configuration containing s. Then $C^{(0)} \vdash^K C^{(K)}$. Suppose, without loss of generality, that $C^{(0)}(i) = s_i^{(0)}$ and $C^{(K)}(i) = s_i$ for $i = 1, 2, \ldots K$. Clearly, any state $C^{(1)}(i)$ for $1 \leq i \leq K$ is independent of any state $C^{(0)}(j)$ for $j < 0$ or $j > K + 1$. By induction, we can conclude that any state $C^{(K)}(i)$ for $1 \leq i \leq K$ is independent of any state $C^{(0)}(j)$ for $j < -K+1$ or $j > 2K$. Therefore, in verifying $s^{(0)} \vdash^K s$, we need only compute with a *finite* CA using the rule δ and starting with the initial configuration $C^{(0)}(j)$ for $-K+1 \leq j \leq 2K$. Since the problem takes a fixed CA as input, each computation of a new cell state, e.g. $\delta(q_1, q_2, q_3)$, takes $O(1)$ time. This has to be done for $3K$ cells and K time steps, and hence the time expended is $O(K^2)$. ∎

We now describe a CA for which we will find that CAP is NP-complete. It should be re-emphasized that this is not part of the reduction 3SAT $\propto$ CAP, and once chosen, the CA will remain fixed. The CA (Q, δ) has states $Q = Q_1 \times Q_2$ where

$$Q_1 = \{\_, \#, l, \bar{l}, 0, 1\}$$

$$Q_2 = \{\_, \#, l, \bar{l}, 0, 1, l^s, \bar{l}^s, l^u, \bar{l}^u, 0^s, 1^s, 0^u, 1^u\}$$

and δ is given in table 1. (Note only rules 1 through 11 apply for the states we consider here, and rules 12 through 16 should be ignored for the remainder of this section.) The functions Π, Σ, and U are defined below. We will refer to the underscore "_" as the "quiescent state." The significance of the s- and u-superscripts as well as the rules will become clear when we describe how 3SAT $\propto$ CAP. Note that this CA has 84 states. It is quite likely that CAs with many fewer states also have NP-complete CAP problems, and it may be possible to find some of them by simulating the one discussed here.

It will be helpful to regard this CA as a CPCA defined by $(Q_1, Q_2, \delta_1, \delta_2)$ where δ_i is the Q_i-component of $\delta \equiv \delta_1 \times \delta_2$. With this interpretation, the proof strategy is as follows. The rules are designed so that all states in CA_1 glide to the left at the rate of one cell per time step. States in CA_2 will remain "stationary" although they may change by gaining or losing a superscript (e.g., l^u may become l, or 1 may become 1^s, etc). Truth assignments will be encoded as substrings of CA_1, and CNF formulas will be encoded as substrings of CA_2. As a truth assignment of CA_1 passes over a corresponding literal in CA_2, if their truth values match, the encoding of the literal in CA_2 will change so as to record the fact that that literal is satisfied. The delicate part is designing the rules and encodings so that the final string can be reached if *and only if* the CNF formula we started with is satisfiable.

The reduction 3SAT $\propto$ CAP proceeds in stages. We start by specifying the encoding scheme for literals. Recall that the alphabet for CA_1 (in which the literal encodings reside) is $Q_1 = \{\_, \#, l, \bar{l}, 1, 0\}$. The encoding for literal l_i is written as

$$e_l(l_i) = l\_b_1^{(i)}\_b_2^{(i)}\_b_3^{(i)}\_\ldots\_b_k^{(i)}$$

where $b_j^{(i)} \in \{0,1\}$ is the j^{th} bit in the binary representation of i, and $k = \lceil \log_2 m \rceil$ (not to be confused with the uppercase K which is the size of the input to CAP). Similarly, $e_l(\bar{l}_i) = \bar{l}\_b_1^{(i)}\_b_2^{(i)}\_b_3^{(i)}\_\ldots\_b_k^{(i)}.e_l(l_i)$ (or $e_l(\bar{l})$) will be referred to as a *literal encoding*. The encoding of a truth assignment τ is written $e_\tau(L, \bar{L})$:

$$e_\tau(L, \bar{L}) = \_\#e_l(\lambda_1)\_e_l(\lambda_2)\_\ldots\_e_l(\lambda_m)\#\_\_\_\_$$

where $\tau = (\lambda_1, \lambda_2, \ldots \lambda_m)$ and there are k quiescent states after the rightmost $\#$. The special case of $\tau = (l_1, l_2, \ldots l_m)$ is denoted by $e_\tau(L)$:

$$e_\tau(L) = \_\#e_l(l_1)\_e_l(l_2)\_\ldots\_e_l(l_m)\#\_\_\_\_$$

and plays an important role in our reduction. $e_\tau(L, \bar{L})$ is a *truth assignment encoding*, and $e_\tau(L)$ is referred to as the *all-true assignment*. Note that in both of them the left endmarker $\#$ has a "_" preceding it, while the right

Rule	Condition	δ_1	δ_2	Interpretation
1	$q_1 \in Q_1$ and $(q_1^R \neq \bar{l}$ or $q_2^R \neq \#)$	q_1^R		CA$_1$ states move to the left
2	$q_1 = \#$ and $q_1^L = \_$ and $q_2 \in \sigma \cup \mu$		$\Pi(q_2)$	_# in CA$_1$ makes s- and u-states in CA$_2$ pure.
3	$q_1 = \#$ and $q_1^L \in \{1,0\}$ and $q_2^R \in \sigma$		$\Sigma(q_2)$	1# or 0# in CA$_1$ propagates s-states in CA$_2$ to left.
4	$q_1^R = \bar{l}$ and $q_2^R = \#$	l		# in CA$_1$ makes $\bar{l}$ in CA$_2$ into an l
5	$q_1 \in \{l, \bar{l}\}$ and $q_1 = q_2$		$\Sigma(q_2)$	Matching l's "satisfies" the l in CA$_2$.
6	$q_2^L \in \sigma$ and $q_2 \in \{l, \bar{l}\}$		$\Sigma(q_2)$	s-superscript propagates to the right if l is met.
7	$q_2^L \in \sigma$ and $q_2 \in \{1,0\}$ and $(q_1 \in \{\_, l, \bar{l}\}$ or $q_1 = q_2)$		$\Sigma(q_2)$	s propagates to the right as long as state in CA$_1$ is not a mismatch.
8	$q_2 \in \sigma$ and $q_1^R \neq q_2^R$ and $q_1^R, q_2^R \in \{1,0\}$		$U(q_2)$	Bits do not match, causing CA$_2$ bit to be unsatisfied.
9	$q_2 \in \mu$ and $q_2^L \in \sigma$		$\Pi(q_2)$	u-states propagate left leaving p-states behind.
10	$q_2^R \in \{1^u, 0^u\}$ and $q_2 \in \sigma$		$U(q_2)$	u-states propagate left until $l(\bar{l})$ is met.
11	$q_2 \in \{l^u, \bar{l}^u\}$		$\Pi(q_2)$	Make an $l^u(\bar{l}^u)$ into a pure state.
12	$q_2^L = \$$		\$	\$ in CA$_2$ moves right
13	$q_2 = \$$ and $q_2^R \in \{\#, \_\} \cup \sigma$ and $q_2^L \neq \$$		$\Lambda(q_2^R)$	\$ makes satisfied states in CA$_2$ into left movers
14	$q_2 = \$$ and $q_2^R \notin \{\#, \_\} \cup \sigma$		_	\$ "annihilates" other states
15	$q_2^L \neq \$$ and $q_2^R \in \Lambda$		q_2^R	Λ-states move left (except past \$)
16	$q_2 \in \Lambda$ and $q_2^R \notin \Lambda$		_	Λ-states move left

Table 1: CA rules.

$$\delta((q_1^L, q_2^L), (q_1, q_2), q_1^R, q_2^R)) \equiv (\delta_1((q_1^L, q_2^L), (q_1, q_2), (q_1^R, q_2^R)), \delta_2((q_1^L, q_2^L), (q_1, q_2), (q_1^R, q_2^R)))$$

is given above for all possible values of q_i^L, q_i, and q_i^R. All rules must be tried in the order given whenever δ is computed. If two different rules apply and yield different δ's, choose the rule with the smaller number. A blank entry means that δ_i is unaffected by the given condition. (Note that only rules 1 through 11 are relevant in the CAP and CASR problems.)

endmarker does not have a "_" preceding it. This is because the left and right endmarkers in CA$_1$ play different roles.

The encodings for CNF formulas reside in CA$_2$, but they still use the alphabet Q_1. The encoding is expressed in terms of the formula's constituent literals. We encode the occurrence of literal l_i as

$$e_o(l_i) = lb_1^{(i)}b_2^{(i)}\dots b_k^{(i)}$$

(a *literal occurrence encoding*) where, again, $b_j^{(i)}$ is the j^{th} bit in the binary representation of i. (The only difference with $e_l(l_i)$ is that there are no intervening _'s). Suppose clause γ_i contains the literals λ_1, λ_2, and λ_3. Then we encode the clause as the string $e_c(\gamma_i) = e_o(\lambda_1)e_o(\lambda_2)e_o(\lambda_3)$. Finally, if the CNF formula $\gamma = \{\gamma_1, \gamma_2, \dots, \gamma_n\}$, we encode it as the string, $e_f(\gamma) = \#\_e_c(\gamma_1)\_e_c(\gamma_2)\_e_c(\gamma_3)\_\dots\_e_c(\gamma_n)$.

It is convenient to define $\pi \equiv \{l, \bar{l}, 0, 1\}, \sigma \equiv \{l^s, \bar{l}^s, 0^s, 1^s\}$, and $\mu \equiv \{l^u, \bar{l}^u, 0^u, 1^u\}$, so that $Q_2 = \{\#, \_\} \cup \pi \cup \sigma \cup \mu$. The states in π are called *pure states* (or p-states), a state in σ is called a *satisfied state* (s-state), and a state in μ is called an *unsatisfied state* (u-state). It is also convenient to define functions which take a state from one of these sets to one of the others by adding, eliminating, or changing a superscript. In particular, define the three functions Π, Σ, and U as follows:

$$\Pi : Q_2 \to \pi \text{ is defined so that } \Pi(\beta^x) = \beta \text{ for any } \beta^x \in \pi \cup \mu \cup \sigma,$$

$$\Sigma : Q_2 \to \sigma \text{ is defined so that } \Sigma(\beta^x) = \beta^s \text{ for any } \beta^x \in \pi \cup \mu \cup \sigma,$$

$$\mathrm{U} : Q_2 \to \mu \text{ is defined so that } \mathrm{U}(\beta^x) = \beta^u \text{ for any } \beta^x \equiv \pi \cup \mu \cup \sigma,$$

where x represents an s, u, or no superscript. Π, Σ, and U have no effect on either $\#$ or _ . We extend them to morphisms over Q_2^*: For any string $s = s_1, s_2 \dots s_k$ in Q_2^*, let $\Pi(s) = \Pi(s_1)\Pi(s_2)\dots\Pi(s_k), \Sigma(s) = \Sigma(s_1)\Sigma(s_2)\dots\Sigma(s_k)$, and $\mathrm{U}(s) = \mathrm{U}(s_1)\mathrm{U}(s_2)\dots\mathrm{U}(s_k)$.

We are finally in a position to describe the instance of CAP to which an instance of 3SAT gets transformed. Let 3SAT be specified by the CNF formula γ, and the sets of literals L and $\bar{L}$. The instance of CAP, described with the CA that has been specified, is then given by the configuration substring,

$$S \equiv \left. e_r(L) \right| \overline{\Sigma(e_f(\gamma))} \tag{3.1}$$

Note:

1. All the cells of S in CA$_1$ are quiescent if they correspond to cells in $\Sigma(e_f(\gamma))$.

2. The cells in CA$_2$ corresponding to the cells in $e_r(L)$ are left unspecified, and their values are irrelevant to the argument.

3. The leftmost state of $\Sigma(e_f(\gamma))$ is in the cell immediately to the right of the cell containing the rightmost state of $e_\tau(L)$.

The main result of this section, which establishes the NP-completeness of CAP, is that S has a K^{th}-preimage (where $K = |e_\tau(L)| + |\Sigma(e_f(\gamma))| = |S|$) if and only if γ is satisfiable. We must, of course, first establish that the reduction 3SAT $\propto$ CAP can be done in polynomial time. But this is obvious. The instance of CAP, represented by S, is simply a reasonable encoding scheme for 3SAT. S consists of a concise listing of the literals, $e_\tau(L)$, and of the formula, $\Sigma(e_f(\gamma))$ (from which $e_f(\gamma)$ can easily be recovered in $O(n \cdot \log(m))$ time by applying Π to each element of the string, although this is not really necessary). To the extent that reasonable encoding schemes are related by polynomial time transformations, we have

Lemma 3.2. *The reduction 3SAT $\propto$ CAP as described above can be executed in a time polynomial in m and n.*

The following three lemmas comprise the rest of the proof that CAP is NP-complete.

Lemma 3.3. *If S, given by equation (3.1), has a K^{th}-preimage specified by some configuration $C^{(0)}$, then there must be at least one such $C^{(0)}$ which contains the string,*

$$S^{(0)} \equiv \begin{array}{c|l} & e_{\tau'}(L,\bar{L}) \\ \cline{1-1} e_f(\gamma) & \end{array} \tag{3.2}$$

where τ' is some truth assignment, $e_f(\gamma)$ occupies the same cells as the $\Sigma(e_f(\gamma))$ of S, and $K = |e_f(\gamma)| + |e_\tau(L,\bar{L})|$.

Proof: Rule 1 in table 1 says that all states in CA_1 propagate to the left at the rate of one cell per time step. The only other way in which states in CA_1 are altered is via the left endmarker, #, in CA_2 (rule 4). The sole effect of the # in CA_2 on states in CA_1 is to change $\bar{l}$'s to l's. Therefore, the $e_\tau(L)$ of S must have originated as a similar string in $C^{(0)}$, shifted over K cells to the right. The only possible difference is that it may have started as some other truth assignment $e_{\tau'}(L,\bar{L})$. This is true provided the endmarker in CA_2 remains unchanged. But according to rules 1 through 11 in table 1, the only way a state in CA_2 can change is for some state in π, σ, or μ to change (or gain or lose) a superscript. Therefore, the only possible preimage of any state $\xi \in \Sigma(e_f(\gamma))$ is $\xi, \Pi(\xi)$, or $\text{U}(\xi)$. Since # cannot have a superscript, its only possible preimage in CA_2 is itself. Furthermore, we see that the only difference between $\Sigma(e_f(\gamma))$ and its K^{th}-preimage is that some of its elements may have originally been p-states or u-states instead of s-states. Note, however, that rule 2 dictates that the leftmost # of $e_{\tau'}(L,\bar{L})$ causes all s-states or u-states in CA_2 to become p-states as it passes them by. This occurs for any state ξ' in the K^{th}-preimage of $\Sigma(e_f(\gamma))$ before any state in

CA_1 to the right of the # in $e_{\tau'}(L, \bar{L})$ has had a chance to affect ξ'. Hence, we may assume without loss of generality that all of the K^{th}-preimages of states in $\Sigma(e_f(\gamma))$ can start out as pure states, i.e., that the K^{th}-preimage of $\Sigma(e_f(\gamma))$, if it exists, is $e_f(\gamma)$. ∎

Lemma 3.4. *Let S be given by equation (3.1), and suppose the γ from which we constructed S is satisfiable. Then S has a K^{th}-preimage, where $K = |S|$.*

Proof: Suppose γ is satisfiable but that S does not have a K^{th}-preimage. We will contradict this by constructing a K^{th}-preimage from the fact that γ is satisfiable. In fact, let $C^{(0)}$ be a configuration which contains the string $S^{(0)}$ given in equation (3.2), where τ' is a satisfying truth assignment of γ. It will now be demonstrated that $C^{(0)} \vdash^K S$, thus proving the lemma.

The rules are such that the string $e_l(l_i)$, as it passes by the string $e_o(l_i)$, causes all states in $e_o(l_i)$ to become satisfied states. More precisely, if $e_l(l_i)$ is a substring of $e_{\tau'}(L, \bar{L})$, then $e_o(l_i) \vdash^K \Sigma(e_o(l_i))$. To see this, first observe that

$$\begin{array}{lcl} e_l(l_i)\_ & \vdash^j & e_l(l_i)\_ \\ e_o(l_i) & & \Sigma(e_o(l_i)) \end{array}$$

where $j = |e_o(l_i)|$. Initially, in the above computation, the leftmost states of $e_l(l_i)$ and $e_o(l_i)$ are in the same cell, and after j steps the quiescent state $\_$ (which trails $e_l(l_i)$) and the *rightmost* state of $\Sigma(e_o(l_i))$ are in the same cell. This can be proven by induction on the number of time steps, using rules 1, 5, and 7. Intuitively, the rules work in this case by propagating s-states to the right as long as the characters in $e_l(l_i)$ and $e_o(l_i)$ match. It is best to give an illustration at this point, for $e_l(l_2)$ and $e_o(l_2)$, in the case of $m = 2$:

$$\begin{array}{lllllll} l\_1\_0\_ & & l\_1\_0\_ & & l\_1\_0\_ & & l\_1\_0\_ \\ & \vdash & & \vdash & & \vdash & \\ l10 & & l^s10 & & l^s1^s0 & & l^s1^s0^s \end{array}$$

In order to establish that $e_o(l_i) \vdash^K \Sigma(e_o(l_i))$, we must still show that no states to the right of $e_l(l_i)$ subsequently change the s-states in $\Sigma(e_o(l_i))$. The only rules in which an s-state can be altered are 2, 8, and 10. However, rule 2 only applies for the *left* endmarker of $e_{\tau'}(L, \bar{L})$, which has already bypassed $e_o(l_i)$. The right endmarker (also #) of $e_{\tau'}(L, \bar{L})$ has a non-quiescent state immediately to its left, so that rule 2 does not apply for it. Rule 10 is invoked only if the state immediately to the right of a CA_2 cell is 1^u or 0^u. However, once $\Sigma(e_o(l_i))$ is generated, this cannot be true for any element of $\Sigma(e_o(l_i))$. This is because the state immediately to the right of $\Sigma(e_o(l_i))$ cannot be in $\{1^u, 0^u\}$, so that by induction rule 10 leaves $\Sigma(e_o(l_i))$ intact. Rule 8 is invoked for a given cell only if the cell immediately to its right contains a 1 or 0 in CA_2. However, we can argue as we did for rule 10 that rule 8 will not apply to any state in $\Sigma(e_o(l_i))$ once it is generated. Hence, we have that $e_o(l_i) \vdash^K \Sigma(e_o(l_i))$ if $e_l(l_i)$ is a substring of $e_{\tau'}(L,\bar{L})$. The same is true of negated literals, i.e., if $e_l(\bar{l}_i)$ is a substring of $e_{\tau'}(L,\bar{L})$, then $e_o(\bar{l}_i) \vdash^K \Sigma(e_o(\bar{l}_i))$. In other words, if a literal is "true" according to the truth assignment, its encoding in the formula is made "true" by an application of Σ.

Now that we have shown that true literals get "satisfied", we must show that a clause gets satisfied if it contains a true literal. That is, it is necessary to show that if $e_o(l_i)$ is a substring of $e_c(\gamma_j)$ and if $e_l(l_i)$ is a substring of $e_{\tau'}(L,\bar{L})$, then $e_c(\gamma_j) \vdash^K \Sigma(e_c(\gamma_j))$. This follows from the preceding argument and rules 3, 6, and 7. By the preceding argument, $e_o(l_i) \vdash^K \Sigma(e_o(l_i))$. We consider the fate of the clause string containing $\Sigma(e_o(l_i))$ after the *right* endmarker of $e_{\tau'}(L,\bar{L})$ has reached the leftmost state of $\Sigma(e_o(l_i))$. Since the right # of $e_{\tau'}(L,\bar{L})$ always has a 1 or 0 to its left, rule 3 applies and any state in CA_2 in the same cell as the # becomes an s-state if there is an s-state to its right. Hence, the s-states propagate to the left (along with the # in CA_1) until a quiescent or # state is reached in CA_2. $\Sigma(e_o(l_i))$ similarly infects any pure states to its right in the same clause string, by rules 6 and 7 (recall that there are at least $|e_f(\gamma)| + k$ quiescent states to the right of the rightmost endmarker in $e_{\tau'}(L,\bar{L})$, so that s-states will be allowed to propagate to the right).

The same argument holds for negated literals: if $e_c(\gamma_j)$ contains $e_o(\bar{l}_i)$ and $e_\tau(L,\bar{L})$ contains $e_l(\bar{l}_i)$, then $e_c(\gamma_j) \vdash^K \Sigma(e_c(\gamma_j))$. But since τ' is a satisfying truth assignment, each $e_c(\gamma_j)$ contains a $e_o(l_i)$ (resp. $e_o(\bar{l}_i)$) such that $e_l(l_i)$ (resp. $e_l(\bar{l}_i)$) is in $e_{\tau'}(L,\bar{L})$. Thus, $e_c(\gamma_j) \vdash^K \Sigma(e_c(\gamma_j))$ for all $j \in \{1,2,\dots n\}$, so that $e_f(\gamma) \vdash^K \Sigma(e_f(\gamma))$.

Finally, note that as the string $e_{\tau'}(L,\bar{L})$ passes the left endmarker # of CA_2, each $\bar{l}$ becomes an l by rule 4. Thus, by time K, $e_{\tau'}(L,\bar{L})$ has changed into the all-true assignment $e_\tau(L)$. But, this establishes lemma 3.4 since we have shown that $C^{(0)} \vdash^K S$. ∎

Lemma 3.5. *Suppose S, given by equation (3.1), has a K^{th}-preimage, where $K = |S|$. The γ from which we constructed S is then satisfiable.*

Proof: By the proof of lemma 3.3, the only possible K^{th}-preimages $C^{(0)}$ of S must contain an $S^{(0)}$ of the form given in equation (3.2), or can be obtained from (3.2) by changing any number of the pure states in $e_f(\gamma)$ to s-states or u-states. However, the left endmarker of $e_{\tau'}(L,\bar{L})$ makes these states pure, so any initial s- or u-states in $S^{(0)}$ are irrelevant in determining S. Hence, we can exhaust all candidate K^{th}-preimages of S by considering only those $C^{(0)}$'s containing $S^{(0)}$ as given in equation (3.2).

Now suppose γ is not satisfiable. Then there is no τ' which satisfies γ. Consider all possible strings $S^{(0)}$ given by equation (3.2), which differ only in the choice of τ'. We will demonstrate that for no such $S^{(0)}$ contained in $C^{(0)}$ do we have $C^{(0)} \vdash^K S$, contradicting the assumption of lemma 3.5.

Since no τ' satisfies γ, for any τ' there is at least one clause γ_j such that no literal in γ_j is also in τ'. For a given τ' consider such a γ_j. It is now argued that $e_c(\gamma_j) \vdash^K e_c(\gamma_j)$. For this, it is essential to show first that for any $i \neq i'$,

$$e_l(l_i)\_e_l(l_{i+1}) \vdash^K e_l(l_i)|\_e_l(l_{i+1}) \qquad\qquad e_o(l_{i'})|e_o(l_{i'}) \tag{3.3}$$

where $k = |e_l(l_i)|$. This makes use of rules 1, 5, 7, 8, 9, 10, and 11. Before demonstrating this, it helps to give an example:

$$\begin{array}{ccccccc} l\_1\_0\_ & & l\_1\_0\_ & & l\_1\_0\_ & & l\_1\_0\_ \\ & \vdash & & \vdash & & \vdash & \\ l11 & & l^s11 & & l^s1^s1 & & l^s1^u1 \end{array}$$

$$\begin{array}{ccccc} & l\_1\_0\_ & & l\_1\_0\_ & \\ \vdash & & \vdash & & \\ & l^u11 & & l11 & \end{array}$$

In the above computation, the second substring results from rule 5, the third from rule 7, the fourth from rule 8, the fifth from rules 9 and 10, and the sixth from rule 11. (The states to the right of $l\_1\_0\_$ cannot affect the resulting $l11$ in CA$_2$ until the next time step.)

The only way of producing an s-state if there are no neighboring s-states is through rule 5. (We will find, by induction, that after any computation of the form (3.3), there will be no neighboring s-states.) Once an s-state is created in $e_o(l_{i'})$, it propagates to the right (by rule 7) *as long as the binary expansions of i' and i are the same (i.e., they "match")*. That is, starting with

$$l\_b_1^{(i)}\_b_2^{(i)}\_\ldots\_b_k^{(i)}$$

$$lb_1^{(i')}b_2^{(i')}\ldots b_k^{(i')}$$

we find

$$lb_1^{(i')}b_2^{(i')}\ldots b_k^{(i')} \vdash^{p+1} l^s b_1^{i')s} b_2^{(i')s} \ldots b_p^{(i')s} b_{p+1}^{(i')} \ldots b_k^{(i')}$$

if and only if $b_r^{(i)} = b_r^{(i')}$ for all r from 1 through p. However, since $i \neq i'$, there is an r such that $b_r^{(i)} \neq b_r^{(i')}$. Then, by rule 8, $b_{r-1}^{(i)s}$ becomes $\mathrm{U}(b_{r-1}^{(i)s}) = b_{r-1}^{(i)u}$. Thus,

$$lb_1^{(i')}b_2^{(i')}\ldots b_k^{(i')} \vdash^{r+1} l^s b_1^{(i')s} b_2^{(i')s} \ldots b_{r-1}^{(i')u} b_r^{(i')} \ldots b_k^{(i')}$$

and the s-states cease propagating to the right. Furthermore, by rules 9 and 10, u-states propagate to the left up to the l, which then becomes a p-state by rule 11:

$$lb_1^{(i')}b_2^{(i')}\ldots b_k^{(i')} \vdash^{2r+1} lb_1^{(i')}b_2^{(i')}\ldots b_k^{(i')}.$$

Since $|e_l(l_i)| = 2k+1 \geq 2r+1$, this occurs before the rightmost element of $e_l(l_i)$ has passed the cell to the left of $e_o(l_{i'})$. Hence, in the time during which $e_l(l_i)$ scans past $e_o(l_{i'})$, no states to the right of $e_l(l_i)$ have a chance to create any new s-states, and $e_o(l_{i'})$ returns to its original form. Since no new s-states are created, we conclude that the next literal assignment string, $e_l(l_{i+1})$, or endmarker #, that bypasses $e_o(l_{i'})$ cannot create new s-states except via rule 5. Thus, when (or if) the matching process starts again, there will be no s-states in the neighborhood of the leftmost l of $e_o(l_{i'})$. By induction, this continues to be the case until time K.

Since none of the literal strings in $e_c(\gamma_j)$ are ever completely satisfied, all s-states in it are erased and we find $e_c(\gamma_j) \vdash^K e_c(\gamma_j)$ as claimed.

We have shown that for any τ', there is at least one $\gamma_j \in \gamma$ such that $e_c(\gamma_j) \vdash^K e_c(\gamma_j)$. But this means that the substring in $C^{(K)}$ corresponding to $e_f(\gamma)$ cannot be $\Sigma(e_f(\gamma))$ for any τ', i.e., that S cannot be reached in K time steps for any τ'. This contradicts our assumption that S has a K^{th}-preimage, so γ must be satisfiable. ∎

From lemmas 3.4 and 3.5, we conclude that S has a K^{th}-preimage if and only if γ is satisfiable. From this and lemmas 3.1 and 3.2, we conclude

Theorem 3.1. *CAP, for the CA described in this section, is NP-complete.*

In addition, since CAP is a special case of GCAP described at the beginning of this section, we immediately have

Corollary 3.1. *GCAP, for the CA described in this section, is NP-hard.*

Further note that the proof of theorem 3.1 can be applied to a *finite* CA whose size is equal to the size K of the input string S. Because of the periodic boundary conditions of a finite CA, the $e_{\tau'}(L, \bar{L})$ which is in the initial configuration that computes S occupies the same cells as the final $e_\tau(L), K$ time steps later. Hence, we have

Corollary 3.2. *CAP, defined for a finite CA in which the input S comprises an entire configuration, is NP-complete for the CA defined in this section.*

We next turn our attention to the CA Subconfiguration Recurrence (CASR) problem, stated formally as follows:

CASR:
Given: A fixed CA (Q, δ) and a configuration substring s.
Question: Is it possible to have $s \vdash^K s$, where $K = |s|$?

The proof of theorem 3.1 can be used almost without change to prove the following.

Theorem 3.2. *CASR is NP-complete.*

Proof: If we are given some initial configuration $C^{(0)}$ containing s, we can easily check if $C^{(0)} \vdash^K s$ using only the K cells to the left and the K cells to the right of s. As in lemma 3.1, this requires $O(K^2)$ time, so CASR $\in$ NP.

We now show that 3SAT $\propto$ CASR using the CA introduced in this section. Simply set $s = S$, where S is given by equation (3.1). The reduction is clearly polynomial time in m and n.

Now, if the γ from which S is constructed is satisfiable, then $S \vdash^K S$. For suppose that γ is satisfiable but it is not true that $S \vdash^K S$ for any choice of $C^{(0)}$ which contains S. We contradict this (as in lemma 3.4) by constructing a $C^{(0)}$ such that $C^{(0)} \vdash^K S$. Choose $C^{(0)}$ to contain the string,

$$e_\tau(L) \left| \, \overline{\Sigma(e_f(\gamma))} \, \right| e_{\tau'}(L, \bar{L}) \tag{3.4}$$

where τ' is a satisfying truth assignment. Then the $e_{\tau'}(L)$ on the left moves to the left and has no effect on the string $\Sigma(e_f(\gamma))$. Furthermore, the s-states in $\Sigma(e_f(\gamma))$ are erased by the left endmarker of $e_{\tau'}(L, \bar{L})$ and do not survive until time K. However, because τ' is a satisfying truth assignment, we nevertheless have, by the proof of lemma 3.4, that $\Sigma(e_f(\gamma)) \vdash^K \Sigma(e_f(\gamma))$, so that $C^{(0)} \vdash^K S$.

Now suppose $S \vdash^K S$. Then γ is satisfiable. For suppose it isn't. Then no τ' is a satisfying truth assignment. But by the proof of lemma 3.3, we lose no generality by assuming that the only way we can end up with the string S in K steps is by starting out with a substring of the form given in (3.4) for some τ'. But since for any τ' there is at least one clause which is not satisfied, we do not have $\Sigma(e_f(\gamma)) \vdash^K \Sigma(e_f(\gamma))$ for any such τ', by the proof of lemma 3.5. Therefore, we cannot have $S \vdash^K S$, a contradiction. Thus, γ is satisfiable if and only if $S \vdash^K S$. ∎

We conclude this section with the observation that the limit language generated by the CA defined here is NP-hard. Before doing this, we must specify what is meant by the language generated by a CA. Let us denote the set of all possible configurations by Ω. Then, the set of configurations generated by CA (Q, δ) in t time steps is defined by $\delta^t(\Omega) \equiv \{C^{(t)} | C^{(t)} = \delta^t(C), C \in \Omega\}$. The *language generated in t steps* by CA (Q, δ) is defined as

$$\Omega^t \equiv \cap_{i=0}^{t} \delta^i(\Omega)$$

and the limit language is $\Omega^\infty \equiv \lim_{t\to\infty} \Omega^t$. We also find it convenient to define the set of *generable substrings in t time steps*, which is the set of substrings of configurations in Ω^t : $\Gamma^t \equiv \{s | s \text{ is a substring of some } C \in \Omega^t\}$. Γ^t is thus the set of strings which have p^{th}-preimages for all $p \leq t$. The *limit substring language* generated by a CA is $\Gamma^\infty \equiv \lim_{t\to\infty} \Gamma^t$. It has been shown [5] that Ω^t (and also Γ^t), for any fixed t, is a regular language. Empirical results indicate that for many CAs the complexity of the regular languages Ω^t, as measured by the minimal finite automata required to recognize them, increase with t. This suggests that the limit languages, while they can be regular, are more often not regular languages. In fact, CAs have been constructed in which the limit languages are strictly non-regular, non-context-free, and non-recursively enumerable [7]. In this paper, we have an example of a CA for which the limit language is NP-hard.

Theorem 3.3. *Ω^∞ is NP-hard.*

Proof: Consider any finite substring s. We can make it into a configuration by concatenating an infinite number of quiescent states to its left and right. Since any substring can be so extended, it follows that $\Gamma^t \propto \Omega^t$ for any t. Therefore, $\Gamma^\infty \propto \Omega^\infty$. We claim that Γ^∞ is an NP-hard language. Since Ω^∞ therefore is also NP-hard, the theorem will be proved.

By lemmas 3.4 and 3.5, Γ^t contains strings of the form S given in equation (3.1), where $|S| = t$, if and only if the γ from which S is constructed is satisfiable. But Γ^t also contains S's of the form (3.1) where $|S| < t$ if and only if γ is satisfiable. The latter is easy to see, since S has a t^{th}-preimage if and only if it also has a p^{th}-preimage for any $p > t$. Such a preimage can be constructed from the t^{th}-preimage of S (which is of the form given in equation (3.2)) by displacing the $e_{\tau'}(L, \bar{L})$ in its t^{th}-preimage $p - t$ cells to the right. By induction we can then conclude that Γ^∞ contains any S if and only if γ is satisfiable. That is, 3SAT $\propto \Gamma^\infty$, so that Γ^∞ is NP-hard. ∎

4. The temporal sequence problem

A *temporal sequence* is a sequence of states in one given cell. We will use the same notational conventions for temporal sequences as for substrings. A temporal sequence $W = W_1 W_2 \ldots W_T$ is said to be computed by a configuration $C^{(0)}$ in T time steps if and only if there is some cell i such that $C^{(1)}(i) = W_1, C^{(2)}(i) = W_2, \ldots C^{(T)}(i) = W_T$, where $C^{(0)} \vdash C^{(1)} \vdash C^{(2)} \ldots \vdash C^{(T)}$. If there is a $C^{(0)}$ which computes W in K time steps, $C^{(0)}$ is said to be a K^{th}-preimage of W. As with substrings, we need only consider a substring $S^{(0)}$ of $C^{(0)}$ in order to determine if $C^{(0)}$ computes some temporal sequence W. The CA Temporal Sequence Preimage problem is stated formally as follows (resisting the temptation to refer to it as "TSP", it is abbreviated as "CATS").

CATS:
Given: A fixed CA (Q, δ) and a temporal sequence w.
Question: Is there a configuration substring $s^{(0)}$ which computes w in K time steps, where $K = |w|$?

The proof of the following sufficiently resembles the proof of lemma 3.1 that it is omitted.

Lemma 4.1. *CATS $\in$ NP.*

The CA for which CATS is NP-complete is an extension of the CA used in section 3. The set of states Q_1 is the same as before, but now we take

$$Q_2 = \{\_, \#, \$\} \cup \pi \cup \mu \cup \sigma \cup \Lambda$$

where π, μ, and σ are defined as in section 3, and

$$\Lambda = \{l^L, \bar{l}^L, 1^L, 0^L\}$$

The additional rules needed are given by rules 12 through 16 of table 1. Analogously to Π, Σ and U we define a function Λ: $Q_2 \to \Lambda$ so that $\Lambda(\beta^s) = \beta^L$ for any $\beta \in \pi$ and $\Lambda(\beta) = \beta$ for $\beta \in \{\#, \_\}$. Its domain is similarly extended to Q_2^*. Note that there are now 114 states.

Again, it is helpful to think in terms of a CPCA. The additional rules allow states in CA_2 to propagate. States in Λ propagate to the left, and the one state that propagates to the right in CA_2 is \$. The proof strategy is basically to use the proof of theorem 3.1 to produce a substring S which has an $|S|^{\text{th}}$-preimage if and only if the formula from which it is constructed is satisfiable. Each state in the part of S representing literals $(e_r(L))$ passes through a particular cell that is used to contain the temporal sequence W. The purpose of the \$ is to dig into the part of S representing the satisfied formula $\Sigma(e_f(\gamma))$ and move all of the states in it to the left, thus "projecting" it along the time axis, in the same cell that the sequence $e_r(L)$ passed through (see figure 2).

Once again, we reduce 3SAT to CATS. Literals are encoded exactly as before (although now, they must be understood to be sequences). We define a new formula encoding as follows: $e_f^L(\gamma)$ is the same as $\Lambda(e_f(\gamma))$, except that there is one extra "_" preceding each non-quiescent symbol in $\Lambda(e_f(\gamma))$. For example, if $e_f(\gamma)$ starts out as $\#\_l10\bar{l}11l00\_l11\ldots$, $e_f^L(\gamma)$ starts out as $\#\_\_l^L\_1^L\_0^L\_\bar{l}^L\_1^L\_1^L\_l^L\_0^L\_0^L\_\_l^L\_1^L\_1^L\ldots$. From any instance of 3SAT, we write the sequence W as

$$W = \begin{array}{l} \rule{1.5em}{0.4pt}\,e_r(L)\,\rule{1.5em}{0.4pt} \\ \qquad\quad \$e_f^L(\gamma) \end{array}$$

where, defining $N = |e_f(L)|$ and $M = |e_r(L)|$ (so $|W| = 3N + M$), there are N quiescent states before the $e_r(L)$ and $2N$ quiescent states after it. As in the case of CAP, this is trivially a polynomial time transformation. Without loss of generality, assume the sequence W occurs in cell 0. Thus, the last symbol in $e_r(L)$ and the \$ both appear in cell 0 at time $N + M$. Also, note that any states that appear in cell 0 in CA_2 before time $N + M$ are irrelevant to the argument.

The main result which allows us to use the proof of theorem 3.1 is as follows.

Lemma 4.2. *W in cell 0 has a $|W|^{\text{th}}$-preimage if and only if*

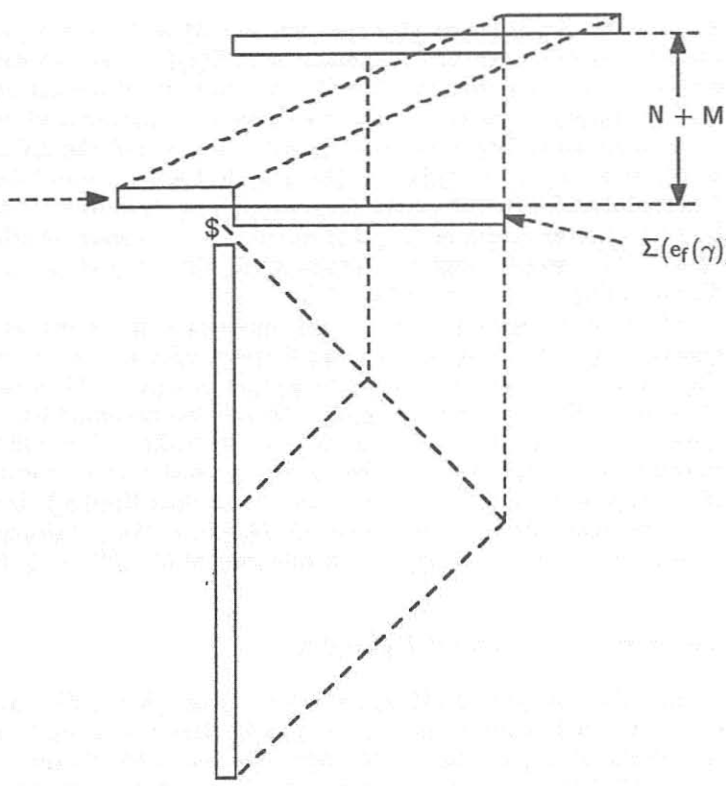

Figure 2: Illustration for theorem 4.1.

$$S = \begin{array}{l} \_\_\_\_e_\tau(L)\_\_\_\_ \\ \qquad\quad \$e_f^L(\gamma) \end{array}$$

is a substring at time $N+M$, where as in W, there are N quiescent states before the $e_\tau(L)$ and $2N$ quiescent states after it, and the \$ in S is in cell 0.

Proof: Suppose S is a substring at time $N+M$ with the \$ in cell 0. Since the $e_\tau(L)$ is to the left of the endmarker in $\Sigma(e_f(\gamma))$, we can use rule 1 in reverse to conclude that the first $N+M$ states in W are correctly given. Now by rules 12 and 13 the \$ in CA$_2$ moves to the right, changing each state in $\Sigma(e_f(\gamma))$ into a left-moving state. Because of the $2N$ _'s trailing $e_\tau(L)$, no states to the right of $e_\tau(L)$ can affect any state in CA$_2$ until the \$ has passed the N cells containing $\Sigma(e_f(\gamma))$ at time $N+M$. Therefore, by rule 13, every state in $\Sigma(e_f(L))$ becomes a left mover, which, by rules 15 and 16, passes through cell 0 every other time step after time $N+M$. The resulting sequence in cell 0 is W.

Conversely, suppose W has a $|W|^{\text{th}}$-preimage and is the temporal sequence in cell 0. We can now work forward with rule 1 to deduce that the first $N+M$ states of S must be as they are given. Then suppose the remaining $2N$ states are not as given. This obviously cannot be the case for the states in CA$_1$, since by rule 1 any other state than "_" would propagate to cell 0 before time $3N+M$, but in fact no such states appear in cell 0. Now suppose in CA$_2$ we have something other than $\Sigma(e_f(\gamma))$. If any of the states are not s-states, then by rules 12, 14, and 15 these states never make it to cell 0. Therefore, S must be a substring of $C^{(N+M)}$, with its \$ in cell 0. ∎

Theorem 4.1. *CATS is NP-complete.*

Proof: From lemma 4.2, W has a K^{th}-preimage ($K = |W| = 3N+M$) if and only if S is a substring at time $N+M$. But S is a substring at time $N+M$ if and only if it has an $N+M^{\text{th}}$-preimage, which in turn is true (by the proof of theorem 3.1) if and only if γ is satisfiable. Therefore, W has a K^{th}-preimage if and only if γ is satisfiable. This plus lemma 4.1 proves the theorem. ∎

5. Discussion

We have seen that it is possible in principle to construct a CA for which the preimage, recurrence, and temporal sequence problems are NP-complete, and for which the limit language is NP-hard, but the CA that is constructed is quite complicated. It is, however, quite likely that much simpler CAs can be constructed with similar properties. What has been accomplished here is a starting point for further investigations of the questions raised in the introduction. Now that at least one CA is known with these properties, it should be easier to find others.

An interesting new question arises regarding the increase in complexity as a function of time that was observed in many CAs in [5]. In particular, consider the sequence of languages Γ^t whose limit forms the limit language Γ^∞. Because Γ^∞ is NP-hard, the computational complexity of Γ^t increases as a superpolynomial function of t provided (as is universally believed) $P \neq NP$. But it was noted in [5] that Γ^t is a regular language for any fixed t. Furthermore, it was conjectured in [5] that the regular language complexity of Γ^t, as measured by the minimum number of states required to recognize it in a deterministic finite automaton, increases as a function of t for a wide class of CAs. In particular, for CAs whose behavior is sufficiently complex (what are referred to in reference [5] as class 3 and 4 CAs), the regular language complexity seems to grow rapidly, perhaps at a superpolynomial rate. Now much more is understood about the computational complexity of languages (assuming the truth of the $P \neq NP$ conjecture) than about regular language complexity in cellular automata. Therefore, if computational complexity could tell us something about regular language complexity, it would be easier to answer another question raised in [3], namely: is regular language complexity generically increasing in cellular automata? One might be able to conclude, for example, that in any CA with an NP-hard limit language, regular language complexity increases superpolynomially. However, it is not at all clear that there is any such relation. The problem is reminiscent of the subtle and deep relationship between computational complexity and circuit complexity (e.g., [11]). This prompts the following new question: is there any relation between the computational complexity of languages generated by CAs in finite time and their regular language complexity?

Acknowledgements

The work in this paper was largely inspired by the CA '86 conference, and I would like to thank the organizers for a stimulating conference. I also wish to thank D. Joyce and S. M. Selkow for helpful discussions.

References

[1] S. Wolfram, "Approaches to complexity engineering", *Physica* **D**, **22** (1986) 385–399.

[2] S. Wolfram, ed., *Theory and Applications of Cellular Automata*, (World Scientific, Singapore, 1986).

[3] S. Wolfram, "Twenty problems in the theory of cellular automata", *Physica Scripta*, **T9** (1985) 170.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability*, (W.H. Freeman, New York, 1979).

[5] S. Wolfram, "Computation theory of cellular automata", *Comm. Math. Phys.*, **96** (1984) 189.

[6] S. Wolfram, "Random sequence generation by cellular automata", *Adv. Appl. Math.*, **7** (1986) 123.

[7] L. Hurd, "Formal language characterizations of cellular automaton limit sets", *Complex Systems*, **1** (1987) 69.

[8] A. R. Smith III, "Real-time language recognition by one-dimensional cellular automata", *J. Comput. System Sci.*, **6** (1972) 233.

[9] R. Sommerhalder and S. C. van Westrhenen, "Parallel language recognition in constant time by cellular automata", *Acta Informatica*, **19** (1983) 397.

[10] O. H. Ibarra, M. A. Palis, and S. M. Kim, "Fast parallel language recognition by cellular automata", *Theor. Comp. Sci.*, **41** (1985) 231.

[11] U. Schoning, *Complexity and Structure*, (Springer-Verlag, Berlin, 1986).