

Two-Way Deterministic Finite Automata (2DFA)

Descriptional Advantages and Reductions to One-Way Automata

Falco Wolf

January 24, 2026

Abstract

Two-way deterministic finite automata (2DFA) are classical finite automata whose reading head may move both to the left and to the right on a read-only input tape. Although 2DFA recognize exactly the regular languages, allowing two-way head motion can make descriptions significantly more succinct than with ordinary one-way DFA and NFA. This seminar paper pursues two main goals. First, we illustrate how 2DFA can describe certain regular languages in a more natural or much more compact way, for example for suffix-dependent and modular properties. Second, we present constructions that simulate 2DFA by one-way automata (1NFA and 1DFA) and discuss the resulting blow-up in the number of states and its consequences for descriptional complexity.

Contents

1	Introduction	3
1.1	Motivation	3
2	Preliminaries	4
2.1	Words and Languages	4
2.2	One-Way DFA and NFA	4
3	Two-Way Deterministic Finite Automata	5
3.1	Formal Definition	5
3.2	Configuration and Acceptance	5
3.3	End-Markers	5

4 Why 2DFA Can Be Better Descriptions	6
4.1 Introduction	6
4.2 Suffix-Dependent Languages	6
4.3 Divisibility / Modular Properties	8
4.4 Known Succinctness Separations (optional)	9
5 Reductions: From 2DFA to One-Way Automata	9
5.1 Introduction	9
5.2 Reduction from 2DFA to 1NFA	9
5.2.1 Lemma for Rejection Condition	9
5.2.2 Construction of the 1NFA	10
5.2.3 Correctness of the Construction	10
5.3 Reduction from 2DFA to 1DFA	12
5.4 State Blow-Up and Complexity Discussion	12
5.5 Worked Example (optional)	13
6 Conclusion	13
A Appendix	14

1 Introduction

1.1 Motivation

Finite automata are one of the most fundamental models in automata theory. In their classical form, deterministic and nondeterministic one-way automata scan the input from left to right in a single pass. Two-way deterministic finite automata (2DFA) extend this model by allowing the read head to move both to the left and to the right on a read-only input tape. Depending on the problem end/start-marker can be added to the input word. With this extend the automata can detect the left and right end of the input. At first sight, this additional freedom might suggest that 2DFA are strictly more powerful than ordinary one-way automata.

However, it is a classical result that two-way finite automata still recognize exactly the regular languages. From the point of view of *expressive power*, nothing is gained by allowing the head to move in both directions. The situation changes completely when one looks at *descriptive complexity*, that is, at the number of states needed to describe a given regular language. Here, two-way motion can make automata significantly more succinct than their one-way counterparts.

This effect is particularly visible for languages whose definition depends on properties of the *suffix* of the input or on *modular* conditions on the word length. For instance, when checking whether a word ends with a fixed pattern, a one-way automaton must implicitly remember the last few input symbols while it moves forward, leading to a number of states that grows with the pattern length. A 2DFA, in contrast, can first move to the right end-marker and then verify the suffix locally while moving left, often using far fewer states. Similar phenomena occur for languages such as “the length of the word is divisible by m ”, where the structure of a succinct two-way automaton can reflect the arithmetic structure of m .

The goal of this seminar paper is twofold. First, we present several examples that illustrate how 2DFA can provide more natural or much more compact descriptions than one-way DFA and NFA, focusing in particular on suffix-dependent and modular properties. Second, we discuss constructions that simulate 2DFA by one-way automata (1NFA and 1DFA) and analyze the resulting blow-up in the number of states. In this way, we relate the descriptive advantages of 2DFA to classical one-way models and highlight the complexity trade-offs between them.

2 Preliminaries

We mostly follow the standard definitions and notation from [5, 1].

2.1 Words and Languages

Fix an alphabet Σ throughout the paper. The set of all finite words over Σ is denoted by Σ^* , the empty word by ε . The length of a word w is $|w|$. A language L over Σ is a subset of Σ^* .

2.2 One-Way DFA and NFA

Definition 1 (1DFA / 1NFA). *A one-way finite automaton is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function, $Q_0 \subseteq Q$ is the set of start states, and $F \subseteq Q$ is the set of accepting states. The automaton A is deterministic iff $|Q_0| = 1$ and $\forall q \in Q \ \forall \sigma \in \Sigma \ |\delta(q, \sigma)| \leq 1$. A deterministic one-way finite automaton is abbreviated as 1DFA, a nondeterministic one-way finite automaton as 1NFA.*

A configuration of a 1DFA/1NFA on input $w = w_1 w_2 \dots w_n \in \Sigma^*$ is a pair (q, i) with $q \in Q$ and $0 \leq i \leq n$, where i indicates the position of the reading head. In the one-way model, the head moves one position to the right in each step.

A run of A on input w is a finite sequence of configurations

$$(q_0, i_0), (q_1, i_1), \dots, (q_m, i_m)$$

such that $q_0 \in Q_0$, $i_0 = 0$, and for every $0 \leq t < m$ we have

$$i_{t+1} = i_t + 1 \quad \text{and} \quad q_{t+1} \in \delta(q_t, w_{i_{t+1}}).$$

The input w is accepted by A iff there exists a run

$$(q_0, i_0), (q_1, i_1), \dots, (q_m, i_m)$$

with $q_m \in F$ and $i_m = n$. Let $L(A)$ denote the language recognized by A .

Definition 2 (Regular Language). *A language L over Σ is called regular iff there exists a 1DFA A such that $L = L(A)$.*

Theorem 1 (Subset Construction). *For every 1NFA A there exists a 1DFA B such that $L(B) = L(A)$.*

Theorem 1 is standard; see [5, 1]. We can conclude that 1DFA, 1NFA, and regular languages are equivalent in terms of expressive power.

3 Two-Way Deterministic Finite Automata

3.1 Formal Definition

The definitions of 2DFA and 2NFA are similar to those of the one-way finite automata. We only need to adjust the transition function and the head movement. We use a definition based on [6].

Definition 3 (2DFA / 2NFA). *A two-way finite automaton is a tuple $A = (\Sigma, S, S_0, \rho, F)$, where S is a finite set of states, Σ is a finite input alphabet, $\rho : S \times \Sigma \rightarrow 2^S \times \{-1,0,1\}$ is the transition function, $S_0 \subseteq S$ is the set of start states, and $F \subseteq S$ is the set of accepting states. The automaton A is deterministic iff $|S_0| = 1$ and $\forall s \in S \ \forall \sigma \in \Sigma \ |\rho(s, \sigma)| \leq 1$. A deterministic two-way finite automaton is abbreviated as 2DFA, a nondeterministic two-way finite automaton as 2NFA.*

3.2 Configuration and Acceptance

We describe the computation of A in terms of configurations and runs. A *configuration* is a pair $(s, j) \in S \times \mathbb{N}$, where s denotes the current state and j the current head position. A *run* on an input word is a finite sequence of configurations.

Let $w = a_0a_1 \cdots a_{n-1} \in \Sigma^*$. A sequence $(s_0, j_0), \dots, (s_m, j_m)$ is a run of A on w if $0 \leq j_m \leq n$ and for every i with $0 \leq i < m$ we have $0 \leq j_i < n$ and there exists a pair $(t, k) \in \rho(s_i, a_{j_i})$ such that $s_{i+1} = t$ and $j_{i+1} = j_i + k$. The run is *accepting* if $s_0 \in S_0$, $j_0 = 0$, $j_m = n$, and $s_m \in F$. The automaton A accepts w iff it has an accepting run on w . [7]

3.3 End-Markers

In many constructions it is convenient to assume explicit boundary symbols. Let b and e be two fixed symbols (left and right endmarker, respectively) with $b, e \notin \Sigma$. A *two-way automaton with endmarkers* $A = (\Sigma, S, S_0, \rho, F)$ is defined as before, except that the transition function may also read the endmarkers, i.e.,

$$\rho : S \times (\Sigma \cup \{b, e\}) \rightarrow 2^{S \times \{-1,0,1\}}.$$

On an input $w \in \Sigma^*$, the tape content is bwe . The automaton accepts w iff it has an accepting run over bwe . If only a left (resp. right) endmarker is available, then the input is presented as bw (resp. we), and acceptance is defined analogously. [7]

We will explicitly state when we assume a left and/or right endmarker. In

the examples on suffix-dependent and modular properties we use both end-markers to simplify the presentation. In contrast, the simulations of two-way automata by one-way automata are presented without endmarkers.

Note that adding endmarkers does not increase the expressive power of two-way automata; see [7] (building on classical results such as [4]).

4 Why 2DFA Can Be Better Descriptions

4.1 Introduction

In this section we present examples of regular languages where 2DFA can provide more compact descriptions than one-way DFA and NFA, focusing in particular on suffix-dependent and modular properties. We will see that for certain languages the descriptive complexity reduces significantly when using two-way motion.

4.2 Suffix-Dependent Languages

We follow an example from Pighizzini [3] to prove the advantages of a 2DFA in contrast to a 1DFA for a specific suffix-dependent language. We will also talk about 1NFA's in this context.

Example 1 (Suffix language). *Let*

$$L_n := \{x \in \{0, 1\}^* \mid \text{the } n\text{-th symbol from the right is } 1\}$$

for $n \geq 1$.

When using two-way automata with both endmarkers, we can construct a 2DFA with $n + 3$ states, i.e. $O(n)$, for L_n as follows:

- The automaton starts in the initial state q_0 and moves right until it reads the right endmarker.
- Then it moves left n positions, using states q_1, \dots, q_n to count the number of steps.
- Finally, it checks whether the current symbol is 1 or 0 and accepts or rejects accordingly using 2 states.

We conclude that a 2DFA with $O(n)$ states can recognize L_n . Lets now consider a 1DFA for the same language. One idea to construct such a 1DFA is the following: keep track of the last n symbols with different states. Therefore we need states $\{0, 1\}^n$, which gives us a total of 2^n states. When reading a

new symbol, the automaton updates its state to reflect the last n accordingly. Accepting states are those where the first symbol of the last n is 1.

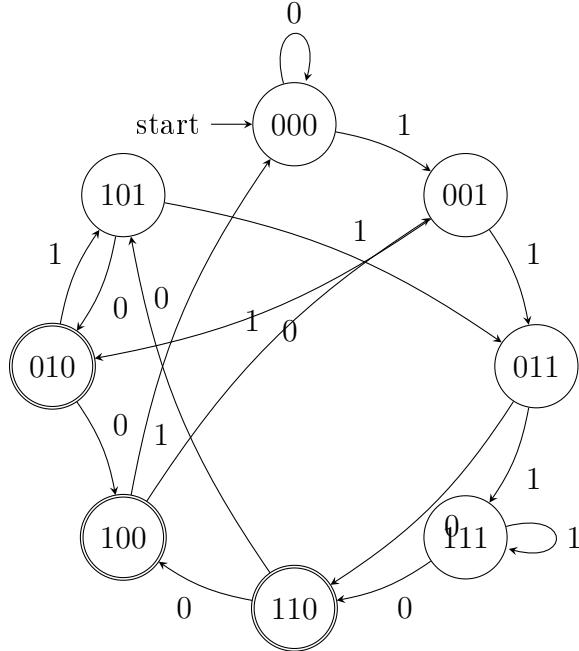


Figure 1: A 1DFA for L_3 . The state stores the last three input symbols; accepting states are those whose leftmost bit is 1.

We can conclude that we can construct a 1DFA for L_n with 2^n states. Next we will prove that this is asymptotically optimal.

Theorem 2. *Let*

$$L_n := \{x \in \{0,1\}^* \mid \text{the } n\text{-th symbol from the right is 1}\}$$

for $n \geq 1$. Any 1DFA deciding L_n requires $\Omega(2^n)$ states.

Proof. Let M be the 1DFA deciding L_n with transition function δ and starting state q_0 .

Let $u, v \in \{0,1\}^n$ be two different strings and let $i \in \{0, \dots, n-1\}$ be the position in which the strings differ first (from the left).

$$w := 0^{n-1-i}$$

We can conclude that one of the words uw, vw is accepted and the other is rejected.

Assume $\delta(q_0, u) = \delta(q_0, v)$ then also $\delta(q_0, uw) = \delta(q_0, vw)$ because M is deterministic.

\Rightarrow contradiction $\Rightarrow \delta(q_0, u) \neq \delta(q_0, v)$

As a result, every string of length n requires a different state.

$\Rightarrow \Omega(2^n)$ different states.

□

This proves that the 1DFA we constructed before is asymptotically optimal. So we have shown that for the suffix-dependent language L_n a 2DFA uses $O(n)$ states while a 1DFA, asymptotically optimal in terms of states, requires 2^n states. The 2DFA has an exponential advantage in terms of descriptional complexity.

We can also construct a 1NFA for L_n with $n + 1$ states. Nondeterminism allows us to guess the position that will become the n-th from the right. This allows us to construct a 1NFA with only $n + 1$ states, as shown in Figure 2.

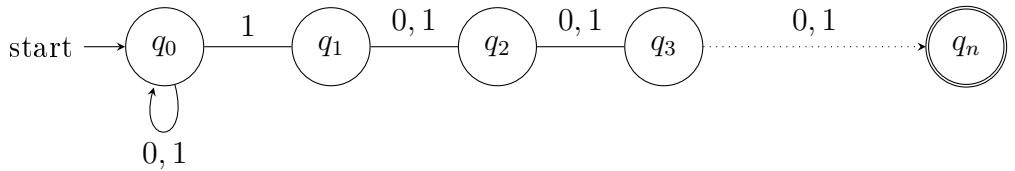


Figure 2: A 1NFA accepting the language L_n (adapted from [3, Fig. 1]).

So there is no improvement in terms of descriptional complexity when using a 2DFA instead of a 1NFA for the suffix-dependent language L_n .

4.3 Divisibility / Modular Properties

Example 2 (Length divisible by m). $L_m = \{ w \in \{a\}^* \mid |w| \equiv 0 \pmod{m} \}$.

What to cover:

- Minimal 1DFA has m states (standard result).
- 2DFA with fewer states when splitting into primes.
- Special cases to reduce states further (e.g., even/odd).

4.4 Known Succinctness Separations (optional)

There are families of regular languages where 2DFA are exponentially smaller than equivalent 1DFA, and cite references. Maybe too much?

5 Reductions: From 2DFA to One-Way Automata

5.1 Introduction

At first sight, it may seem that allowing the head to move left increases the power of finite automata beyond the regular languages. However, it is a classical result that 2DFA still recognize exactly the regular languages; hence there is no increase in expressive power.

In this section we present two constructions that simulate a given 2DFA by a one-way automaton and analyze the resulting blow-up in the number of states. The first construction yields a one-way NFA for the complement language, and the second uses crossing-sequence analysis to convert a 2DFA into an equivalent one-way DFA. Our presentation follows [6].

There are also proofs of the equivalence based on regular expressions; see, e.g., [2]. We do not cover this approach here.

5.2 Reduction from 2DFA to 1NFA

We follow the construction from [6].

5.2.1 Lemma for Rejection Condition

Before presenting the construction, we need a lemma.

Lemma 1 (Vardi [6]). (Statement quoted verbatim from [6].) *Let $A = (\Sigma, S, S_0, \rho, F)$ be a two-way automaton and let $w = a_0a_1 \cdots a_n \in \Sigma^*$. The automaton A rejects w iff there exist sets $T_0, \dots, T_{n+1} \subseteq S$ such that:*

1. $S_0 \subseteq T_0$,
2. $T_{n+1} \cap F = \emptyset$, and
3. for every $0 \leq i \leq n$, every $s \in T_i$, and every transition $(s', k) \in \rho(s, a_i)$ with $i + k > 0$, we have $s' \in T_{i+k}$.

Proof. (\Rightarrow) Suppose A rejects w . For each $i \in \{0, \dots, n+1\}$ define

$$T_i := \{s \in S \mid \text{there exists a run of } A \text{ on } w \text{ that reaches a configuration } (s, i)\}.$$

For every $s \in S_0$ there is a (trivial) run starting in configuration $(s, 0)$, hence $S_0 \subseteq T_0$. Since A rejects w , no accepting configuration at position $n + 1$ is reachable. Therefore $T_{n+1} \cap F = \emptyset$. Finally, let $0 \leq i \leq n$, let $s \in T_i$, and let $(s', k) \in \rho(s, a_i)$ with $i + k > 0$. By definition of T_i , there exists a run reaching (s, i) . Extending this run by one transition yields a run reaching $(s', i + k)$, hence $s' \in T_{i+k}$.

(\Leftarrow) Suppose there exist sets $T_0, \dots, T_{n+1} \subseteq S$ satisfying the three conditions. Assume for contradiction that A accepts w . Then there exists a run $(s_0, j_0), \dots, (s_m, j_m)$ with $s_0 \in S_0$, $j_0 = 0$, $j_m = n+1$, and $s_m \in F$. By (1), we have $s_0 \in T_{j_0} = T_0$. Using (3) inductively along the run, we get $s_i \in T_{j_i}$ for all i , and in particular $s_m \in T_{j_m} = T_{n+1}$. This contradicts (2) since $s_m \in F$. Hence, A rejects w . \square

5.2.2 Construction of the 1NFA

The properties of the lemma contain local conditions and can therefore be checked by a one-way automaton while scanning the input. We implement these checks by constructing a 1NFA for the complement language. Assume there is a 2DFA $A = (\Sigma, S, S_0, \rho, F)$, we then construct a 1NFA $B = (\Sigma, Q, Q_0, \delta, G)$ deciding $\Sigma^* \setminus L(A)$. The state set Q is $2^S \cup (2^S)^2$, i.e., sets of states and pairs of sets of states. The starting state set Q_0 is $\{T : S_0 \subseteq T \subseteq S\}$, i.e., the collection of state sets that contain S_0 . The accepting state set G is

$$\{T : T \cap F = \emptyset\} \cup \{(T, U) : U \cap F = \emptyset\},$$

i.e., the collection of sets that do not intersect F and pairs of sets whose second component does not intersect F .

It remains to define the transition function δ . We have $(T, U) \in \delta(T, a)$ if the following holds:

- If $s \in T$ and $(t, 0) \in \rho(s, a)$, then $t \in T$, and
- if $s \in T$ and $(t, 1) \in \rho(s, a)$, then $t \in U$.

We have $(U, V) \in \delta((T, U), a)$ if the following holds:

- If $s \in U$ and $(t, -1) \in \rho(s, a)$, then $t \in T$,
- if $s \in U$ and $(t, 0) \in \rho(s, a)$, then $t \in U$, and
- if $s \in U$ and $(t, 1) \in \rho(s, a)$, then $t \in V$.

5.2.3 Correctness of the Construction

The following lemma helps to proof the correctness of the construction.

Lemma 2. Let $A = (\Sigma, S, S_0, \rho, F)$ be a two-way automaton, $B = (\Sigma, Q, Q_0, \delta, G)$ be a 1NFA constructed as above and $w = a_0a_1\cdots a_n \in \Sigma^*$. Then B accepts w iff there exist sets $T_0, \dots, T_{n+1} \subseteq S$ satisfying the conditions lemma 1.

Proof. (\Rightarrow) Suppose B accepts w . Then there exists an accepting path C_0, C_1, \dots, C_{n+1} with $n \in \mathbb{N}$.

Set $T_0 := C_0 \in 2^S$.

Since $C_1 \in \delta(T_0, a_0)$, we have $C_1 = (T_0, T_1)$ with $T_1 \in 2^S$.

...

Since $C_{n+1} \in \delta((T_{n-1}, T_n), a_n)$, we have $C_{n+1} = (T_n, T_{n+1})$ with $T_{n+1} \in 2^S$.

The accepting path rewritten:

$$T_0, (T_0, T_1), \dots, (T_n, T_{n+1})$$

with $T_i \in 2^S$.

We can now proof that T_0, \dots, T_{n+1} has the properties of lemma 1:

1. $T_0 \in Q_0 \Rightarrow S_0 \subset T_0$.
2. $(T_n, T_{n+1}) \in G \Rightarrow T_{n+1} \cap F = \emptyset$.
3. ($i = 0$) We have that $(T_0, T_1) \in \delta(T_0, a_0)$. For every $s \in T_0$ and $(s', k) \in \rho(s, a_i)$ with $i + k \geq 0$ the following holds:

$$k \geq 0.$$

If $k = 0$, then $s' \in T_0 = T_{i+k}$ (because of the construction of B). If $k = 1$, then $s' \in T_1 = T_{i+k}$ (because of the construction of B). Hence $s' \in T_{i+k}$.

($i > 0$) We have that $(T_i, T_{i+1}) \in \delta((T_{i-1}, T_i), a_i)$. For every $s \in T_i$ and $(s', k) \in \rho(s, a_i)$ with $i + k \geq 0$ the following holds:

- if $k = -1$, then $s' \in T_{i-1} = T_{i+k}$ (because of the construction of B),
- if $k = 0$, then $s' \in T_{i+0} = T_{i+k}$ (because of the construction of B),
- if $k = 1$, then $s' \in T_{i+1} = T_{i+k}$ (because of the construction of B).

Hence $s' \in T_{i+k}$.

T_0, \dots, T_{n+1} sequence exists as defined in Lemma 1.

(\Leftarrow) Suppose there exists a sequence T_0, \dots, T_{n+1} as defined in Lemma 1. We proof that $T_0, (T_0, T_1), \dots, (T_n, T_{n+1})$ is a accepting path for w in B .

We use property 1 $S_0 \subset T_0 \Rightarrow T_0 \in Q_0$. We use property 2 $T_{n+1} \cap F = \emptyset \Rightarrow (T_n, T_{n+1}) \in G$.

Now we proof that there is a transition between the states:

- ($i = 0$) $(T_0, T_1) \in \delta(T_0, a_0)$, because of property 3.
- ($i > 0$) $(T_i, T_{i+1}) \in \delta((T_{i-1}, T_i), a_i)$, because of property 3.

□

Theorem 3 (Vardi [6]). (Statement quoted verbatim from [6].) *Let A be a two-way automaton with n states. Then there is a one-way automaton B with $O(\exp n)$ states such that $L(B) = \Sigma^* - L(A)$. !!Its $O((\exp n)^2)$ states!!*

Proof. Let A be a two-way automaton and let B be the 1NFA obtained from A by the construction above. Let $w \in \Sigma^*$.

$w \notin L(A) \Leftrightarrow$ There exists a sequence T_0, \dots, T_{n+1} satisfying the conditions of Lemma 1 (Lemma 1). $\Leftrightarrow w \in L(B)$ (Lemma 2). □

5.3 Reduction from 2DFA to 1DFA

Crossing sequence analysis.

Theorem 4. (Statement quoted verbatim from [6].) *Let A be a two-way automaton with n states. Then there is a one-way deterministic automaton B with $O(\exp n)$ states such that $L(B) = L(A)$.*

5.4 State Blow-Up and Complexity Discussion

The construction in Chapter 5.2 by Vardi [6] yields a 1NFA with $O((\exp n)^2)$ states for a given 2DFA with n states. The 1NFA automata decides the complement language. We can convert the 1NFA to a 1DFA, which results into a exponentially blow-up of states. The resulting 1DFA has $O(\exp((\exp n)^2))$ states and decides the complement language. We can flip the accepting and non-accepting states to get a 1DFA for the original language, without increasing the number of states. This trick doesnt work for the 1NFA, because of nondeterminism. As a result we get a 1DFA with $O(\exp((\exp n)^2))$ states for a given 2DFA with n states.

The construction in Chapter 5.3 by Vardi [6] based on Shepherdson [4] with crossing sequences yields a 1DFA with $O(\exp n)$ states for a given 2DFA with n states.

To solve the task: "Given a two-way automata with n states, construct an equivalent one-way automata." Theorem 4 is the better choice, because of less state blow-up.

Also the task: "Given a two-way automata with n states, construct a one-way automata deciding the complement." Theorem 4 is the better choice, because $O(\exp n)$ is better then a blowup of $O(\exp(n)^2)$.

5.5 Worked Example (optional)

Show one small 2DFA and (part of) its converted 1NFA/1DFA to make the construction concrete.

Maybe not possible, because of size?

6 Conclusion

Short wrap-up tied back to the goals.

References

- [1] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, 3 edition, 2006.
- [2] Mans Hulden. From two-way to one-way finite automata—three regular expression-based methods. In Frank Drewes, editor, *Implementation and Application of Automata (CIAA 2015)*, volume 9223 of *Lecture Notes in Computer Science*, pages 176–187, Cham, 2015. Springer.
- [3] Giovanni Pighizzini. Two-way finite automata: Old and recent results. In Enrico Formenti, editor, *Proceedings 18th international workshop on Cellular Automata and Discrete Complex Systems and 3rd international symposium Journées Automates Cellulaires, AUTOMATA & JAC 2012, La Marana, Corsica, September 19-21, 2012*, volume 90 of *EPTCS*, pages 3–20, 2012.
- [4] J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, April 1959.
- [5] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3 edition, 2012.
- [6] Moshe Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30(5):261–264, 1989.
- [7] Moshe Y. Vardi. Endmarkers can make a difference. *Information Processing Letters*, 35(3):145–148, July 1990.

A Appendix

Extra proofs, larger automata diagrams, additional examples.