Two-Dimensional Automata

Taylor J. Smith

January 2019
External Technical Report
ISSN-0836-0227-
2019-637

School of Computing
Queen's University
Kingston, Ontario, Canada K7L 3N6

Document prepared January 7, 2019
Copyright ©2019 Taylor J. Smith

Keywords: automata theory, closure properties, complexity theory, decision problems, formal languages, recognizability, two-dimensional automata, two-dimensional words

Abstract

In traditional automata theory, a finite automaton operates on linear strings of symbols. However, the automaton model can be extended to handle nonlinear inputs, such as trees, graphs, and multidimensional words. The well-known cellular automaton is an example of an automaton that operates on multidimensional words. Here, we consider automata that operate on two-dimensional words in a sequential manner. Such automata are called "two-dimensional automata".

Two-dimensional automata exhibit a number of interesting properties compared to one-dimensional automata. Due to their increased computational power, two-dimensional automata can recognize properties of input words that one-dimensional automata cannot recognize. Additionally, unlike in the one-dimensional case, a separation exists between deterministic and nondeterministic two-dimensional automata. However, this increase in power comes at the expense of losing some desirable automaton properties.

In this report, we introduce the two-dimensional automaton model and its variants. We discuss a number of properties this model possesses, operations under which this model is closed, and properties of input words that can be recognized by this model. We analyze decision problems for this model and present computational complexity results for some of these decision problems. We also review a number of open problems.

This report is based on the research proposal paper written as part of the author's PhD comprehensive examination.

Contents

List of Figures

List of Tables

1 Introduction

A finite automaton, also called a finite-state machine, is an abstract model of computation. In a finite automaton, a computation being performed by the automaton is in one of some finite number of states at any given moment, and the next step in the computation is given by a transition from that state to some possibly-different state based on the input word supplied to the automaton.

Traditionally, the input word supplied to a finite automaton is considered to be a linear object. However, the finite automaton model can be extended to allow for automata that operate on nonlinear objects; such extensions include tree automata [9], graph automata [8], and multidimensional automata [32, 50]. Here, we focus on multidimensional automata.

Nonlinear objects can be processed by their corresponding automaton model either sequentially (by reading one symbol at a time) or in parallel (by reading multiple symbols at a time). There are a number of multidimensional automaton models designed to process multidimensional words in parallel; these models include cellular automata [32], bounded cellular acceptors [55], online tessellation acceptors [16], parallel/sequential array automata [51], and pyramid cellular automata [11]. In the sequential case, however, only a small variety of models exist.

In this report, we consider extensions to the finite automaton model that operate on two-dimensional words in a sequential manner. Such automata are called "two-dimensional automata". The two-dimensional automaton model was originally introduced by Blum and Hewitt in 1967 [4, 5]. Some surveys on two-dimensional automata have since appeared in the literature [24, 35, 50], though the survey by Inoue and Takanami [24] has a stronger focus on the more general two-dimensional Turing machine model. Surveys are also available on the related fields of two-dimensional language theory [13, 50] and two-dimensional grammars [50, 52, 59, 60].

A two-dimensional automaton, given a sufficiently large input word, can simulate the computations of a two-counter machine [6]. Since two-counter machines are known to be Turing-equivalent, the two-dimensional automaton is much more powerful than its one-dimensional counterpart. Two-dimensional automata have both deterministic and nondeterministic variants. Although deterministic and nondeterministic finite automata in one dimension are equivalent [49], nondeterministic two-dimensional automata have more computational power than deterministic two-dimensional automata [4, 5]. It is possible to augment two-dimensional automata with a rudimentary kind of memory in the form of markers, and this addition gives us more computational power [4, 5, 46]. However, we will not discuss marker automata in this report.

Due to their increased computational power, two-dimensional automata can recognize properties of input words that one-dimensional automata cannot recognize. For instance, two-dimensional automata can recognize square words by moving diagonally and checking whether the input head reaches a corner or an edge [4, 5]. It is even possible for the automaton to recognize more peculiar properties, such as if a word has exponential side length [34, 40]. This may come as a surprise, since the two-dimensional automaton model has no ability to write symbols; in fact, the enhanced movement of the input head is enough to recognize such properties. However, this increased power comes at the cost of losing the ability to decide certain language problems, such as emptiness and universality [5, 63]. In fact, most decision problems for two-dimensional automata are undecidable, apart from language membership.

We can restrict some aspects of the two-dimensional automaton model to obtain other interesting models of computation. For example, a straightforward restriction limits the alphabet size to one symbol, but unary two-dimensional automata have no particular advantage over the usual model. One particularly useful variant of the two-dimensional automaton model is the three-way two-dimensional automaton [50]. This model is permitted to move its input head left, right, or down, but not up. Similarly, a two-way two-dimensional automaton can only move its input head right or down [10], but this results in the automaton not being able to read its entire input. Restricting the input head movement allows us to regain some—but not all—of the decision properties lost with the four-way model; emptiness is once again decidable, and universality is decidable in the deterministic case [22, 47]. However, with this restriction, we forfeit certain crucial abilities, such as the ability to read a column more than once.

Despite the amount of research on automata theory, relatively little is known about two-dimensional automata. Basic results on closure, recognizability, and decidability properties are known, but certain problems in these areas remain open. Other areas, such as complexity measures for two-dimensional automata, have very few known results. Remarkably, the existing literature has almost no results on the state complexity of two-dimensional automata, with the most meaningful results being Sipser's construction for converting an n-state two-dimensional automaton to an equivalent $O(n^2)$-state halting automaton by performing a depth-first search through the computation tree [57] and an improvement on this bound to $O(n)$ by Kunc and Okhotin [37, 38]. This gap in the literature persists despite state complexity being a much-investigated topic in automata theory. Throughout this report, we will see a number of open problems that offer promising opportunities for future research.

2 One-Dimensional Automata

In this section, we recall some fundamental definitions and results from the area of automata theory. Many of the definitions we recall here can be found in the textbook by Hopcroft and Ullman [14], and citations are provided in-line for other results.

2.1 Deterministic and Nondeterministic Automata

A finite automaton is an abstract machine that receives input in the form of a string of symbols from some finite, nonempty alphabet Σ. Formally, it is defined as follows.

Definition 2.1 (Deterministic finite automaton)**.** A deterministic finite automaton is a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is the input alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state of the computation, and $F \subseteq Q$ is a finite set of final states of the computation.

A deterministic finite automaton will always produce the same computation on the same input, since the transition function δ maps pairs of states and alphabet symbols to exactly one state. The notation $\delta(q_i, a_j)$ denotes the state corresponding to the transition on the pair consisting of state q_i and input symbol a_j. We may instead consider nondeterministic finite automata by making one small modification to the transition function.

Definition 2.2 (Nondeterministic finite automaton)**.** A nondeterministic finite automaton is a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0, and F are defined as in Definition 2.1 and $\delta : Q \times \Sigma \to 2^Q$ is the transition function.

We can now consider what it means for a finite automaton to compute something; specifically, under which conditions a finite automaton accepts or rejects some input.

Definition 2.3 (Accepting and rejecting computations)**.** Given an input word $w = a_1 a_2 \cdots a_n$ consisting of symbols chosen from an alphabet Σ, a finite automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ accepts w if there exists a sequence of states $r_0, r_1, \ldots, r_n \in Q$ such that

1. $q_0 = r_0$;
2. $r_{i+1} = \delta(r_i, a_{i+1})$ for all $i = \{0, 1, \ldots, n-1\}$; and
3. $r_n \in F$.

If such a sequence of states does not exist, then the finite automaton rejects w.

The language of an automaton $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of all input words accepted by the automaton $\mathcal{A}$. We denote the class of all languages accepted by deterministic (respectively, nondeterministic) finite automata by DFA (respectively, NFA).

In the sequel, to distinguish the previous notions of deterministic and nondeterministic finite automata from the main topic of two-dimensional automata, we will occasionally refer to the former notions collectively as "one-dimensional automata".

2.2 Two-Way Automata

In the traditional "one-way automaton", the input head reads the input word only in the forward direction. We can extend the one-way automaton model in such a way that, although it still operates on linear input words, it gains the ability to reread symbols in the input word. Since the input head can now move both forward and backward through the input word, we say that it is a "two-way automaton". We denote the class of all languages accepted by deterministic (respectively, nondeterministic) two-way automata by DFA-2W (respectively, NFA-2W).

When introducing a second direction of movement to our automaton model, we must make some assumptions about the input word in order to avoid undesirable effects. For instance, we assume that our input word is surrounded by two boundary symbols. The inclusion of these symbols does not affect the computational power of our model; it only prevents the input head from moving off of the input word.

In fact, not even the addition of the second direction of movement affects the computational power of our model. Rabin and Scott [49] and, independently, Shepherdson [56] proved that every two-way finite automaton has an equivalent one-way representation. As a consequence, both deterministic and nondeterministic two-way automata are equivalent in terms of computational power.

For an overview of other results relating to two-way automata, see, for example, the survey paper by Pighizzini [48]. We will revisit the two-way automaton model later during our discussion on two-dimensional automata.

2.3 Closure and Decidability Results

When we talk about a set having closure under an operation, we mean that, given two elements from that set, applying the operation to those elements produces an element again from that same set. Studying the closure properties of an abstract machine model is important, since it gives us insight into how we can combine multiple machines to achieve certain results while still maintaining the same amount of computational power.

Most "natural" operations are closed for languages accepted by one-dimensional automata. These operations include the set operations of union ($\cup$), intersection ($\cap$), and complement ($\neg$) and the word operations of concatenation ($\circ$), Kleene star (*), and reversal (^{R}).

Similar to how closure problems investigate operations on languages, decidability problems investigate questions we can ask about languages in general. We say that a problem is decidable if there exists an algorithmic procedure to solve that problem. However, decidability does not necessarily imply efficiency in either time or space.

One of the most common decidability problems is the membership problem. For a fixed finite automaton $\mathcal{A}$ accepting the language $L(\mathcal{A})$, the membership problem asks, given $w \in \Sigma^*$, whether $w \in L(\mathcal{A})$.

Remark. The membership problem, as stated here, is also known as the non-uniform variant of the problem. In the uniform variant of the problem, $\mathcal{A}$ is given as input instead of being fixed. Unless otherwise stated, we will assume in future sections that we are dealing with the non-uniform variant of the membership problem.

There are a number of other decidability problems for languages, but here we will focus on five problems in particular. Assume that we are given two finite automata $\mathcal{A}$ and $\mathcal{B}$ that accept the languages $L(\mathcal{A})$ and $L(\mathcal{B})$, respectively. Then the emptiness problem asks whether $L(\mathcal{A}) = \emptyset$, the universality problem asks whether $L(\mathcal{A}) = \Sigma^*$, the equivalence problem asks whether $L(\mathcal{A}) = L(\mathcal{B})$, the containment problem asks whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$, and the disjointness problem asks whether $L(\mathcal{A}) \cap L(\mathcal{B}) = \emptyset$. Each of these six problems are decidable for languages accepted by one-dimensional automata.

2.4 Complexity Results

There exist a number of complexity measures in formal language and automata theory. The most well-known of these measures is computational complexity: the amount of resources in time or space required to perform some task, like deciding a problem.

In a sense, the membership problem is the most straightforward decision problem for an abstract machine to solve, since it only has to determine one thing: whether or not a given word is in the machine's language. Therefore, this problem serves as a good baseline to evaluate the computational complexity and power of an abstract machine model.

The non-uniform membership problem for both deterministic and nondeterministic one-dimensional automata is in the complexity class NC^1 [1, 2]. It is known that $\mathsf{NC}^1 \subseteq \mathsf{L}$, where L contains all problems that are solvable by a deterministic Turing machine using logarithmic space.

Another measure we can investigate is state complexity: the number of states that are both necessary and sufficient for an m-state minimal deterministic finite automaton to accept the language corresponding to some operation. For binary operations, we assume we have

	DFA		DFA-2W		
	Upper/lower bound	Ref.	Upper bound	Lower bound	Ref.
$\cup$	$m \cdot n$	[3]	$4m + n + 4$	$m + n - o(m + n)$	[30]
$\cap$	$m \cdot n$	[3]	$m + n + 1$	$m + n - o(m + n)$	[30]
$\neg$	m	—	$4n + 3$	n	[12]
$\circ$	$m \cdot 2^n - 2^{n-1}$	[64]	$2m^{m+1} \cdot 2^{n^{n+1}}$	$\Omega\left(\frac{m}{n}\right) + \frac{2^{\Omega(n)}}{\log(m)}$	[30]
$*$	$2^{m-1} + 2^{m-2}$	[64]	$2^{O\left(n^{n+1}\right)}$	$\frac{1}{n} \cdot 2^{\frac{n}{2}-1}$	[30]
R	2^m	[39]	$n + 2$	$n + 1$	[30]

Table 1: State complexity results for one-dimensional automaton models

two minimal deterministic finite automata that consist of m and n states, respectively. We can define nondeterministic state complexity in the same way by making the appropriate change to the abstract machine model.

Table 1 summarizes a number of state complexity results for both one-way and two-way automaton models. Note that the bound for the complement of DFA is immediate, since taking the complement is equivalent to converting all final states in an automaton to non-final states (and vice versa).

A well-known result in automata theory is that both nondeterministic and deterministic finite automata have the same computational power. However, the process of determinization may result in an exponential blowup of states in the worst case. The proof of equivalence is an elementary result that can be found in any standard automata theory textbook; for example, Hopcroft and Ullman [14] or Sipser [58]. The proof of possibly-exponential blowup is due to Meyer and Fischer [42], and it uses the subset construction of Rabin and Scott [49].

Although exponential blowup does not always occur, 2^n is the tight upper bound for the number of states in the worst case after determinization; that is, there exist n-state nondeterministic finite automata where the 2^n-state bound will always be reached after determinization [41, 45].

3 Two-Dimensional Automata

We can extend the power of one-dimensional automata in two ways: first, by considering arrays of symbols instead of linear strings of symbols, and second, by allowing the automaton to traverse the input in more than two directions. Together, these modifications result in an abstract machine model known as a two-dimensional automaton (also known as a four-way automaton, a picture automaton, or an array automaton).

3.1 Definitions

To begin, we define the objects on which a two-dimensional automaton operates. A two-dimensional word $W \in \Sigma^{m \times n}$ is a map from $\{0, 1, \ldots, m-1\} \times \{0, 1, \ldots, n-1\}$ to some alphabet Σ, for some dimensions m and n. In other terms, a two-dimensional word is an $m \times n$ array of symbols. (More generally, a two-dimensional word can be any contiguous shape of symbols from Σ, but here we only consider rectangular and square two-dimensional

$$\begin{array}{cccccc} \# & \# & \# & \cdots & \# & \# \\ \# & a_{1,1} & a_{1,2} & \cdots & a_{1,n} & \# \\ \# & a_{2,1} & a_{2,2} & \cdots & a_{2,n} & \# \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \# & a_{m,1} & a_{m,2} & \cdots & a_{m,n} & \# \\ \# & \# & \# & \cdots & \# & \# \end{array}$$

Figure 1: An example of a two-dimensional input word

words.) The notation $\Sigma^{m \times n}$ represents the set of all two-dimensional words of dimension $m \times n$, and we have similar notations for other special sets: Σ^{**} denotes the set of all two-dimensional words of any dimension, for example.

Two-dimensional automata receive two-dimensional words as input. The input word to a two-dimensional automaton is special, in that the first and last row and column of the input word consists entirely of a unique boundary symbol, denoted here by #. To accommodate these boundary symbols, we must adjust the indexing of symbols in the input word so that the top-left symbol of the input word is at index $(1, 1)$ instead of index $(0, 0)$. An example of a two-dimensional input word is shown in Figure 1.

A two-dimensional automaton uses an input head to scan the symbols of the input word in both the horizontal and vertical directions. The input head is initially positioned over the top-left symbol of the input word—that is, at position $(1, 1)$—and the two-dimensional automaton proceeds with its computation by reading symbols, transitioning between states, and moving the input head. Once the two-dimensional automaton either accepts or rejects the word, it halts the computation. The notions of acceptance and rejection will be presented shortly.

We are now ready to formalize our definition of a two-dimensional automaton.

Definition 3.1 (Two-dimensional deterministic finite automaton)**.** A two-dimensional deterministic finite automaton is a tuple $(Q, \Sigma, \delta, q_0, q_A, q_R)$, where

- Q is a finite set of states;
- Σ is the input alphabet (with $\# \notin \Sigma$ acting as a boundary symbol);
- $\delta : (Q \setminus \{q_A, q_R\}) \times (\Sigma \cup \{\#\}) \to Q \times \{U, D, L, R\}$ is the transition function;
- $q_0 \in Q$ is the initial state of the computation;
- $q_A \in Q$ is the accepting state of the computation; and
- $q_R \in Q$ is the rejecting state of the computation.

At a glance, the definition of a two-dimensional automaton appears to be very similar to its one-dimensional analogue, apart from the added dimension. However, there are a few

differences that set two-dimensional automata apart from one-dimensional models in other ways.

The first difference is in the way the automaton input head moves. While a one-dimensional automaton may move its input head in at most two directions, a two-dimensional automaton may move its input head in four directions: up, down, left, or right. We denote these directions using the symbols U, D, L, and R, respectively. We may optionally specify a "no move" option, where the input head does not move in any of the four directions after reading a symbol. This report does not assume automata are able to use the "no move" option.

To make the notion of a two-dimensional transition more precise, consider some two-dimensional automaton $\mathcal{A}$ with transition function δ. If $\mathcal{A}$ is in state q_i and its input head reads symbol $a_j \in (\Sigma \cup \{\#\})$, and if $\delta(q_i, a_j) = (q_k, d)$ where $d \in \{U, D, L, R\}$, then in the next computation step, $\mathcal{A}$ transitions to state q_k and moves its input head by one square in the direction d. Since the domain of the transition function δ includes only non-accepting and non-rejecting states, the computation must necessarily halt upon reaching either an accepting or a rejecting state.

The second difference is that, while the alphabet of a two-dimensional automaton may again consist of any set of symbols, the input word is required to be surrounded by the boundary symbol #. This symbol ensures that the two-dimensional automaton does not move its input head outside of the boundaries of its input word. If the input head encounters a boundary symbol after moving in some direction d, then it makes a second move d^{-1} to return to the input word. (In this context, d^{-1} is the inverse of d; for example, if $d = L$, then $d^{-1} = R$.) We may assume that the last move made by the input head is stored by the current state in order to perform this inverse move. Alternatively, the inverse move may be specified directly by using distinct boundary symbols for each border of the word.

Finally, a two-dimensional automaton can specify explicitly a rejecting state q_R in addition to an accepting state q_A. Once the automaton reaches either q_A or q_R, the computation halts.

Remark. The first two differences—enhanced input head movement and inclusion of boundary symbols—also apply to the two-way automaton model from Section 2.2. Indeed, a two-way automaton is essentially a two-dimensional automaton operating on an input word with one row.

Similar to the one-dimensional case, we can introduce nondeterminism to the two-dimensional automaton model in the usual way.

Definition 3.2 (Two-dimensional nondeterministic finite automaton)**.** A two-dimensional nondeterministic finite automaton is a tuple $(Q, \Sigma, \delta, q_0, q_A, q_R)$, where Q, Σ, q_0, q_A, and q_R are defined as in Definition 3.1 and $\delta : (Q \setminus \{q_A, q_R\}) \times (\Sigma \cup \{\#\}) \to 2^{Q \times \{U,D,L,R\}}$ is the transition function.

In order to perform a computation, a two-dimensional automaton moves through a sequence of configurations specified by the automaton's state and the symbol being read by the automaton's input head at each stage of the computation.

Definition 3.3 (Configuration of a two-dimensional automaton)**.** A configuration of a two-dimensional automaton $\mathcal{A}$ on an input word $W \in \Sigma^{m \times n}$ is a tuple (q, i, j), where q is the current state of $\mathcal{A}$ and i and j are the current positions of the input head in W, where $1 \leq i \leq m$ and $1 \leq j \leq n$.

We naturally define the starting configuration of a two-dimensional automaton as the tuple $(q_0, 1, 1)$. Note that the automaton does not have access to the values i or j; these values are only used to distinguish configurations in writing. We connect configurations to the transition function δ in the following way. Given a two-dimensional automaton $\mathcal{A}$ with input word W, if

- $\mathcal{A}$ is currently in state $p \in Q$;
- the symbol at position (i, j) in W is $a \in \Sigma$; and
- $\delta(p, a) = (q, U)$;

then we say that configuration (p, i, j) yields configuration $(q, i - 1, j)$. If $i = 1$, then the input head moves onto the boundary symbol # and the upward move is reversed. The configurations yielded for directions D, L, and R are defined similarly.

With this notion of configurations, we can now specify how a two-dimensional automaton accepts and rejects input words.

Definition 3.4 (Accepting and rejecting computations)**.** A two-dimensional automaton $\mathcal{A}$ accepts a word W if, at some point in its computation, it enters a configuration (q_A, i, j), where $0 \leq i \leq m + 1$ and $0 \leq j \leq n + 1$. Likewise, $\mathcal{A}$ rejects W if, at some point in its computation, it enters a configuration (q_R, i, j).

Note that our definition of acceptance for two-dimensional automata differs from Definition 2.3 for one-dimensional automata, since a one-dimensional automaton may visit any number of accepting states during its computation before halting. Acceptance for a one-dimensional automaton is determined only by the state reached upon reading the last symbol in the input word.

Moreover, it is possible for a two-dimensional automaton not to halt. If the automaton never enters one of the configurations specified in Definition 3.4, then it will continue its computation indefinitely.

We denote the class of all languages accepted by two-dimensional deterministic (respectively, nondeterministic) finite automata by 2DFA (respectively, 2NFA).

Remark. It is important to note that our notion of a two-dimensional automaton is distinct from the more well-known notion of a cellular automaton [32], which also operates in two-dimensional space. The two-dimensional automata we consider in this report are purely sequential models, whereas cellular automata are capable of processing each symbol, or "cell", of their input in parallel.

3.2 General Properties

By adding a dimension to a finite automaton, it stands to reason that we will gain computational power. Intuitively, this must be the case, since one-dimensional automata cannot process the input words supplied to two-dimensional automata.

In an early paper, Blum and Sakoda proved a much stronger result: two-dimensional automata can, in some sense, simulate one-dimensional Turing machines [6]. Blum and Sakoda proved this result by modeling a two-dimensional automaton as a two-counter machine and using a result of Minsky stating that two-counter machines are Turing-equivalent [44].

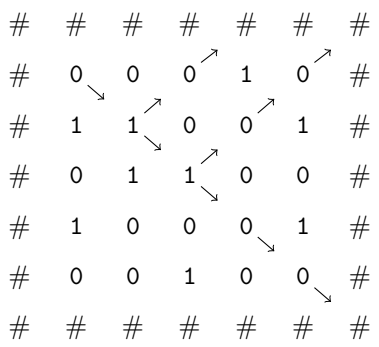

Figure 2: An illustration of Lemma 3.5

Since introducing an additional dimension increases the computational power of our machine model, does the introduction of nondeterminism have the same effect? Although nondeterministic and deterministic finite automata in one dimension have the same computational power, Blum and Hewitt proved that this is not the case in two dimensions [4, 5]. They proved this result by showing that the following problem for two-dimensional automata is decidable with nondeterminism, but is undecidable without nondeterminism.

Lemma 3.5. *A nondeterministic two-dimensional automaton can decide the problem of whether a square input word of odd side length contains a* `1` *as its center symbol.*

The idea behind Lemma 3.5 is that nondeterminism allows the automaton to "guess" its next move based on what symbol is read by the input head. The input head moves along the diagonal of the input word and, upon reading a `1`, nondeterministically switches its direction with a 90-degree counterclockwise turn. The movement of the input head on an example input word is shown in Figure 2, where a combination of one R movement and one $\{U, D\}$ movement is illustrated by a diagonal arrow. If the input head reaches the upper-right corner of the input word, then it must have read a `1` in the center square of the input word.

The key argument given by Blum and Hewitt [4, 5] is that deterministic two-dimensional automata do not have sufficient power to distinguish distinct input words in this way, so they cannot decide the problem. As a consequence, we get a separation between deterministic and nondeterministic two-dimensional automata.

Theorem 3.6. *A nondeterministic two-dimensional automaton has more computational power than a deterministic two-dimensional automaton.*

By definition, we know that two-dimensional input words are surrounded by a boundary symbol #, and the input head of a two-dimensional automaton is not permitted to leave the boundary of the input word. However, even if we allowed the input head to travel outside of the input word and onto an infinite plane of boundary symbols, we would not gain any computational power. Milgram showed that deterministic two-dimensional automata that can traverse boundary symbols in this way are no more powerful than those that cannot [43], and Salo showed the same result for nondeterministic two-dimensional automata [53].

A nice property for a machine model to have is an assurance that the machine will halt on every input. We can use this property to simplify certain proofs of closure or decidability. Sipser proved that, for any space-bounded deterministic Turing machine $\mathcal{T}$, there exists a deterministic Turing machine $\mathcal{T}'$ using the same amount of space as $\mathcal{T}$ such that $L(\mathcal{T}') = L(\mathcal{T})$ and $\mathcal{T}'$ is guaranteed to halt on every input [57]. Moreover, if the machine $\mathcal{T}$ consists of n states, then $\mathcal{T}'$ consists of $O(n^2)$ states. It is possible to obtain a similar result for two-dimensional automata by using Sipser's construction [34, 40].

Theorem 3.7. *For any deterministic two-dimensional automaton $\mathcal{A}$, we can construct a deterministic two-dimensional automaton $\mathcal{A}'$ with $L(\mathcal{A}') = L(\mathcal{A})$ that is guaranteed to halt on every input.*

The automaton $\mathcal{A}'$ specified in the theorem operates by working backward through its computation. Starting in the accepting state of $\mathcal{A}$, the automaton $\mathcal{A}'$ performs a depth-first search through the computation tree to reach the initial state of $\mathcal{A}$. This technique produces a loop-free automaton that accepts if and only if the original automaton accepts, and the loop-freeness property ensures that $\mathcal{A}'$ halts.

In order to obtain precise complexity results for problems where we construct a specified two-dimensional automaton from a given initial automaton, we must know how long the construction takes (time complexity) and how many additional states are needed compared to the initial automaton (state complexity). State complexity is the most salient question, since in the case of automaton constructions, the time complexity cannot be better than the state complexity.

Although Sipser's construction gives a $O(n^2)$ bound on the number of states for a loop-free deterministic two-dimensional automaton, Kunc and Okhotin have since improved this bound to $8n + 9$ states for an n-state deterministic two-dimensional automaton [37, 38]. The model used by Kunc and Okhotin is slightly different than the model considered in this report, since their model uses different boundary symbols on each edge of the input word. Adapting this construction to our model will likely change the additive constant, but not the multiplicative constant.

Problem 1. Can the $8n + O(1)$ state bound on the number of states for a loop-free deterministic two-dimensional automaton be improved (either in the multiplicative constant or the additive constant)?

3.3 Variants

We can define a number of variants of the two-dimensional automaton model by either restricting or enhancing some aspect of the definition of a two-dimensional automaton. The simplest variant is obtained by restricting the automaton's alphabet Σ to consist of one symbol. This gives us a unary two-dimensional automaton, and we denote the classes of deterministic and nondeterministic unary two-dimensional automata by $\mathsf{2DFA\text{-}1}\Sigma$ and $\mathsf{2NFA\text{-}1}\Sigma$, respectively.

If we restrict the transition function, then we obtain yet another variant. In the three-way two-dimensional automaton model, we restrict the movement of the input head to three directions instead of four. Specifically, we prohibit the input head from moving upward. This change has a substantial impact on the ability of a two-dimensional automaton to read its input word, since once the input head leaves a row, it cannot return to that row by moving

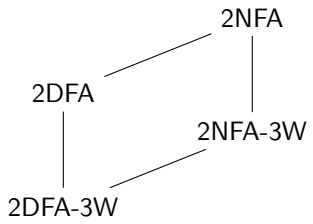

Figure 3: Inclusions among two-dimensional automaton classes

upward. We denote the classes of three-way deterministic and nondeterministic automata by 2DFA-3W and 2NFA-3W, respectively. We also define the equivalent unary classes by 2DFA-3W-1Σ and 2NFA-3W-1Σ, respectively.

There exist a number of inclusion results between two-dimensional automaton models. Figure 3 illustrates some of the results most relevant to this report. From the result of Blum and Hewitt given in Theorem 3.6, we get the proper inclusion 2DFA $\subset$ 2NFA [4, 5]. Moreover, this inclusion is proper even over a unary alphabet [33]. Rosenfeld gives the proper inclusions 2DFA-3W $\subset$ 2NFA-3W, 2DFA-3W $\subset$ 2DFA, and 2NFA-3W $\subset$ 2NFA [50].

No other non-trivial inclusions exist between four-way and three-way automata in the deterministic or nondeterministic cases [28]. Namely, both 2DFA $\setminus$ 2NFA-3W $\neq \emptyset$ and 2NFA-3W $\setminus$ 2DFA $\neq \emptyset$, where the first result comes as a consequence of Theorem 5.3 and the second result comes from a stronger separation due to Kari and Salo [35].

4 Closure Results

Two-dimensional automata are closed under certain operations, depending on the model being considered. However, positive closure results for two-dimensional automata are sparser due to the increase in complexity incurred by adding a second dimension to our model.

To begin, let us define the two-dimensional analogues to each operation we introduced in Section 2.3. We follow the definitions given in the survey by Inoue and Takanami [24]. Assume we have two two-dimensional words

$$A = \begin{matrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{matrix} \quad \text{and} \quad B = \begin{matrix} b_{1,1} & \cdots & b_{1,n'} \\ \vdots & \ddots & \vdots \\ b_{m',1} & \cdots & b_{m',n'} \end{matrix}$$

of dimension $m \times n$ and $m' \times n'$, respectively. Note that the words A and B are not surrounded by the boundary symbol #; this is because we do not assume here that the words are being used as the input to some two-dimensional automaton.

The vertical and horizontal concatenations of two two-dimensional words, denoted by $\odot$

and $\ominus$, respectively, are defined only when $m = m'$ (for $\oslash$) or $n = n'$ (for $\ominus$).

$$A \oslash B = \begin{matrix} a_{1,1} & \cdots & a_{1,n} & b_{1,1} & \cdots & b_{1,n'} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} & b_{m,1} & \cdots & b_{m,n'} \end{matrix} \qquad A \ominus B = \begin{matrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \\ b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{m',1} & \cdots & b_{m',n} \end{matrix}$$

The reversal operation, also called "row reflection" [24], is performed by reversing the order of the rows in a two-dimensional word. The rotation operation is performed by rotating a two-dimensional word clockwise by 90 degrees. Here, we denote reversal by a superscript letter R, which is a common notation in the literature, and we denote rotation by the less-common symbol $\circlearrowright$.

$$A^R = \begin{matrix} a_{m,1} & \cdots & a_{m,n} \\ \vdots & \ddots & \vdots \\ a_{1,1} & \cdots & a_{1,n} \end{matrix} \qquad A^{\circlearrowright} = \begin{matrix} a_{m,1} & \cdots & a_{1,1} \\ \vdots & \ddots & \vdots \\ a_{m,n} & \cdots & a_{1,n} \end{matrix}$$

Using concatenation operations, we can define additional operations on rows and columns of two-dimensional words. For a set of two-dimensional words L, let $L^{1\ominus} = L$ and $L^{(i+1)\ominus} = L^{i\ominus} \ominus L$. We define the row closure of L as $L^{\ominus} = \bigcup_{i \geq 1} L^{i\ominus}$. We define the column closure of L, denoted by $L^{\oslash}$, in a similar way.

Row and column cyclic closures are similar to row and column closures, but instead of operating on an entire two-dimensional word, we operate only on individual rows or columns. We say that a row cyclic shift of a two-dimensional word A is a rearrangement of rows such that the k top rows of A, for some value $1 \leq k \leq m$, are removed and concatenated beneath the bottom row of A. Diagrammatically, we have the following situation:

$$\begin{matrix} a_{k+1,1} & \cdots & a_{k+1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \\ a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{k,1} & \cdots & a_{k,n} \end{matrix}$$

Then the row cyclic closure of a set of two-dimensional words L is the set of all row cyclic shifts of words in L. We define column cyclic shifts and column cyclic closures in a similar way.

With our operations defined, we can now discuss specific closure results. Each of the results discussed in this section are summarized in Table 2.

For deterministic two-dimensional automata, we have positive closure results for all set-theoretic operations. We also have positive results for some word operations, but not all.

Theorem 4.1. *The class 2DFA is closed under union, intersection, complement, reversal, and rotation. It is not closed under concatenation, row/column closure, or row/column cyclic closure.*

	2DFA	2NFA	2DFA-3W	2NFA-3W
$\cup$	✓	✓	✗	✓
$\cap$	✓	✓	✗	✗
$\neg$	✓	✗	✓	✗
$\circledcirc/\ominus$	✗	✗	✗	$\checkmark_{\ominus}$ $\times_{\circledcirc}$
$\overset{}{}_{R}$	✓	✓	✗	✓
$\circlearrowright$	✓	✓	✗	✗
row/column closure	✗	✗	✗	$\checkmark_{R}$ $\times_{C}$
row/column cyclic closure	✗	✗	✗	✗

Table 2: Closure results for two-dimensional automaton models

We can prove deterministic two-dimensional automata are closed under set-theoretic operations such as union, intersection, and complement by using Sipser's construction [57]. Non-closure under concatenation, row/column closure, and row/column cyclic closure was proved by Inoue, Takanami, and Nakamura [18, 19, 25].

Recall the state complexity bounds for two-way deterministic one-dimensional automata from Table 1 in Section 2.4. Since the two-way one-dimensional automaton model is similar to the two-dimensional automaton model, we should expect to obtain similar—but not identical—state complexity bounds for operations on two-dimensional automata.

We conjecture that the union of two nondeterministic two-dimensional automata has a state complexity of $m + n - 1$. A proof sketch for the upper bound is immediate: we make a nondeterministic choice for which automaton to run first, then accept if either the first or second automaton accepts. The constant term comes from not needing to repeat states q_A and q_R, but since we require a new initial state for our nondeterministic choice, we only save one state.

Furthermore, we conjecture that the intersection of two nondeterministic two-dimensional automata has a state complexity of $m + n$. Here, we run one automaton until it halts, then move the input head back to the initial state before running the second automaton, and accept if both automata accept. In this case, we again don't need to repeat states q_A and q_R, but we do require two states to move the input head back to the initial state.

A proof for the lower bound is not as immediate, though we conjecture that a matching lower bound holds for both operations.

Problem 2. Prove that $m+n-1$ (respectively, $m+n$) is the tight state complexity bound of the union (respectively, intersection) of two nondeterministic two-dimensional automata.

The state complexity of the intersection of two deterministic two-dimensional automata should match the previous bound for the nondeterministic case. For the union operation, we have a similar (but more interesting) conjecture since we must convert one of the automata to a loop-free halting automaton by the construction of Kunc and Okhotin [37, 38]. Without loss of generality, we may convert the smaller of the two automata, leading to a state complexity upper bound of $O(m + n)$.

Problem 3. Prove that $\Omega(m + n)$ is the lower bound for the state complexity of the union of two deterministic two-dimensional automata.

Interestingly, if we consider three-way deterministic two-dimensional automata, then we lose all closure properties except for complement. Non-closure under union, intersection, concatenation, reversal, rotation, row/column closure, and row/column cyclic closure were all proved by Inoue and Takanami [20, 21, 23, 27]. Closure under complement was proved by Szepietowski [62]. Since the general construction due to Sipser [57] cannot be applied to automata with restricted moves, Szepietowski used a different technique to establish closure under complement.

In the nondeterministic case, we preserve all of the same closure properties as deterministic two-dimensional automata except for complement.

Theorem 4.2. *The class 2NFA is closed under union, intersection, reversal, and rotation. It is not closed under complement, concatenation, row/column closure, or row/column cyclic closure.*

Non-closure of nondeterministic two-dimensional automata under complement was proved by Kari and Moore [33], while non-closure under concatenation, row/column closure, and row/column cyclic closure was proved by Inoue, Takanami, and Nakamura [18, 19, 25, 26].

Three-way nondeterministic two-dimensional automata again lose some closure properties compared to the corresponding four-way model. Closure under union is preserved with three-way automata, but closure under intersection and rotation are lost [21]. In exchange, however, three-way automata gain two closure results: horizontal concatenation and row closure [20]. The fact that three-way automata gain only row-centric closure properties stems from the fact that the input head cannot move upward, thus preventing the automaton from reading multiple columns.

Szepietowski showed that three-way nondeterministic two-dimensional automata were closed under reversal by proving the same result for the more powerful two-dimensional nondeterministic Turing machine model [61, 62].

We may ask similar state complexity questions for three-way automata as before. In particular, the state complexity of the closure properties gained by three-way nondeterministic two-dimensional automata would be interesting to investigate. For instance, in the case of horizontal concatenation, it seems that a reasonable state complexity bound is $m + n$.

Problem 4. What is the state complexity of the horizontal concatenation of two three-way nondeterministic two-dimensional automata?

5 Recognition Results

Earlier, we learned by Lemma 3.5 that nondeterministic two-dimensional automata are capable of recognizing two-dimensional words that have a `1` as their center symbol, while deterministic two-dimensional automata are not able to recognize the same property. This is one of the earliest results relating to the recognition power of two-dimensional automata; that is, the ability of such an automaton to answer questions about individual two-dimensional words. In this section, we review a number of other results relating to the recognition power of two-dimensional automata.

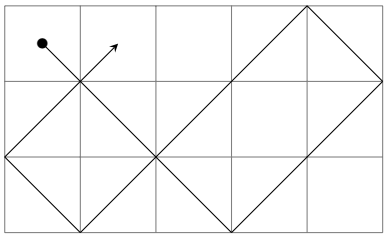

Figure 4: Recognizing a two-dimensional word with coprime dimensions

5.1 Recognizing General Properties

In their seminal paper, Blum and Hewitt presented a number of properties of two-dimensional words that a deterministic two-dimensional automaton $\mathcal{A}$ is capable of recognizing [4, 5]. These properties include deciding:

1. whether A contains exactly k occurrences of a given symbol;
2. whether A contains a single, contiguous rectangle/square consisting of the same symbol, with sides parallel to the edges of A; and
3. whether one of any number of pathwise-connected components in A is square.

In the same paper, Blum and Hewitt gave the following example of a property that cannot be recognized by a deterministic two-dimensional automaton.

Theorem 5.1. *No deterministic two-dimensional automaton can recognize whether a two-dimensional word A is symmetric about its central column.*

The fact that a deterministic two-dimensional automaton cannot recognize symmetry within an input word relates to our earlier discussion about Lemma 3.5 and is a consequence of Theorem 3.6.

5.2 Recognizing Dimensions

It is easy to see that two-dimensional automata are capable of recognizing two-dimensional words of given dimension $m \times n$ for $m, n \geq 1$; construct a two-dimensional automaton that recognizes input words of dimension $m \times n$ for fixed m and n, then check if the given two-dimensional word is accepted by this automaton.

Two-dimensional automata can also recognize two-dimensional words that are square (that is, of dimension $n \times n$ for $n \geq 1$) by moving along the diagonal of the word. In this case, we can use a general automaton that works for any value n.

A number of recognizability results relate to the side lengths of a given two-dimensional word and the relationship between dimensions. Many of these recognizability results illustrate the increase in power that comes with adding a dimension to our computational model; two-dimensional automata are capable of testing properties that one-dimensional automata cannot test. For instance, Lindgren, Moore, and Nordahl showed that two-dimensional automata are capable of recognizing the following relationship between dimensions [40].

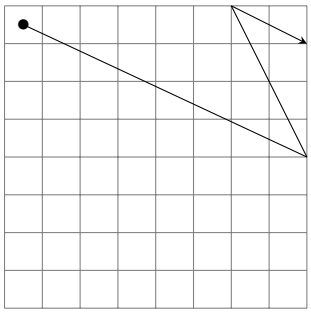

Figure 5: Recognizing a two-dimensional word with exponential dimensions

Theorem 5.2. *Deterministic two-dimensional automata can recognize the language of two-dimensional words with dimension $m \times n$, where $m, n \geq 1$ and m and n are coprime.*

An automaton recognizing such a "coprime dimension" language moves its input head around the input word like a billiard ball around a table, reflecting off of each border of the word at a 90-degree angle. If the input head moves onto the symbol to the right of its initial position, then it accepts the input word. Essentially, this construction mimics a two-counter machine that branches both when a counter reaches zero and when a counter reaches its maximum value. An illustration of this technique is shown in Figure 4.

Lindgren et al. [40] also proved that deterministic two-dimensional automata can recognize two-dimensional words with exponential side length.

Theorem 5.3. *Deterministic two-dimensional automata can recognize the language of two-dimensional words with dimension $2^n \times 2^n$, where $n \geq 0$.*

In this case, an automaton recognizing the "exponential dimension" language uses a series of knights' moves to verify that its input word belongs to the language. An illustration of this technique is shown in Figure 5. The same figure originally appeared in the paper by Lindgren et al. [40].

Kari and Moore gave a more complex construction to show that two-dimensional automata were capable of recognizing two-dimensional words of doubly-exponential side length [34].

Theorem 5.4. *Deterministic two-dimensional automata can recognize the language of two-dimensional words with dimension $2^{2^n} \times 2^{2^n}$, where $n \geq 0$.*

In the more general case of two-dimensional words of dimension $n \times f(n)$ for some function $f(n)$, there exist some results showing that certain functions cannot be recognized. Inoue and Nakamura introduced the notion of a function acceptable by a unary two-dimensional automaton [17] and proved that the functions $f_1(n) = n^2$, $f_2(n) = k^n$ for $k \geq 2$, and $f_3(n) = n!$ are unacceptable; that is, no unary two-dimensional automaton can recognize words of dimension $n \times f_i(n)$ for $i = \{1, 2, 3\}$.

Despite the number of positive results in the literature relating to recognizability, some problems remain unsolved. Kari and Moore note that there currently exist no methods to prove that a language L cannot be square-recognized by a two-dimensional automaton,

where "square recognition" denotes a matching of horizontal and vertical dimensions in each word of the language [34]. In the same paper, the authors make the following conjecture, which they believe to be false.

Problem 5. Can deterministic two-dimensional automata recognize the language of unary two-dimensional words with dimension $p \times p$, where p is prime?

Recognizability for three-way two-dimensional automata is not understood as well as the four-way model. The main result specifying the kinds of words recognizable by three-way automata is given by Kinber, who proves that the dimensions of a two-dimensional input word recognized by a three-way deterministic two-dimensional automaton are related by certain bilinear forms [36].

One approach to gaining a better understanding of the recognition power of three-way and four-way automata is to investigate properties of new operations on languages accepted by those automata. We define one such operation as follows: given a two-dimensional automaton $\mathcal{A}$, let the "row projection" of $L(\mathcal{A})$ be the set of all first rows of words in $L(\mathcal{A})$. We define the "column projection" of $L(\mathcal{A})$ in a similar way. Note that these operations produce one-dimensional languages.

For four-way automata, row/column projection languages are not always regular; consider, for example, taking the projection of the "exponential dimension" languages in Theorems 5.3 and 5.4. Even for unary four-way automata, regularity is not guaranteed. Asking the same question about the three-way automaton model could give us additional insight about the kinds of words recognized by this weaker model.

Problem 6. Does applying the row/column projection operation to the language of a three-way two-dimensional automaton produce a regular language?

We note that Kinber's result [36] solves this problem for three-way deterministic automata over a unary alphabet, since bilinear forms define unary regular languages.

6 Decidability Results

In this section, we revisit the problems we introduced in Section 2.3 and investigate whether these same problems are decidable for two-dimensional automata. Recall that the six problems we introduced previously were the membership, emptiness, universality, equivalence, containment, and disjointness problems.

In the one-dimensional case, we found that each of our six decision problems was decidable for both deterministic and nondeterministic finite automata. However, as we will see, the situation is quite different for two-dimensional automaton models. Results for four-way two-dimensional automata are summarized in Table 3, while results for three-way two-dimensional automata are summarized in Table 4. Open problems relating to decidability properties are represented in each table by "**?**".

6.1 Four-Way Two-Dimensional Automata

To begin, the membership problem for two-dimensional automata is decidable, since we can simply check whether a given input word is accepted by a given two-dimensional automaton. Decidability holds for all two-dimensional automaton models, though the procedure used to

	2DFA	2NFA	2DFA-1Σ	2NFA-1Σ
membership	✓	✓	✓	✓
emptiness	✗	✗	✗	✗
universality	✗	✗	✗	✗
equivalence	✗	✗	✗	✗
containment	✗	✗	✗	✗
disjointness	✗	✗	✗	✗

Table 3: Decidability results for four-way two-dimensional automaton models

test membership differs based on the model. We will discuss specific membership testing procedures and their complexity in Section 7.

Unfortunately, the remaining problems are mostly undecidable in two dimensions. Both the emptiness and the universality problems are undecidable for two-dimensional automata [4], even when we restrict the automaton to be deterministic and the alphabet to be unary [63].

Theorem 6.1. *The emptiness and universality problems are undecidable for the class 2DFA-1Σ.*

The proof of Theorem 6.1 uses Minsky's result that two-counter machines are Turing-equivalent [44]. To prove the undecidability of the emptiness problem, we construct a deterministic unary two-dimensional automaton $\mathcal{A}$ in such a way that $L(\mathcal{A}) \neq \emptyset$ if and only if a given two-counter machine $\mathcal{M}$ accepts the empty word. Since the problem of determining whether a given Turing machine accepts the empty word is undecidable by Rice's theorem, the problem of determining whether $L(\mathcal{A}) \neq \emptyset$ is also undecidable. The proof of the undecidability of the universality problem is similar, since the problem of determining whether a given Turing machine accepts all words is also undecidable by Rice's theorem.

Since the emptiness problem is a special case of the equivalence and containment problems (in that the emptiness problem is the same as asking whether $L(\mathcal{A}) = \emptyset$ or $L(\mathcal{A}) \subseteq \emptyset$, respectively), we get negative results for decidability with these problems over unary deterministic two-dimensional automata.

Corollary 6.2. *The equivalence and containment problems are undecidable for the class 2DFA-1Σ.*

Since the emptiness problem is also a special case of the disjointness problem (in that $L(\mathcal{A}) = \emptyset$ if and only if $L(\mathcal{A}) \cap \Sigma^{**} = \emptyset$, where Σ^{**} is the notation for the set of all two-dimensional words from Section 3.1), we get a similar outcome for the decidability of this problem.

Corollary 6.3. *The disjointness problem is undecidable for the class 2DFA-1Σ.*

6.2 Three-Way Two-Dimensional Automata

If we restrict the input head movement, then our decidability results become much more favourable. Both the emptiness and the universality problems are decidable for deterministic

	2DFA-3W	2NFA-3W	2DFA-3W-1Σ	2NFA-3W-1Σ
membership	✓	✓	✓	✓
emptiness	✓	✓	✓	✓
universality	✓	✗	✓	?
equivalence	?	✗	✓	?
containment	✗	✗	✓	?
disjointness	✗	✗	✓	?

Table 4: Decidability results for three-way two-dimensional automaton models

two-dimensional automata when movement is restricted to three directions. Decidability holds for both unary automata and general-alphabet automata.

However, when we introduce nondeterminism, we get a different result. While the emptiness problem remains decidable for three-way nondeterministic two-dimensional automata over both unary and general alphabets, the universality problem becomes undecidable over general alphabets.

Theorem 6.4. *The emptiness problem is decidable for the classes 2NFA-3W and 2NFA-3W-1Σ. The universality problem is undecidable for the class 2NFA-3W.*

Inoue and Takanami proved the decidability of the emptiness problem for unary three-way two-dimensional automata [22]. Their proof uses the fact that the emptiness problem for two-way nondeterministic one-dimensional automata is decidable. The corresponding result for general alphabets was later proved by Petersen [47].

The proof of the decidability of the universality problem for three-way deterministic two-dimensional automata is also due to Inoue and Takanami as well as Petersen. It follows from the decidability of the emptiness problem and the closure of the complement operation for three-way deterministic two-dimensional automata.

Since we know that the universality problem is undecidable for three-way nondeterministic automata over a general alphabet, it would be useful to know if the same outcome occurs for automata over a unary alphabet.

Problem 7. Is the universality problem decidable for the class 2NFA-3W-1Σ?

For the remaining decision problems on three-way two-dimensional automata, we have a variety of results. In the deterministic case, Kinber obtained positive results for the containment and disjointness problems over a unary alphabet, but negative results for the same problems over general alphabets [36]. In the nondeterministic case over a general alphabet, the containment and disjointness problems are undecidable [22]. A lack of similar results for unary alphabets suggests that the following problem would be worth investigating.

Problem 8. Are the containment and disjointness problems decidable for the class 2NFA-3W-1Σ?

For the equivalence problem in particular, we have only the following results.

Theorem 6.5. *The equivalence problem is decidable for the class 2DFA-3W-1Σ. The equivalence problem is undecidable for the class 2NFA-3W.*

Kinber uses a novel technique to obtain the result for the class 2DFA-3W-1Σ by formulating the problem algebraically and reducing to a solvable system of bilinear equations [36]. This technique extends to Kinber's decidability proofs for the containment and disjointness problems, suggesting that it could be used as a template for similar proofs. Inoue and Takanami proved the result for the class 2NFA-3W [22].

A natural question to ask is whether either of the techniques used in the proof for Theorem 6.5 can be used to prove the remaining unknown results for the equivalence problem.

Problem 9. Is the equivalence problem decidable for the classes 2DFA-3W or 2NFA-3W-1Σ?

Note that Problems 7, 8, and 9 are all related due to the formulation of each decision problem. A positive result for the containment problem would imply a positive result for the equivalence problem, since testing whether $L(\mathcal{A}) = L(\mathcal{B})$ is equivalent to testing whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$ and $L(\mathcal{B}) \subseteq L(\mathcal{A})$. Likewise, a positive result for the equivalence problem would imply a positive result for the universality problem, since the universality problem can be decided by testing whether $L(\mathcal{A}) = \Sigma^{**}$. Therefore, settling Problem 8 positively would also settle Problems 7 and 9 for the class 2NFA-3W-1Σ.

7 Complexity Results

As was mentioned in Section 2.4, the membership problem can be considered to be the most straightforward problem for an abstract machine to solve. Indeed, for an abstract machine as powerful as a four-way two-dimensional automaton, the membership problem is the only decidable problem. Therefore, the membership problem makes a good candidate for investigating complexity properties of that model.

Although many complexity results are known for one-dimensional automata, the membership problem seems to be the only problem studied in this context for two-dimensional automata. In this section, we will review some existing results relating to the complexity of the membership problem for four-way two-dimensional automata. We will also consider a few directions for future research.

7.1 Complexity of Membership

Deciding membership of a given input word for a nondeterministic two-dimensional automaton is a straightforward process. We store the current automaton configuration (state and input head position) on the work tape of a nondeterministic Turing machine, then count the number of configurations we have entered to ensure that the computation is not looping. Since the nondeterministic Turing machine requires only a logarithmic amount of space to store this information, the uniform membership problem is in NL; the nondeterministic variant of the class L.

Lindgren, Moore, and Nordahl alternatively showed that the uniform membership problem for nondeterministic two-dimensional automata is a special case of the graph reachability problem [40]. Since the graph reachability problem is known to be NL-complete [54], we arrive at a similar result.

Remark. Although the result of Lindgren et al. [40] seems to be the first to show that the uniform membership problem for nondeterministic two-dimensional automata is in NL, the

proof of NL-completeness is not novel. The uniform membership problem for nondeterministic one-dimensional automata was known to be NL-hard since 1975 [31]. As a corollary, the same problem in two dimensions is NL-hard. Together with our Turing machine construction, we get the same completeness result for nondeterministic two-dimensional automata.

Lindgren et al. [40] also showed that deciding the uniform membership problem for deterministic two-dimensional automata can be done using a procedure similar to their procedure for the nondeterministic case. The deterministic uniform membership problem is essentially a special case of the graph reachability problem where each node in the graph has outdegree at most one. This variant of the graph reachability problem is L-complete, where L is defined as in Section 2.4.

Remark. Again, the result of Lindgren et al. [40] can be derived from previous results in the literature. The uniform membership problem for deterministic one-dimensional automata was known to be L-complete as early as 1991 [29]. As a result, L-hardness follows for the two-dimensional case, and we can use a similar Turing machine construction to show completeness.

Given that all of these results pertain to the uniform variant of the membership problem, it would be worthwhile to investigate the same questions for the non-uniform variant to determine the differences in the complexity of the problem. In the deterministic case, we can decide non-uniform membership by simulating the computation of the automaton on the input word and keeping track of the current state to ensure loop-freeness. Since this procedure runs in linear time on a two-dimensional Turing machine, we obtain a $O(n^2)$ algorithm for a single-tape one-dimensional Turing machine.

In the nondeterministic case, we do not know of any corresponding procedure. However, since the membership problem is in the class NL, the time complexity of such a procedure would be polynomial. Thus, finding a low-rank polynomial upper bound for the non-uniform membership problem would be desirable.

Problem 10. What is a reasonable upper bound for the non-uniform membership problem for the class 2NFA?

7.2 Complexity of Other Problems

As we saw in Section 6.2, more problems are decidable for three-way two-dimensional automata than for the four-way model. For example, emptiness is decidable for all variants of the three-way model and universality is decidable for deterministic variants. Investigating the complexity of decision problems in the three-way model could give further insight into how restricting the movement of the input head impacts the performance of the automaton.

One of the obvious candidates for study is the emptiness problem for three-way two-dimensional automata. Since the emptiness problem is decidable for both deterministic and nondeterministic three-way variants, we could get complexity results that are more interesting and fine-grained than the corresponding results for the membership problem.

The nonemptiness problem—the problem of determining whether $L(\mathcal{A}) \neq \emptyset$ for some automaton $\mathcal{A}$—is PSPACE-complete for two-way deterministic one-dimensional automata [15]. Clearly, the nonemptiness problem is equivalent to the emptiness problem by switching "yes" and "no" answers. As a corollary, the emptiness problem for three-way two-dimensional automata is PSPACE-hard.

If we restrict ourselves to a unary alphabet, then we can develop a straightforward method for deciding the emptiness problem. Given a three-way two-dimensional automaton $\mathcal{A}$, we construct a two-way one-dimensional automaton $\mathcal{B}$ that operates only on the first row of the input word to $\mathcal{A}$. Then $L(\mathcal{A}) = \emptyset$ if and only if $L(\mathcal{B}) = \emptyset$, and we can test both the emptiness of $L(\mathcal{B})$ and the equivalence of $L(\mathcal{A})$ to $L(\mathcal{B})$ in polynomial space for both deterministic [15] and nondeterministic [7] two-way one-dimensional automata. Therefore, the emptiness problem for three-way two-dimensional automata over a unary alphabet is in PSPACE, so the problem in this case is PSPACE-complete.

Since we know that the emptiness problem is PSPACE-hard for general alphabets and PSPACE-complete for unary alphabets, the natural question to ask is whether we can obtain completeness results for the general alphabet case.

Problem 11. Does the emptiness problem for either the class 2DFA-3W or the class 2NFA-3W belong to PSPACE?

8 Conclusion

Two-dimensional automata are a natural extension of the classical finite automaton model with a number of interesting properties. Despite the fact that the model was introduced in the late 1960s, very little work has been done to investigate the power and applicability of two-dimensional automata in contrast to the vast body of work on similar models (i.e., cellular automata).

Although this report focuses on a few specific operations, decision problems, and complexity measures, the open problems presented in this report are representative of the overall research that remains to be done in this area.

Investigating problems relating to the closure and decidability properties of two-dimensional automata would give a clearer picture of the capabilities of this model of computation. Positive results could then lead to further work on the state complexity of operations or on the time and space complexities of decision problems.

The same types of problems asked in this report could be generalized to obtain results for other novel closure or decidability properties. Certain problems, such as the problem of recognizing unary two-dimensional words with prime dimension given in Section 5.2, seem difficult to solve using only the currently-known techniques. However, there is a possibility that new techniques developed through the study of other questions could be used to settle these more difficult problems.

Finally, it would be worthwhile to investigate applications of two-dimensional automata to domains such as image processing, pattern recognition, or searching. These applied questions would likely only be investigated after the more foundational problems stated in this report are resolved, but such results could serve to introduce the powerful two-dimensional automaton model to a new audience of computer scientists and stimulate further research in this area.

References

[1] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

[2] David A. Mix Barrington, Kevin Compton, Howard Straubing, and Denis Thérien. Regular languages in NC^1. *Journal of Computer and System Sciences*, 44(3):478–499, 1992.

[3] Jean-Camille Birget. Intersection and union of regular languages and state complexity. *Information Processing Letters*, 43(4):185–190, 1992.

[4] Manuel Blum and Carl Hewitt. Automata on a 2-dimensional tape. In R. E. Miller, editor, *Proceedings of the 8th Annual Symposium on Switching and Automata Theory (SWAT 1967)*, pages 155–160, 1967.

[5] Manuel Blum and Carl Hewitt. Automata on a 2-dimensional tape. Technical report AIM-135, Massachusetts Institute of Technology, Cambridge, 1967.

[6] Manuel Blum and William J. Sakoda. On the capability of finite automata in 2 and 3 dimensional space. In P. Young, editor, *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 147–161, 1977.

[7] Sougata Bose, Anca Muscholl, Vincent Penelle, and Gabrielle Puppis. Origin-equivalence of two-way word transducers is in PSPACE, 2018. arXiv:1807.08053.

[8] Symeon Bozapalidis and Antonios Kalampakas. Graph automata. *Theoretical Computer Science*, 393(1–3):147–165, 2008.

[9] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. Self-published, 2008. `http://tata.gforge.inria.fr`.

[10] Jing Dong and Wenbing Jin. Comparison of two-way two-dimensional finite automata and three-way two-dimensional finite automata. In X. Yang, editor, *Proceedings of the 2nd International Conference on Computer Science and Service System (CSSS 2012)*, pages 1904–1906, 2012.

[11] Charles R. Dyer and Azriel Rosenfeld. Triangle cellular automata. *Information and Control*, 48(1):54–69, 1981.

[12] Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173–1187, 2007.

[13] Dora Giammarresi and Antonio Restivo. Two-dimensional languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 215–267. Springer-Verlag, Berlin Heidelberg, 1997.

[14] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, 1979.

[15] Harry B. Hunt III. On the time and tape complexity of languages I. In A. V. Aho, editor, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC 1973)*, pages 10–19, 1973.

[16] Katsushi Inoue and Akira Nakamura. Some properties of two-dimensional on-line tessellation acceptors. *Information Sciences*, 13(2):95–121, 1977.

[17] Katsushi Inoue and Akira Nakamura. Two-dimensional finite automata and unacceptable functions. *International Journal of Computer Mathematics*, 7(3):207–213, 1979.

[18] Katsushi Inoue and Itsuo Takanami. Cyclic closure properties of automata on a two-dimensional tape. *Information Sciences*, 15:229–242, 1978.

[19] Katsushi Inoue and Itsuo Takanami. A note on closure properties of the classes of sets accepted by tape-bounded two-dimensional turing machines. *Information Sciences*, 15:143–158, 1978.

[20] Katsushi Inoue and Itsuo Takanami. Closure properties of three-way and four-way tape-bounded two-dimensional Turing machines. *Information Sciences*, 18(3):247–265, 1979.

[21] Katsushi Inoue and Itsuo Takanami. Three-way tape-bounded two-dimensional Turing machines. *Information Sciences*, 17:195–220, 1979.

[22] Katsushi Inoue and Itsuo Takanami. A note on decision problems for three-way two-dimensional finite automata. *Information Processing Letters*, 10(4–5):245–248, 1980.

[23] Katsushi Inoue and Itsuo Takanami. A note on deterministic three-way tape-bounded two-dimensional Turing machines. *Information Sciences*, 20(1):41–55, 1980.

[24] Katsushi Inoue and Itsuo Takanami. A survey of two-dimensional automata theory. *Information Sciences*, 55(1–3):99–121, 1991.

[25] Katsushi Inoue, Itsuo Takanami, and Akira Nakamura. A note on two-dimensional finite automata. *Information Processing Letters*, 7(1):49–52, 1978.

[26] Katsushi Inoue, Itsuo Takanami, and Akira Nakamura. Nonclosure property of nondeterministic two-dimensional finite automata under cyclic closure. *Information Sciences*, 22(1):45–50, 1980.

[27] Katsushi Inoue, Itsuo Takanami, and Hiroshi Taniguchi. Three-way two-dimensional simple multihead finite automata—closure properties. *Transactions of the Institute of Electronics and Communications Engineers of Japan*, J62-D:273–280, 1979. In Japanese.

[28] Akira Ito, Katsushi Inoue, and Itsuo Takanami. A note on three-way two-dimensional alternating Turing machines. *Information Sciences*, 45:1–22, 1988.

[29] Tao Jiang and Bala Ravikumar. A note on the space complexity of some decision problems for finite automata. *Information Processing Letters*, 40(1):25–31, 1991.

[30] Galina Jirásková and Alexander Okhotin. On the state complexity of operations on two-way finite automata. *Information and Computation*, 253(1):36–63, 2017.

[31] Neil D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11(1):68–85, 1975.

[32] Jarkko Kari. Theory of cellular automata: A survey. *Theoretical Computer Science*, 334(1–3):3–33, 2005.

[33] Jarkko Kari and Cristopher Moore. New results on alternating and non-deterministic two-dimensional finite-state automata. In A. Ferreira and H. Reichel, editors, *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2001)*, volume 2010 of *Lecture Notes in Computer Science*, pages 396–406, Berlin Heidelberg, 2001. Springer-Verlag.

[34] Jarkko Kari and Cristopher Moore. Rectangles and squares recognized by two-dimensional automata. In J. Karhumäki, H. Maurer, G. Paun, and G. Rozenberg, editors, *Theory is Forever: Essays Dedicated to Arto Salomaa on the Occasion of His 70th Birthday*, volume 3113 of *Lecture Notes in Computer Science*, pages 134–144, Berlin Heidelberg, 2004. Springer-Verlag.

[35] Jarkko Kari and Ville Salo. A survey on picture-walking automata. In W. Kuich and G. Rahonis, editors, *Algebraic Foundations in Computer Science: Essays Dedicated to Symeon Bozapalidis on the Occasion of His Retirement*, volume 7020 of *Lecture Notes in Computer Science*, pages 183–213, Berlin Heidelberg, 2011. Springer-Verlag.

[36] Efim B. Kinber. Three-way automata on rectangular tapes over a one-letter alphabet. *Information Sciences*, 35:61–77, 1985.

[37] Michal Kunc and Alexander Okhotin. Making graph-walking automata reversible. Technical report 1042, Turku Centre for Computer Science, Turku, 2012.

[38] Michal Kunc and Alexander Okhotin. Reversibility of computations in graph-walking automata. In K. Chatterjee and J. Sgall, editors, *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS 2013)*, volume 8087 of *Lecture Notes in Computer Science*, pages 595–606, Berlin Heidelberg, 2013. Springer-Verlag.

[39] Ernst Leiss. Succinct representation of regular languages by boolean automata. *Theoretical Computer Science*, 13(3):323–330, 1981.

[40] Kristian Lindgren, Cristopher Moore, and Mats Nordahl. Complexity of two-dimensional patterns. *Journal of Statistical Physics*, 91(5/6):909–951, 1998.

[41] Oleg B. Lupanov. O sravnenii dvukh tipov konechnykh istochnikov. *Problemy Kibernetiki*, 9:328–335, 1963. In Russian.

[42] Albert R. Meyer and Michael J. Fischer. Economy of description by automata, grammars, and formal systems. In F. C. Hennie, editor, *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT 1971)*, pages 188–191, 1971.

[43] David L. Milgram. A region crossing problem for array-bounded automata. *Information and Control*, 31(2):147–152, 1976.

[44] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, 1967.

[45] Frank R. Moore. On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Transactions on Computers*, C-20(10):1211–1214, 1971.

[46] John Mylopoulos. On the recognition of topological invariants by 4-way finite automata. *Computer Graphics and Image Processing*, 1(3):308–316, 1972.

[47] Holger Petersen. Some results concerning two-dimensional Turing machines and finite automata. In H. Reichel, editor, *Proceedings of the 10th International Conference on Fundamentals of Computation Theory (FCT 1995)*, volume 965 of *Lecture Notes in Computer Science*, pages 374–382, Berlin Heidelberg, 1995. Springer-Verlag.

[48] Giovanni Pighizzini. Two-way finite automata: Old and recent results. *Fundamenta Informaticae*, 126(2–3):225–246, 2013.

[49] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[50] Azriel Rosenfeld. *Picture Languages: Formal Models for Picture Recognition*. Computer Science and Applied Mathematics. Academic Press, New York, 1979.

[51] Azriel Rosenfeld and David L. Milgram. Parallel/sequential array automata. *Information Processing Letters*, 2(2):43–46, 1973.

[52] Azriel Rosenfeld and Rani Siromoney. Picture languages—a survey. *Languages of Design*, 1(3):229–245, 1993.

[53] Ville Salo. Classes of picture languages defined by tiling systems, automata and closure properties. Master's thesis, Turku Centre for Computer Science, 2011.

[54] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[55] Shigeko Seki. Real-time recognition of two-dimensional tapes by cellular automata. *Information Sciences*, 19(3):179–198, 1979.

[56] John C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.

[57] Michael Sipser. Halting space-bounded computations. *Theoretical Computer Science*, 10(3):335–338, 1980.

[58] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, Boston, 1997.

[59] Rani Siromoney. Advances in array languages. In H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors, *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, volume 291 of *Lecture Notes in Computer Science*, pages 549–563, Berlin Heidelberg, 1986. Springer-Verlag.

[60] Rani Siromoney. Array languages and Lindenmayer systems —a survey. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pages 413–426. Springer-Verlag, Berlin Heidelberg, 1986.

[61] Andrzej Szepietowski. On three-way two-dimensional Turing machines. *Information Sciences*, 47(2):135–147, 1989.

[62] Andrzej Szepietowski. Some remarks on two-dimensional finite automata. *Information Sciences*, 63(1–2):183–189, 1992.

[63] Kenichi Taniguchi and Tadao Kasami. Some decision problems for two-dimensional nonwriting automata. *Transactions of the Institute of Electronics and Communications Engineers of Japan*, 54-C(7):578–585, 1971. In Japanese.

[64] Sheng Yu, Qingyu Zhuang, and Kai Salomaa. The state complexities of some basic operations on regular languages. *Theoretical Computer Science*, 125(2):315–328, 1994.