Positional Simulation of Two-Way Automata: Proof of a Conjecture of R. Kannan and Generalizations

JEAN-CAMILLE BIRGET

Computer Science and Engineering Department, University of Nebraska, Lincoln, Nebraska 68588-0115

Received June 6, 1989

R. Kannan conjectured that every non-deterministic two-way finite automaton can be positionally simulated by a deterministic two-way finite automaton. The conjecture is proved here by reduction to a similar problem about finite monoids. The method and the result are then generalized to alternating two-way finite automata, certain alternating one-tape linear-time Turing machines, and one-pebble automata. © 1992 Academic Press, Inc.

1. INTRODUCTION

In [K], Ravindran Kannan introduces the following notion:

**Definition.** Let $A_1$ and $A_2$ be two two-way finite automata (possibly non-deterministic or alternating), with input alphabet $\Sigma$. Then $A_1$ is **positionally simulated** by $A_2$ iff we have: For every state $q_1$ of $A_1$ there exists a state $q_2$ of $A_2$ such that for all $w \in \Sigma^*$ and all $i$ with $0 \leq i \leq |w|$: when $A_1$ and $A_2$ are both started on input $w$ at position $i$ in their respective states $q_1$ and $q_2$, then they both accept or both reject.

Equivalently, $(\forall q_1)(\exists q_2)(\forall w \in \Sigma^*)(\forall u, v \in \Sigma^* with w = uv): A_1$ with initial configuration $u q_1 v$ accepts $\iff A_2$ with initial configuration $u q_2 v$ accepts.

**Kannan's Conjecture** [K]. For every non-deterministic two-way finite automaton there exists a deterministic two-way finite automaton which positionally simulates it.

This conjecture came up in the study of space complexity. One important question is how NSPACE($S$) is related to DSPACE($S$), for a space-constructible function $S$. In particular, the question whether NSPACE(log) $\neq$ DSPACE(log) has been studied, but remains open. A non-uniform version of this problem is whether every non-deterministic two-way finite automaton is equivalent (in the usual sense) to a deterministic two-way finite automaton, with only polynomially many more states; Sakoda and Sipser conjecture that the answer is negative (see [SS, S]).

Kannan [K] considers those problems, but he uses a stronger notion of equivalence (namely, positional simulation). He proves (among other results):
Theorem [K]. Let \( f: \mathbb{N} \to \mathbb{N} \) be such that every non-deterministic Turing machine with any space complexity \( S(\cdot) \) can be positionally simulated by a deterministic Turing machine with space complexity \( \leq f(S(\cdot)) \). Here \( S(\cdot) \) is restricted to be fully space-constructible, and to satisfy \( S(n) \geq \frac{1}{2} \log \log n \) infinitely often. Then for all \( n \) large enough: \( f(n) \geq 2^\left(\log n\right)^{\lfloor \log \log n \rfloor} \).

Theorem [K]. Let \( g: \mathbb{N} \to \mathbb{N} \) be such that every non-deterministic two-way finite automaton with \( n \) states can be positionally simulated by a deterministic two-way finite automaton with at most \( g(n) \) states. Then for all \( n \) large enough: \( g(n) \geq 2^\left(\log n\right)^{\lfloor \log \log n \rfloor} \).

Thus Kannan proves a lower bound for \( g(n) \); but he does not prove that \( g(n) \) is always finite. The conjecture that \( g(n) \) is always finite is precisely Kannan’s conjecture, stated earlier.

2. Definitions and Basics

In this paper, for technical reasons, I will use a model of two-way automata that is slightly different from the usual one (as used, e.g., in [HU]); but the two models can (positionally) simulate each other.

Definition 2.1. A two-way finite automaton is given by \((\bar{Q}, \bar{Q}, \Sigma \cup \{\#\}, \bullet, q_0, F)\), where: \( \Sigma \) is the input alphabet, \( \# (\notin \Sigma) \) is the endmarker of the tape, \( \bar{Q} \) (resp. \( \bar{Q} \)) is the set of right-(resp. left-) moving states, \( q_0 \in \bar{Q} \cup \bar{Q} \) is the start-state, \( F \subseteq \bar{Q} \cup \bar{Q} \) is the set of accept states, and \( \bullet: (q, a) \in (\bar{Q} \cup \bar{Q}) \times (\Sigma \cup \{\#\}) \to q \cdot a \subseteq \bar{Q} \cup \bar{Q} \) is the next-state relation. All sets are assumed to be finite.

Non-determinism arises when \( \bullet \) is a relation (not just a partial function), or when \( \bar{Q} \cap \bar{Q} \neq \emptyset \).

A configuration (or “instantaneous description”) of the automaton will be of the form \( uqv \), with \( u, v \in (\Sigma \cup \{\#\})^* \) and \( q \in \bar{Q} \cup \bar{Q} \). We picture such a configuration by drawing the reading head exactly at the boundary between \( u \) and \( v \). So the head is between cells, not on a cell (as in the usual model).

When the automaton is currently in configuration \( \ldots a_{k-1} a_k q a_{k+1} a_{k+2} \ldots \), with \( q \in \bar{Q} \cup \bar{Q} \) then a next configuration must be of the form:

- If \( q \in \bar{Q} \), we obtain \( \ldots a_{k-1} a_k q' a_{k+1} a_{k+2} \ldots \), with \( q' \in q \cdot a_{k+1} \) (i.e., the reading head moves right, reads \( a_{k+1} \), and applies the next-state relation \( \bullet \) to \( q \) and \( a_{k+1} \)).
- If \( q \in \bar{Q} \), we obtain \( \ldots a_{k-1} q'' a_k a_{k+1} a_{k+2} \ldots \), with \( q'' \in q \cdot a_k \) (i.e., the reading head moves left, reads \( a_k \), and applies \( \bullet \) to \( q \) and \( a_k \)).

Thus the current state alone determines whether the head will next move left or right.

This model of a two-way automaton (see [B2]) can simulate the classical model (see [HU], where the direction of the next move depends on the current state and
the content of the cell next to the head): the simulating automaton of this model first visits the left neighboring cell of its current position; if the classical automaton went left, the simulating automaton now stays in this position (in the new state); if the classical automaton went right, the simulating automaton will now go two steps to the right (using an intermediary state set \( Q \times \{1, 2\} \)) and then goes into the new state. (The state set \( Q \) is replaced by \( Q \cup Q \times \{1, 2\} \)).

**Remark Concerning the End Marker \( \# \).** It is useful to consider configurations which contain any number of occurrences of \( \# \), or in which \( \# \) does not occur at all. The conventions are:

1. When the reading head “falls off the tape,” the computation halts.
2. When an end marker has just been read then either the computation halts (i.e., no next state is defined with respect to \( \bullet \)), or the reading head immediately moves back over the end marker.

**Definition 2.2 (from [B2]).** Consider a fixed (non-deterministic) two-way automaton \( A \). To every input word \( w \in (\Sigma \cup \{\#\})^* \) we associate four global state transition relations \( \{\rightarrow w\rightarrow\}, \{\leftarrow w\leftarrow\}, \{\Rightarrow w\Rightarrow\}, \{\Leftarrow w\Leftarrow\} \), as follows:

The graph of the relation \( \{\rightarrow w\rightarrow\} \) is a subset of \( \mathcal{Q} \times \mathcal{Q} \) defined by:

\[(q, q') \in \{\rightarrow w\rightarrow\} \text{ iff there exists a computation of the automaton } A \text{ which starts in configuration } qw \text{ (where } q \in \mathcal{Q} \text{) and reaches the configuration } wq' \text{ (where } q' \in \mathcal{Q} \text{).}\]

Similarly \( \{\Rightarrow w\Rightarrow\} \) is a subset of \( \mathcal{Q} \times \mathcal{Q} \), and

\[(q, q') \in \{\Rightarrow w\Rightarrow\} \text{ iff there exists a computation of the two-way automaton } A \text{ starting with configuration } qw \text{ (where } q \in \mathcal{Q} \text{) and reaching the configuration } q'w \text{ (where } q' \in \mathcal{Q} \text{).}\]

Symmetrically one defines \( \{\Leftarrow w\Leftarrow\} = \{(q, q') \in \mathcal{Q} \times \mathcal{Q} \mid \text{there exists a computation starting with } wq \text{ and reaching } wq'\} \), and

\[\{\leftarrow w\leftarrow\} = \{(q, q') \in \mathcal{Q} \times \mathcal{Q} \mid \text{there exists a computation starting with } wq \text{ and reaching } q'w\}.\]

We will denote \( \{w\} = (\{\rightarrow w\rightarrow\}, \{\Rightarrow w\Rightarrow\}, \{\Leftarrow w\Leftarrow\}).\)

To a large extent the idea of these global transition relations goes back to Shepherdson [Sh].

**Theorem 2.3 (from [B2]).** Let \( A \) be a (non-deterministic) two-way automaton and let \( u, v \in (\Sigma \cup \{\#\})^* \). Then for the concatenation \( uv \) we have the following formulas:

\[
\begin{align*}
\{\rightarrow uv\rightarrow\} &= \{\rightarrow u\rightarrow\}(\{\Rightarrow v\Rightarrow\}[u\Rightarrow v])* \{\rightarrow v\rightarrow\} \\
\{\Rightarrow uv\Rightarrow\} &= \{\Rightarrow u\Rightarrow\} \cup \{\rightarrow u\rightarrow\}(\{\Rightarrow v\Rightarrow\}[u\Rightarrow v])* \{\Rightarrow v\Rightarrow\} \{\leftarrow u\leftarrow\} \\
\{\Leftarrow uv\Leftarrow\} &= \{\Leftarrow v\Leftarrow\} \cup \{\Leftarrow v\Leftarrow\}(\{\Rightarrow v\Rightarrow\}[u\Rightarrow v])* \{\Rightarrow v\Rightarrow\} \{\rightarrow v\rightarrow\} \\
\{\Leftarrow uv\Leftarrow\} &= \{\Leftarrow v\Leftarrow\}(\{\Rightarrow v\Rightarrow\}[u\Rightarrow v])* \{\leftarrow u\leftarrow\}.
\end{align*}
\]
Notation. Juxtaposition of relations denotes \textit{relational composition}. Relations are composed from left to right, so if $R$ is a relation then the image of an element $x$ under $R$ is denoted by $(x)R$. The \textit{star} $\ast$ denotes the \textit{reflexive–transitive closure} of a relation. The \textit{union} of relations is defined by taking the union of their graphs.

The main consequence of these formulas is that, if $[u]$ and $[v]$ are known then $[uw]$ can be calculated (without knowing the actual inputs $u$ and $v$).

The set $S = \{[w]/w \in \Sigma^\ast\}$ together with the multiplication of elements given by the formulas above, constitutes a monoid (which is finite if the original two-way automaton $A$ is finite), and will be called the \textit{monoid} of $A$. The function $w \in \Sigma^\ast \rightarrow [w] \in S$ is a homomorphism.

If $F$ is a subset of $\hat{Q}$ and if the \textit{acceptance} rule is that $w \in \Sigma^\ast$ is accepted iff there exists a computation of the two-way automaton which starts at the left end in configuration $q_0 \# w \#$ and reaches the right end in a configuration $\# w \# f$ (for some $f \in F$), then we have: $w$ is accepted iff $(q_0)[\rightarrow \# w \# \rightarrow] \cap F \neq \emptyset$.

The following fact is important in connection with positional simulation.

\textbf{Fact 2.4.} If a (non-deterministic) two-way automaton $A$ is started in the configuration $\# uv \#$, where $u$ and $v$ belong to $\Sigma^\ast$, then it will accept (i.e., $A$ can reach a configuration $\# uv \# f$ for some $f$ in $F$, with the reading head at the right end of the tape) iff

$$
(q)([\# u \rightarrow] \cup id)([\# v \rightarrow]([\# u \rightarrow])^\ast [\rightarrow \# \rightarrow] \cap \emptyset \neq \emptyset
$$

(where $id$ is the identity function).

The proof of fact (2.4) is similar to the proof of Theorem 2.3. See [B.2]. We need $id$ to appear in the formula, unless $q \in \hat{Q} \cap \hat{Q}$.

The importance of the above is the following: To decide whether the non-deterministic two-way automaton $A$ accepts the configuration $\# uv \#$, all we need to know is $q$, $[u]$ and $[v]$; $[\# u]$ and $[v \#]$ are then known, since we always know $[\#]$. Thus the monoid $S$ will be useful, and Kannan's conjecture will be proved if from $(u, v)$ we can compute $([u], [v])$ using a \textit{deterministic} two-way finite automaton (Main Lemma 3.2).

3. \textbf{Theorems}

\textbf{Theorem 3.1.} Every \textit{non-deterministic} two-way finite automaton can be \textit{positionally simulated} by some \textit{deterministic} two-way finite automaton.

As mentioned in the Introduction, R. Kannan [K] introduced the notion of positional simulation and conjectured this theorem. The following lemma, which combines two-way automata and monoids, is at the heart of the proof of the theorem and its generalizations and is of interest by itself.
**Main Lemma 3.2.** Let \( \pi : \Sigma^* \to S \) be a semigroup homomorphism, where \( S \) is a finite monoid, \( \Sigma \) is a finite set (alphabet), and \( \Sigma^* \) is the free monoid. Then there exists a deterministic two-way finite automaton \( A \) with input alphabet \( \Sigma \), such that for every pair of words \( u, v \in \Sigma^* \): if \( A \) is started in configuration \( \#uq_0v\# \) (where \( q_0 \) is the start state of \( A \) and \( \# \) is the tape end marker), then the computation of \( A \) ends in a state which "determines \( \pi(u) \) and \( \pi(v) \)." (One way to make the phrase between quotation marks precise is to say that the cartesian product \( S \times S \) is subset of the state set of \( A \) and that the computation ends in state \((\pi(u), \pi(v))\).)

The proof of Lemma 3.2 will be given in Section 4.

**Proof of Kannan's Conjecture from the Main Lemma.** Let \( A_1 \) be a non-deterministic two-way finite automaton with input alphabet \( \Sigma \) and end marker \( \# \), and let \( S = \{ [w] / w \in \Sigma^* \} \) be the monoid of \( A_1 \) (as defined at the end of Section 2). Then we also have a homomorphism \( \pi : u \in \Sigma^* \to \pi(u) = [u] \in S \). Let us now apply the Main Lemma to this situation: there exists a deterministic two-way finite automaton \( A \) which, when given a pair of words \( (u, v) \in \Sigma^* \times \Sigma^* \), computes \( ([u], [v]) \in S \times S \). Knowing \( q_1, [u], [v] \), a deterministic two-way finite automaton \( A_2 \) can then immediately decide (by Fact 2.4) whether it should accept or reject the configuration \( \#uq_1v\# \).

**Theorem 3.3.** Every alternating two-way finite automaton can be positionally simulated by some deterministic two-way finite automaton.

This theorem strengthens the result of Ladner, Lipton, and Stockmeyer \([LLS]\) (just as Theorem 3.1 strengthens the result of Rabin \([R]\) and Shepherdson \([Sh]\)).

One can consider other models of computation known to recognize only regular languages, and see whether they can be positionally simulated by a two-way finite automaton.

**Theorem 3.4.**
1. Every non-deterministic one-tape Turing machine with bounded-length crossing sequences can be positionally simulated by a deterministic two-way finite automaton.

2. Every non-deterministic one-pebble two-way automaton can be simulated by a deterministic two-way finite automaton.

This strengthens the result of F. C. Hennie \([He]\) and the result of M. Blum and C. Hewitt \([BH]\). More generally:

**Theorem 3.5.** Every alternating one-tape Turing machine with bounded-length crossing sequences can be positionally simulated by a deterministic two-way finite automaton.

The proofs of Theorems 3.3, 3.4, and 3.5 use Lemma 3.2, just like the proof of Theorem 3.1. We have to associate a monoid to an alternating two-way automaton (define analogues of \([w \to w] \), \([z \cdot w] \), etc.), in such a way that the corresponding
analyses of Theorem 2.3 and Fact 2.4 apply; see Section 5. For Theorems 3.4 and 3.5 one studies how positional simulation can be obtained for homomorphic images of a two-way automaton; this will be done in Section 6.

Another application of the Main Lemma: Given a 2 DFA $A$ there exists a 2 DFA $B$ which, from a configuration $\#uq_0v\#$ of $A$ computes the crossing sequence of $A$ at that position. (Indeed the knowledge of $[u]$, $[v]$, $q_0$, determines that crossing sequence.)

Remarks on the Size of the Constructions. (1) The deterministic two-way automaton, constructed from $S$ in Lemma (3.2), has a number of states which is less than some polynomial in $|S|$ (where $|S|$ is the cardinality of $S$). This will follow from the proof of of Lemma 3.2.

(2) Let $A$ be a non-deterministic two-way automaton having $n$ states. Then the monoid associated with $A$ (see Definition 2.2, Theorem 2.3, and what follows) has at most $2^{4n^2}$ elements (since each monoid element $[w]$ consists of four relations on the states; if $\mathcal{O} \cap \mathcal{O} = \emptyset$ then the bounds is $2^{n^2}$).

(3) As a consequence of (1) and (2) and the proof of Theorem 3.1 we have: Let $A$ be a non-deterministic two-way with $n$ states. Then there is a deterministic two-way automaton which positionally simulates $A$ and which has at most $c^{n^2}$ states (for some constant number $c$).

The bound $c^{n^2}$ is not much bigger than the best bound known for ordinary equivalence (see $[Sh]$), which is $2^n + n$. However, it is much larger than Kannan's lower bounds (see $[K]$ and Section 1).

Remarks on the reversal complexity. The deterministic two-way automaton $A$ of Lemma 3.2 satisfies: The number of reversals of $A$, when started in any configuration of the form $uuq_0v$, is bounded above by a number depending only on the monoid $S$ (i.e., it does not tend to infinity as the length the inputs increases).

A specific upper bound is $1 + 2 |S|$. This (in fact, a much better upper bound) will be proved at the end of Section 4.

4. Proof of the Main Lemma

It helps to first consider the (very simple) one-way analogue of the Main Lemma:
If $\pi: \Sigma^* \rightarrow S$ is a homomorphism (where $\Sigma$ is a finite set and $S$ is a finite monoid with identity element $I$), then there exists a deterministic one-way finite automaton $A_1 = (Q, \Sigma, \cdot, q_0, ...)$, such that for every input $u \in \Sigma^*$ we have: if it is started in configuration $q_0u$, then at the end of the computation $A_1$ will be in a state which determines $\pi(u)$. In fact $A_1$ can be taken to be $(S, \Sigma, \cdot, I, ...)$, where the next-state function $\cdot$ is given by $(q) a = q \cdot \pi(a)$ (i.e., multiply the elements $q$ and $\pi(a)$ in $S$). Then, if the starting configuration is $Iu$, the state at the end of the computation will be $\pi(u)$.

In the starting situation of the Main Lemma it is easy to find either $\pi(u)$ or $\pi(v)$, using the above one-way analogue. But finding both in one computation is more
difficult (in particular, since a two-way finite automaton cannot remember positions on an arbitrarily long input).

Another helpful preliminary remark: If $S$ is a group then the Main Lemma is easy to prove. A deterministic two-way automaton $A$ which, from an initial configuration $#w_0\#v$ computes $(\pi(u), \pi(v))$ would work as follows: First $A$ goes right and computes $\pi(v)$ (by just using the one-way analogue of the Main Lemma); when $A$ encounters the right endmarker it remembers $\pi(v)$, turns around, and goes left while multiplying on the left by $\pi(a)^{-1}$ for every letter $a$ read—until it finds the left endmarker. Now $A$ has the information $\pi(v)$ and $\pi(uv)^{-1}$; then, in a last state, it stores $\pi(u)$ ($=(\pi(v) \cdot \pi(uv)^{-1})^{-1}$) and $\pi(v)$.

To prove the Main Lemma in general we associate to every element $s$ of a monoid $S$ two functions, and their inverses (which in general are no longer functions):

(1) The right-multiplication function $(\cdot)s: x \in S \to (x)s = x \cdot s \in S$.

(2) The left-multiplication function $s(\cdot): x \in S \to s(x) = s \cdot x \in S$.

(1') The inverse of the right-multiplication function $(\cdot)s^{-1}: y \in S \to (y)s^{-1} = \{x \in S/\ x \cdot s = y\}$.

(2') The inverse of the left-multiplication function $s^{-1}(\cdot): y \in S \to s^{-1}(y) = \{x \in S/\ s \cdot x = y\}$.

The function $(\cdot)s$ and the relation $(\cdot)s^{-1}$ act (are written) on the right side of their argument, $s(\cdot)$ and $s^{-1}(\cdot)$ act on the left. In the following it is important to indicate on what side a function is applied. We will need the following classical and easy properties. (We will sometimes apply a function, say $f$, to a set, say $X$; then $f(X)$ stands for $\{f(x)/x \in X\}$.)

**FACT 4.1.** (1) Inversion law for functions. If $f$ is a function acting on the left then

$$(\forall y): ff^{-1}(y) = y \quad \text{if} \quad f^{-1}(y) \neq \emptyset \quad (\iff y \in \text{Range}(f)),$$

$$= \emptyset \quad \text{otherwise}.$$  

If $f$ is a function acting on the right then

$$(\forall y): (y)f^{-1}f = y \quad \text{if} \quad (y)f^{-1} \neq \emptyset \quad (\iff y \in \text{Range}(f)),$$

$$= \emptyset \quad \text{otherwise}.$$  

(2) Regularity law for functions. If $f$ is a function (acting on the left or the right) then $f^{-1}ff^{-1} = f^{-1}$ and $ff^{-1}f = f$.

(3) Partition mod $f$. If $f: S \to S$ is a function acting on the left then the collection of sets $\{f^{-1}f(x)/x \in S\}$ forms a partition of $S$; also $x \in f^{-1}f(x)$. A similar fact holds for functions acting on the right.

(4) If $f$ and $g$ are relations then $(fg)^{-1} = g^{-1}f^{-1}$ and $(f^{-1})^{-1} = f$. 
Except for point (4), the above properties only hold for functions, not for relations in general.

We now start with the construction of the deterministic two-way automaton $A$ whose existence is claimed in the Main Lemma. Any computation path of $A$ will consist of three phases

1. a start-up phase,
2. a sequence of executions of a main cycle,
3. an application of halting rules.

To simplify the notation, we will always write $(-)w$, $(.)w'$, $w(-)$, $w'(-)$, instead of $(-)\pi(w)$, $(.)(\pi(w))^{-1}$, $\pi(w)(.)$, $(\pi(w))^{-1}(.)$, where $w \in \Sigma^*$ and $\pi(w) \in S$. So, in particular, $(I)w = \pi(w) = w(I)$, where $I$ is the identity element of the monoid $S$.

The state set of $A$ will form a subset $P(S) \times P(S) \times \Sigma \times \Sigma \times P(S) \times P(S) \times fixed$ where $P(S)$ is the power set of $S$, and $fixed$ is a fixed finite set (independent of $S$, $\Sigma$, and $\pi$). The coordinate of the state that belongs to $fixed$ remembers what the current phase (start-up, main cycle, halting) of the computation is and what case and subcase of that phase is currently being applied. See the following proof for details about these cases; the number of possible cases is fixed and finite. In the sequel we shall refer to states as being of the form $(X, Y, ...)$ and we will mention separately what other information is being remembered (in the other six coordinates). This informality makes the proof easier to follow.

As we saw, the states of $A$ form a subset of $P(S) \times P(S) \times ...$. At first the appearance of $P(S)$ makes one expect an exponential upper bound. But the subsets of $S$ that actually appear in the proof are of a restricted form: each of these sets is actually determined by a sequence of at most four elements of $S$. Therefore we will obtain a polynomial bound in $|S|$ (see the end of Section 4).

When $A$ is in state $(X, Y, ...) \in P(S) \times P(S) \times ...$ while moving right on the tape, and if the letter $a \in \Sigma$ is read, then the next state will be $((X)a, a^{-1}(Y), ...)$. This is continued as long as $a^{-1}(Y) \neq \emptyset$. We shall see shortly what $A$ does when $a^{-1}(Y) = \emptyset$.

When $A$ is in state $(X, Y, ...)$ while moving left, and if the letter $a$ is read, then the next state will be $((X)a^{-1}, a(Y), ...)$ unless a certain condition holds that will be explained soon.

Start-up Phase of $A$

The deterministic two-way finite automaton $A$ is started in configuration $#u \uparrow v#$, in state $(I, I, ...)$, where $I$ is the identity element of the monoid $S$. The goal of $A$ is to find $\pi(u)$ and $\pi(v)$.

First, the reading head moves right, and for every letter $a \in \Sigma$ of $v$ read, $A$ undergoes a state transition of the form $(X, Y, ...) \rightarrow ((X)a, a^{-1}(Y), ...)$. This is continued as long as $a^{-1}(Y) \neq \emptyset$. Two cases are possible:
Case 1. \( a^{-1}(Y) \) never becomes \( \emptyset \). Eventually the reading head encounters the right-end marker of the tape; the head then places itself just left of the endmarker. The configuration of \( A \) at that point is \( \# w \uparrow \# \), in state \((1)v, v^{-1}(1), \ldots \) with \( v^{-1}(1) \neq \emptyset \). In this case \( \pi(v) = (1)v \) is directly known to \( A \). To determine \( \pi(u) \), the automaton \( A \) moves left all the way, undergoing state transitions of the form \((..., Y, \ldots) \rightarrow (..., a(Y), \ldots)\) for every letter \( a \) of \( uv \) read (here the first coordinates do not matter). Eventually the reading head encounters the left-end marker; it then places itself just right of the endmarker. The configuration now is \( \# w \uparrow \#, \) with state \((..., uwv^{-1}(1), \ldots)\); but \( uwv^{-1}(1) = u(1) = \pi(u) \). Indeed the inversion law for functions, Fact 4.1, applies to \( v^{-1}(1) \), since \( v^{-1}(1) \neq \emptyset \). So now \( A \) knows \( \pi(u) \) and \( \pi(v) \).

Case 2. \( a^{-1}(Y) \) becomes \( \emptyset \) after a strict prefix of \( v \) has been read. More precisely, let \( v_1 \) be the largest prefix of \( v \) such that \( v_1^{-1}(1) \neq \emptyset \). Of course \( v_1 \) could be the empty string; then \( \pi(v_1) = 1 \). Let \( a_1 \in \Sigma \) be the next letter of \( v \); thus \((v_1, a_1)^{-1}(1) = a_1^{-1}v_1^{-1}(1) = \emptyset \). When \( A \) sees that \( a_1^{-1}v_1^{-1}(1) = \emptyset \), it places its reading head just left of \( a_1 \). Now the configuration of \( A \) is \( \# w_1 \uparrow a_1w_1 \) \# (where \( v = v_1a_1w_1 \), with state \((1)v_1, v_1^{-1}(1), \ldots \)). We also want \( A \) to remember the letter \( a_1 \) in its finite control. The position on the tape where the head currently is will be called "the current right boundary."

Now \( A \) remembers \((1)v_1, v_1^{-1}(1), a_1, \ldots \), goes left, and undergoes state transitions of the form \((X, Y, \ldots) \rightarrow ((X)a^{-1}, a(Y), \ldots)\) for every letter \( a \in \Sigma \) read. \( A \) also remembers \( \pi(z) \), where \( z \) is the segment read so far (going left from the right boundary). After a segment \( z \) has been read the state will be \((..., zwz^{-1}(1), \ldots)\), with \( a_1, \pi(z) \), and \((1)v_1, v_1^{-1}(1)) \) also remembered. If \( A \) would now move right again and undergo the usual state transitions, \( A \) would come back to the right boundary in a state \((..., z^{-1}zwz^{-1}(1), \ldots)\). By law (3) of Fact 4.1, \( v_1^{-1}(1) \) is a subset of \( z^{-1}zwz^{-1}(1) \). Thus although \( a_1^{-1}z^{-1}zwz^{-1}(1) = \emptyset \), it might happen that \( a_1^{-1}z^{-1}zwz^{-1}(1) \neq \emptyset \) (since \( z^{-1}zwz^{-1}(1) \) may be a bigger set); when this happens, we say that the "right boundary has moved." It is important that the automaton actually does not have to go back to the right boundary in order to check whether the right boundary has moved; instead, it remains in position \( \# \ldots \uparrow zwz, a_1, w_1 \) \#, and there it evaluates \( a_1^{-1}z^{-1}zwz^{-1}(1) \) (which it can do, since \( v_1^{-1}(1), \pi(z) \), and \( a_1 \) were remembered) and checks if the result is \( \emptyset \). As long as it is \( \emptyset \), \( A \) keeps going left.

Two cases are possible now:

Case 2A. The right boundary never moves \((a_1^{-1}z^{-1}zwz^{-1}(1) = \emptyset \) for every suffix \( z \) of \( uv_1 \)). Eventually \( A \) reaches the left end marker of the tape. At that point the head places itself just right of the end marker; the configuration is \# \( \uparrow uv_1, a_1w_1 \) \# with state \((..., uv_1, v_1^{-1}(1), \ldots)\); \( A \) also remembers \( \pi(uv_1) \) (here \( z = uv_1 \)) and \( a_1 \) and the information \((1)v_1, v_1^{-1}(1)) \) about the state at the right boundary. By Fact 4.1, \( v_1v_1^{-1}(1) = 1 \), since \( v_1^{-1}(1) \neq \emptyset \). Thus from \( uv_1v_1^{-1}(1) = u(1) \) the automaton knows \( \pi(u) = u(1) \).

To determine \( \pi(v) \), the automaton goes right, and searches for the position of the
right boundary; $A$ is indeed able to find the position of the right boundary back, as follows:

Let $x$ be the segment read so far while going right from the left-end marker. Now the state is $(...,$ $x^{-1}u(1), ...)$, and $A$ also still remembers $\pi(u)$, $a_1$, $(1)v_1$. We have the following characterization of the right boundary:

1. If $x$ is a prefix of $uw_1$ then $x^{-1}u(1) \neq \emptyset$.
2. If $x = uw_1a_1$ then $x^{-1}u(1) = \emptyset$.

(For a proof, which is easy anyway, look ahead at the proof of Fact 4.5.)

Using this characterization, $A$ can find the right boundary back while moving right. When the position $#uw_1a_1w_1#$ is reached again, $A$ uses $(1)v_1 = \pi(v_1)$ (which was remembered), and continues going right while multiplying $\pi(v_1)$ with $\pi(a_1)$, then $\pi(a_2)$, ... for every letter $a_1, a_2, ... \in \Sigma$ read. When the right end marker is reached, this will yield $\pi(v)$. So now $A$ knows $\pi(u)$ and $\pi(v)$.

Case 2B. The right boundary moves (after a strict suffix $z$ of $uw_1$ has been read). Now we shall find a new right boundary further right, as in the Main Cycle (replacing $u_1$ by the empty word in the notation of the Main Cycle).

Main Cycle

The Main Cycle begins with the head of $A$ placed at the current "right boundary." The head will then move left (applying state transitions), trying to "make the right boundary move." There exists a condition for deciding whether the right boundary has moved; $A$ can test this condition on the fly while it is going left, using information gathered in its finite control.

If, while $A$ goes left, the right boundary never moves (i.e., the left end marker of the tape is encountered), then we apply the Halting Rule (2).

If the right boundary moves, $A$ turns back right and finds the old right boundary (it is a fact that $A$ is able to find the position of the old right boundary back, using information gathered in its finite control). Then it searches for a new right boundary further to the right:

Either no new right boundary is found (i.e., the right end marker of the tape is encountered): apply the Halting Rule (1).

Or a new right boundary is found, strictly to the right of the old one: now start the Main Cycle again.

Now a more detailed description of the Main Cycle follows.

We start the cycle in configuration $#uw_1a_1w_1#$ with state $(1)v_1, v_1^{-1}u_1^{-1}u_1(1), ...)$, where $A$ remembers also $a_1 \in \Sigma$. We assume that $v_1^{-1}u_1^{-1}u_1(1) \neq \emptyset$ but $a_1^{-1}v_1^{-1}u_1^{-1}u_1(1) = \emptyset$ (that is, the head of $A$ is positioned at the current right boundary). Here $u_1$ is a suffix of $u$, and $v = v_1a_1w_1$. 
Making the Right Boundary Move

The reading head of \( A \) goes left and tries to "make the right boundary move": While going left it makes state transitions of the form \((X, Y, \ldots) \rightarrow ((X)a^{-1}, a(Y), \ldots)\), for each letter \( a \in \Sigma \) read. \( A \) continues until, after reading say a suffix \( z \) of \( uv \), it is in a state \(((1)v_1z^{-1}, zv_1^{-1}u_1^{-1}u_1(1), \ldots)\), satisfying

\[
a_1^{-1}z^{-1}z(v_1^{-1}u_1^{-1}u_1(1)) \neq \emptyset. \quad (4.2)
\]

For \( A \) to be able to check this condition, it must have remembered \( \pi(z) \) in its finite control. Actually, \( A \) will remember the left-most letter of \( z \) (call it \( b \)) and \( \pi \) of the rest of \( z \); all this determines \( \pi(z) \) and is still a finite amount of information, belonging to \( \Sigma \times S \). As soon as \( A \) finds that condition (4.2) holds for \( z \), \( A \) places itself one cell to the right. The configuration now is \( \# \ldots b \uparrow v_1a_1w_1 \# \), where \( v = v_1a_1w_1 \) and \( z = by \).

Just like in Case 2 of the Start-up phase, condition (4.2) decides whether the right boundary has moved. Indeed, if \( A \) would now go back to its right boundary, the state it would be in when it gets back to this right boundary would be \(((z-1z^{-1}y^{-1}u_1^{-1}u_1(1), \ldots); applying \( a_1^{-1} \) will then produce \( a_1^{-1}z^{-1}zv_1^{-1}u_1^{-1}u_1(1) \); this set is \( \emptyset \) if we are at the right boundary, and is \( \neq \emptyset \) if now the boundary has shifted (to the right). Note that condition (4.2) is checked by \( A \) in its current position (using the information remembered); \( A \) does not have to travel back to the right boundary to check if the right boundary has moved.

**FACT 4.3.** Suppose \( A \) is going left, starting at the right boundary, and trying to make the right boundary move. The position reached by \( A \) when the right boundary moves, is strictly to the left of any position of the tape that \( A \) has ever visited so far in the computation. So, in the above notation, the segment \( u_1v_1 \in \Sigma^* \) is a strict suffix of \( z \) (that is, \( z = bu_2u_1v_1 \) for some \( u_2 \in \Sigma^* \)).

**Proof of Fact 4.3.** We must show that as long as \( z \) is a suffix of \( u_1v_1 \), the condition (4.2) evaluates to \( a_1^{-1}z^{-1}zv_1^{-1}u_1^{-1}u_1(1) = \emptyset \). Let \( u_1v_1 = yz \) for some \( y \in \Sigma^* \). So \( v_1^{-1}u_1^{-1}y^{-1}(\cdot) = (u_1v_1)^{-1}(\cdot) = z^{-1}y^{-1}(\cdot) \). Then, indeed, \( a_1^{-1}z^{-1}zv_1^{-1}u_1^{-1}u_1(1) = a_1^{-1}z^{-1}z^{-1}y^{-1}u_1^{-1}u_1(1) \). By Fact (4.1) (regularity law: \( z^{-1}z^{-1}(\cdot) = z^{-1}(\cdot) \)) this is equal to \( a_1^{-1}z^{-1}y^{-1}u_1^{-1}u_1(1) = a_1^{-1}v_1^{-1}u_1^{-1}u_1(1) \). But this is indeed \( = \emptyset \) (by the form of the state at the old right boundary, given at the beginning of the Main Cycle). \( \square \)

Finding the New Right Boundary

Assuming we made the right boundary move, we next have to go right and find the position of the new right boundary—situated to the right of the old right boundary. (See the subsection "Halting Rules" (2) concerning the case where \( A \) encounters the left-end marker of the tape, without ever making the right boundary move.)

Suppose the current configuration is \( \# \ldots b \uparrow u_2u_1v_1a_1w_1 \# \), where the segment
previously called "z" (= by) is denoted $bu_2u_1v_1$ (see Fact 4.3). Here $u = t_2bu_2u_1$ and $v = v_1a_1w_1$. The current state is $(...)u_2u_1v_1v_1^{-1}u_1^{-1}(1), ...$. By fact 4.1 (Inversion law), $u_1v_1(u_1v_1)^{-1}(u_1(1)) = u_1(1)$, since $(u_1v_1)^{-1}(u_1(1)) = v_1^{-1}u_1^{-1}u_1(1) \neq \emptyset$. Therefore the current state in the above configuration is $(...)u_2u_1(1), ...$. So we proved:

**Fact (4.4).** At the position $\# ... b \uparrow u_2u_1v_1a_1 ... \#$ at which $A$ is when the right boundary is just about to move $A$ is in state $(...)u_2u_1(1), ...$.

Suppose $A$ went to the position $\# ... b \uparrow u_2u_1v_1a_1 ... \#$ at which it notices that the right boundary moves; now it goes to position $\# ... b \uparrow u_2u_1v_1a_1 ... \#$. In addition to $u_2u_1(1)$ (which is part of the current state) the automaton $A$ also remembers $\pi(u_2u_1v_1) \in S$ and $b \in \Sigma$ (in the above notation $z = bu_2u_1v_1$); $A$ also remembered $((v_1, v_1^{-1}u_1^{-1}u_1(1))$ and $a_1$ (which are components of the state that $A$ was in at the right boundary). Using this information $A$ must find the position of the new right boundary.

First $A$ must again find the position of the old right boundary. This is done as in Case 2A of the start-up phase:

**Fact 4.5.** Consider the position $\# ... b \uparrow u_2u_1v_1a_1 ... \#$ at which $A$ places itself immediately after the right boundary moved. Then, going right, $A$ is able to again find the position of the old right boundary, as follows: After reading a segment $x$, the state will be $(...)x^{-1}u_2u_1(1), ...$. The right boundary is characterized by the two properties:

1. As long as $x$ is a prefix of $u_2u_1v_1$ we have $x^{-1}u_2u_1(1) \neq \emptyset$.
2. When $x = u_2u_1v_1a_1$ we have $x^{-1}u_2u_1(1) = \emptyset$.

From the information that $A$ remembered, $A$ is indeed able to check properties (1) and (2).

**Proof.** Properties (1) and (2) characterize the old right boundary:

**Property (1).** If $x$ is a prefix of $u_2u_1v_1$ then there exists $y \in \Sigma^*$ such that $xy = u_2u_1v_1$. Recall also that $u_2u_1(1) = u_2u_1v_1v_1^{-1}u_1^{-1}u_1(1)$ (proved just before Fact 4.4). Thus $x^{-1}u_2u_1(1) = x^{-1}u_2u_1v_1v_1^{-1}u_1^{-1}u_1(1) = x^{-1}u_2u_1v_1v_1^{-1}u_1^{-1}u_1(1); \emptyset$ does this set contains the set $yv_1^{-1}u_1^{-1}u_1(1)$ (by Fact 4.1 (3)), which is non-empty since $v_1^{-1}u_1^{-1}u_1(1) \neq \emptyset$. Therefore $x^{-1}u_2u_1(1) \neq \emptyset$.

**Property (2).** If $x = u_2u_1v_1a_1$, we have:

$x^{-1}u_2u_1(1) = (u_2u_1v_1a_1)^{-1}u_2u_1v_1v_1^{-1}u_1^{-1}u_1(1)$. But this set is $\emptyset$, by Fact 4.3. (The intuitive reason for Property (2) is that at position $\# ... b \uparrow u_2u_1v_1a_1 ... \#$, the right boundary has not yet moved.)

When $A$ reaches the position of the old right boundary it goes into state $((v_1, v_1^{-1}bu_2u_1)^{-1}bu_2u_1(1), ...)$, which it can determine from the information it had remembered. Now it reads $a_1$ and goes to state $((v_1, v_1^{-1}bu_2u_1)^{-1}bu_2u_1(1), ...)$.

It is important to note that $a_1^{-1}v_1^{-1}(bu_2u_1) = a_1^{-1}z_0^{-1}zv_1^{-1}u_1^{-1}u_1(1) \neq \emptyset$ (by the choice of $z$, see condition (4.2))—whereas for the
old state at the right boundary we had $a_1^{-1}(u_1^{-1}u_1v_1(1)) = \emptyset$. Thus, the new right boundary is at least one step to the right of the previous right boundary.

Now, $A$ keeps going right, doing state transitions of the form $(X, Y, \ldots) \rightarrow ((X)a, a^{-1}(Y), \ldots)$ for each letter $a \in \Sigma$ read on the tape, as long as $a^{-1}(Y) \neq \emptyset$. When $a^{-1}(Y)$ is $\emptyset$ we have found a new right boundary. Now we start the Main Cycle again with this new right boundary.

It could happen that the right end marker of the tape is encountered while $A$ goes right looking for a new right boundary; see the Halting Rule (1) for that case.

**Halting Rules**

1. While the reading head of $A$ goes right in search of a (new) right boundary, it encounters the right end marker: Then $A$ will position itself left of the end marker and at this point the configuration is $\#uv\uparrow\#$ and the state is $((1)v_1, v_1^{-1}u_1^{-1}u_1(1), \ldots)$, where $u_1$ is a suffix of a $u$ and where $v_1^{-1}u_1^{-1}u_1(1) \neq \emptyset$. From the state, $(1)v = \pi(v)$ is immediately determined. The determination of $\pi(u)$ is similar to Case 1 of the Start-up phase.

2. While the reading head of $A$ goes left while trying to make the right boundary move it encounters the left-end marker: Then it positions itself right of the endmarker, and at that point the configuration is $\#tv\downarrow\#$, and the state is $(\ldots, u(1), \ldots)$ (by adapting Fact 4.3); $A$ also remembers the information $((1)v_1, v_1^{-1}u_1^{-1}u_1(1))$ about the state at the right-boundary, where $v_1$ is a prefix of $v$ and $u_1$ is a suffix of $u$.

From $u(1) (= \pi(u))$, $A$ knows $\pi(u)$. To find $\pi(v)$, the automaton $A$ proceeds as in Case 2A of the Start-up phase.

This completes the proof of the Main Lemma. The techniques of this proof are similar to a construction of [B1]. See also [P].

Let us prove our claim that the number of states of the deterministic two-way finite automaton $A$ is less than some polynomial in $|S|$ and $|\Sigma|$.

As we say at the beginning of Section 4, the states of $A$ form a subset of $\mathcal{P}(S) \times \mathcal{P}(S) \times \Sigma \times \Sigma \times \mathcal{P}(S) \times \mathcal{P}(S) \times \text{FIXED}$. At first the appearance of $\mathcal{P}(S)$ makes one expect an exponential upper bound. But the subsets of $S$ that actually appear in the proof are of the form $\{x\}$, $(x)s^{-1}$, $(x)s^{-1}t$, $s^{-1}(x)$, $ts^{-1}(x)$, $r^{-1}ts^{-1}(x)$ (where $x, s, t, r \in S$); thus each of these sets is actually determined by a sequence of at most four elements of $S$. Therefore we obtain a polynomial bound in $|S|$.

Let us finally prove that the number of reversals of $A$ is bounded by a number depending only on the monoid $S$. First we need a few definitions from semigroup theory. If $S$ is a monoid and $s, t \in S$, we define:

- $s \leq_R t$ iff there exists $x \in S$ with $s = tx$.
- $s \leq_\ast t$ iff there exists $y \in S$ with $s = yt$.
- $s \leq_\ast \ast t$ iff there exist $a, b \in S$ with $s = abt$. 


Also, if $\leq$ denotes any one of the above three relations, we define:

\[ s = t \quad \text{iff} \quad s \leq t \quad \text{and} \quad t \leq s. \]

\[ s < t \quad \text{iff} \quad s \leq t \quad \text{but not} \quad s = t. \]

We define $R\text{-DEPTH}(S)$ to be the length of the longest strict $<_R$-chain in $S$, where we count the number of $<_R$’s. Similarly, we define $L\text{-DEPTH}(S)$.

At the right boundary $\# \ldots u_1v_1 \uparrow a_1 \ldots \#$ we have (see the beginning of the Main Cycle): $(u_1v_1)^{-1}u_1(1) \neq \emptyset$ and $(u_1v_1a_1)^{-1}u_1(1) = \emptyset$. This implies $\pi(u_1v_1) = R \pi(u_1)$ and $\pi(u_1v_1a_1) <_R \pi(u_1)$; therefore $\pi(u_1v_1a_1) <_R \pi(u_1v_1)$. It follows that $\pi(v_1a_1) <_R \pi(v_1)$.

This implies that the number of right boundaries that appear during the computation is $\leq R\text{-DEPTH}(S)$. Since every right-to-left reversal occurs at a right boundary (or at the right end-marker, at most once) we have: the number of right-to-left reversals is $\leq 1 + R\text{-DEPTH}(S)$.

When the right boundary moves (while $A$ is in a configuration $\# \ldots b \uparrow u_2u_1v_1a_1 \ldots \#$) we have (letting $z = bu_2u_1v_1$): $a_1^{-1}(bu_2u_1v_1)^{-1}bu_2u_1v_1v_1^{-1}u_1^{-1}u_1(1) = (bu_2u_1v_1a_1)^{-1}bu_2u_1(1) = \emptyset$ (by (4.2) and Fact 4.3), and $(u_2u_1v_1a_1)^{-1}u_2u_1(1) = \emptyset$ (by Fact 4.5). Therefore $\pi(bu_2u_1v_1a_1) = R \pi(bu_2u_1)$ and $\pi(u_2u_1v_1a_1) <_R \pi(u_2u_1)$; hence $\pi(bu_2u_1) = R \pi(bu_2u_1v_1a_1) <_1 \pi(u_2u_1v_1a_1) <_R \pi(u_2u_1)$. Since $S$ is finite this implies $\pi(bu_2u_1) <_1 \pi(u_2u_1)$, and therefore $\pi(bu_2u_1) <_1 \pi(u_2u_1)$.

This will imply that the number of left-to-right reversals is $\leq 1 + L\text{-DEPTH}(S)$. Finally, we obtain: The total number of reversals in any computation of $A$ is $\leq 1 + 2 \cdot \min\{R\text{-DEPTH}(S), L\text{-DEPTH}(S)\}$.

Remark. If one wants to further decrease the number of reversals of $A$ (constructed from a homomorphism $\pi: \Sigma^* \to S$) one can replace $S$ by any other monoid $T$ containing $S$ as a sub-semigroup, in the construction of $A$. In doing so one may decrease the $R$- and $L$-depths. Note that we did not require that the homomorphism be surjective.

5. ALTERNATING TWO-WAY FINITE AUTOMATA

In this section we will associate a finite monoid to a two-way alternating finite automaton, in such a way that analogues of Theorem 2.3 and Fact 2.4 hold. Then Theorem 3.3 (Kannan’s conjecture, generalized to alternating two-way automata) can be proved in exactly the same way as Theorem 3.1. See [Ko, ChS, ChKS] for background on alternation.

It helps to first look at the case of one-way alternating automata. Assume the state set $Q$ consists of $\forall$-states, $\exists$-states, and negation states. An alternating one-way finite automaton with state set $Q$ is equivalent to a deterministic one-way finite automaton, whose states are boolean functions with $|Q|$ variables (i.e., elements of
\{0, 1\}^{(0,1)Q}$. See, e.g., [BL, pp. 25–26] for details. We shall denote the set \{0, 1\}^{(0,1)Q} by \(B_Q\). It is easiest to think of these boolean functions \(\varepsilon B_Q\) as represented by boolean expressions over \(\lor, \land, \neg\), with the elements of \(Q\) as variables. The monoid of the alternating one-way automaton is the transformation monoid of the equivalent deterministic one-way automaton; thus it consists of operators \(B_Q \rightarrow B_Q\). Every input letter \(a \in \Sigma\) acts as such an operator whose action on a boolean expression \(\beta\) is as follows: for every state \(q\) occurring in \(\beta\) look at the edges labeled by \(a\) and exiting from \(q\) in the state graph of the automaton; let \(p_1, \ldots, p_n\) be the states pointed to. If \(q\) is a \(\forall\)-state, replace \(q\) by \(p_1 \land \cdots \land p_n\) in \(\beta\); if \(q\) is an \(\exists\)-state, replace \(q\) by \(p_1 \lor \cdots \lor p_n\) in \(\beta\); if \(q\) is a negation state then \(n=1\) and \(q\) is replaced by \(-p_1\) in \(\beta\). Do this simultaneously for all states occurring in \(\beta\). The monoid is generated by the operators associated to the letters, under composition.

Let us now consider a two-way alternating finite automaton \(A\) ([LLS] showed that the language accepted by such an automaton is regular). We assume from now on that \(\tilde{Q} \cap \tilde{Q} = \emptyset\); an automaton can be easily (positionally) simulated by one for which this condition holds (just make two copies of the intersection).

For a given input \(w \in (\Sigma \cup \{\#\})^*\) (where \(\#\) is the tape end marker), a configuration of \(A\) can be considered to be an element of \((\tilde{Q} \cup \tilde{Q}) \times \{0, 1, \ldots, |w|\}\). We consider the computation tree (see [ChS, ChKS]) of \(A\) on input \(w\) for some initial configuration \((q, i)\), and we distinguish between leaves and interior vertices. We will use the complete computation tree, with vertices corresponding to all the configurations reachable from \((q, i)\). A leaf is a vertex corresponding to a configuration of the form \((\tilde{q}_1, |w|)\) with \(\tilde{q}_1 \in \tilde{Q}\), or of the form \((\tilde{q}_2, 0)\) with \(\tilde{q}_2 \in \tilde{Q}\), or a configuration \((q, i)\) with \(0 < i < |w|\) for which no next state is defined. In these three cases we will label the leaf respectively by \(\tilde{q}_1\) or \(\tilde{q}_2\), or by TRUE/FALSE (depending on whether the state is an accept state or not).

An interior vertex will be labeled by \(\land\) or \(\lor\) or \(-\), if the state of the corresponding configuration is respectively \(\forall\) or \(\exists\) or a negation state. For each \(w \in \Sigma\) we shall now define two operators \(B_Q \rightarrow B_Q\), denoted \([-w]\) and \([w\leftarrow]\).

\textbf{Notation.} \(Q = \tilde{Q} \cup \tilde{Q}\). The pair \([-w], [w\leftarrow]\) will be denoted \([w]\), and \([w]/w \in \Sigma^*\) will be the finite monoid associated to \(A\) (once suitable formulas for the multiplication are established, similar to Theorem 2.3).

\textbf{Definition of \([-w]\) and \([w\leftarrow]\)}

Again, it is easier to view \([-w]\) as acting on boolean expressions rather than on boolean functions \(\varepsilon B_Q\). Then we only have to define \((q)[[-w]]\) for each \(q \in \tilde{Q} \cup \tilde{Q}\). First, define \((q)[[-w]] = \text{FALSE}\) for \(q \in \tilde{Q}\). (Recall also that we assume \(\tilde{Q} \cap \tilde{Q} = \emptyset\).)

To define \((q)[[-w]] \in B_Q\) for \(q \in \tilde{Q}\), consider the computation tree of \(A\) for the starting configuration \(qw\). This tree can be viewed as the parse tree of a boolean expression (see [ChKS]); we shall define \((q)[[-w]]\) to be this boolean expression (or, more rigorously, the boolean function \(\varepsilon B_Q\) represented by this boolean expression).
One technical difficulty arises if the computation tree is infinite. In that case, we take the computation tree with vertices labeled by configurations and truncate it at progressively greater depth $k$ (for any $k \geq 1$). From each such finite tree we obtain a boolean expression $(q)[\rightarrow w]_k$, over the set of boolean variables $Q$, as follows: we label the interior vertices by $\wedge$, $\lor$, or $\neg$, as above; any leaf of the form $(q, i)$ that would be a leaf of the infinite tree (so either $q \in Q$ and $i = 0$, or $q \in \bar{Q}$ and $i = |w|$, or the machine halts) is labeled by $q$, or TRUE, or FALSE, as before; a leaf $(q, i)$ which is not a leaf of the infinite tree is just labeled by $(q, i)$. This new finite tree is the parse tree of a boolean formula $\beta_k$ over the set of variables $Q \cup Q \times \{0, 1, \ldots, |w|\}$. Finally, the boolean formula $(q)[\rightarrow w]_k$ is defined to be $(\forall(q_1, 1)) \cdots (\forall(q_{|Q|}, |w| - 1)) : \beta_k$. (I.e., one takes $\beta_k$ and $\forall$-quantifies it with respect to all variables in $Q \times \{0, 1, \ldots, |w|\}$.)

Then one defines $(q)[\rightarrow w]$ to be $\lim_{k \to \infty} (q)[\rightarrow w]_k$. The limit $x = \lim_{k \to \infty} x_k$ of a sequence $x_k$ of boolean functions of $|Q|$ variables is a boolean function defined as follows: to define $x(t_1, \ldots, t_{|Q|})$ for $(t_1, \ldots, t_{|Q|}) \in \{\text{TRUE}, \text{FALSE}\}^{|Q|}$, we take the sequence $x_k(t_1, \ldots, t_{|Q|})$ as $k \to \infty$; if it eventually stabilizes to TRUE then we take this as the value, otherwise we take FALSE. In a similar way one defines $(q)[w\leftarrow] \in B_Q$ for all $q \in Q$.

Once $(q)[\rightarrow w]$ and $(q)[w\leftarrow]$ have been defined for $q \in Q \cup \bar{Q}$, one defines $(\beta)[\rightarrow w]$ and $(\beta)[w\leftarrow]$ (for a boolean expression $\beta$) by replacing each occurrence of a state $q$ in $\beta$ by $(q)[\rightarrow w]$ (resp. $(q)[w\leftarrow]$) and then taking the resulting boolean expression. If $A$ has an end-of-tape marker $\#$ we define $[\rightarrow w]$ and $[w\leftarrow]$ on all of $(\Sigma \cup \{\#\})^*$. From $[\rightarrow w]$ and $[w\leftarrow]$ we will define the operators $[\rightarrow w \rightarrow]$, $[\leftarrow w \leftarrow]$, $[w \leftarrow]$, $[\leftarrow w \rightarrow]$; this will be needed to express the multiplication formulas of Theorem 5.1.

We first need to introduce a second copy of $Q = \bar{Q} \cup \bar{\bar{Q}}$. Let $\bar{Q}$, $\bar{\bar{Q}}$ be new sets that are in one-to-one correspondence with $\bar{Q}$, respectively $\bar{\bar{Q}}$. Denote $Q = \bar{Q} \cup \bar{\bar{Q}}$. The reason for introducing $Q$ is as follows: In the non-deterministic case, $(q)[\rightarrow w \rightarrow]$ represented the set of states the automaton could be in when the reading head leaves $w$ on the right. In the alternating case, $(q)[\rightarrow w]$ is a boolean expression containing both states in which the automaton leaves $w$ on the right, and on the left. In this case it would not be useful to simply delete the left-moving states; instead, we mark the states we are really interested in, namely the right-moving ones, in bold. This will become clearer in the proof of Facts 5.1 and 5.2, and in a remark at the end of that proof.

**Definition.** $[\rightarrow w \rightarrow]$ is a function $B_{Q \cup \bar{Q}} \rightarrow B_{Q \cup \bar{Q}}$. We view $(q)[\rightarrow w \rightarrow]$ as a boolean expression, defined by: For $q \in Q$, $(q)[\rightarrow w \rightarrow] = q$; for $\bar{q} \in \bar{Q}$, $(\bar{q})[\rightarrow w \rightarrow] = \text{FALSE}$; and for $q \in Q$, $(q)[\rightarrow w \rightarrow] = (\bar{q})[\rightarrow w]$, where in this boolean expression, states in $Q$ are replaced by the corresponding states in $\bar{Q}$ (and where $\bar{q}$ is the element of $\bar{Q}$ that corresponds to $q$). In other words, to obtain the boolean expression $(q)[\rightarrow w \rightarrow]$ one first replaces $\bar{q}$ by $\bar{q}$, and then one applies $[\rightarrow w]$ to $\bar{q}$; this yields a boolean expression with variables in $Q \cup \bar{Q}$. Finally, in this expression one replaces the variables in $\bar{Q}$ by their copy in $\bar{Q}$.
DEFINITION. \([\varphi w]\) is a function \(B_{Q_0 \cup \bar{Q}} \to B_{Q_0 \cup \bar{Q}}\), where, as a boolean expression \((q)[z w]\) is defined by: For \(q \in Q\), \((q)[\varphi w] = q\); for \(\bar{q} \in \bar{Q}\), \((\bar{q})[\varphi w] = \text{false}\); and for \(q \in Q\), \((q)[\varphi w] = "(q)[\rightarrow w]", where in this boolean expression, states in \(\bar{Q}\) are replaced by the corresponding states in \(\bar{Q}'\) (and where \(q \in Q\), corresponds to \(\bar{q} \in \bar{Q}\)).

Similarly, one defines \([\leftarrow w\right]\) and \([w \varphi]\) from \([w \rightarrow]\).

Now we can state the analogues of Theorem 2.3 and Fact 2.4, for alternating two-way finite automata.

FACT 5.1. Let \(A\) be an alternating two-way finite automaton with input alphabet \(\Sigma\) and endmarker \(\#\), and let \(u, v \in (\Sigma \cup \{\#\}^*)\). Let \(q \in Q\) be any state of \(A\) and let \(\bar{q}\) be the element of \(\bar{Q}\) that corresponds to \(q\). Then:

\[
(q)[\rightarrow uv] = \lim_{k \to \infty} s_k,
\]

where for all \(n > 0\), \(s_{2n} = (\forall q_1) \ldots (\forall q_{|Q|}) : (q)[\rightarrow u \rightarrow]([\varphi v][u \varphi])^n\) and \(s_{2n+1} = (\forall q_1) \ldots (\forall q_{|Q|}) : (q)[\rightarrow u \rightarrow]([\varphi v][u \varphi])^n[\varphi v]\), and where \(Q = \{q_1, \ldots, q_{|Q|}\}:

\[
(q)[\leftarrow uv] = \lim_{k \to \infty} t_k,
\]

where for all \(n > 0\), \(t_{2n} = (\forall q_1) \ldots (\forall q_{|Q|}) : (q)[\leftarrow v \leftarrow]([u \varphi][v])^n\) and \(t_{2n+1} = (\forall q_1) \ldots (\forall q_{|Q|}) : (q)[\leftarrow v \leftarrow]([u \varphi][v])^n[u \varphi]\).

In words, \(s_{2n}\) is obtained by taking the boolean expression \((q)[\rightarrow u \rightarrow] ([\varphi v][u \varphi])^n\) over the set of variables \(Q = Q \cup \bar{Q}\), and then \(\forall\)-quantifying all variables in \(Q\). One proceeds similarly for \(s_{n+1}, t_{2n}, \) and \(t_{2n+1}\). The limit of a sequence of boolean functions was defined previously (see the definition of \((q)[\rightarrow w]\))

FACT 5.2. Let \(A\) be an alternating two-way finite automaton with input alphabet \(\Sigma\) and endmarker \(\#\). Let \(u, v \in (\Sigma \cup \{\#\}^*)\), \(q \in Q\), and let \(F\) (subset of \(Q\)) be the set of accept states. Let \(\bar{q}\) be the element of \(\bar{Q}\) that corresponds to \(q\). Then \(A\) accepts the configuration \(uv\) iff \(\lim_{k \to \infty} r_k\) evaluates to \(\text{true}\) when the elements of \((Q - F) \cup Q\) are set to \(\text{false}\), and the elements of \(F\) are set to \(\text{true}\); here \(r_{2n} = (\forall q_1) \ldots (\forall q_{|Q|}) : (q)([\varphi v][u \varphi])^n\) and \(r_{2n+1} = (\forall q_1) \ldots (\forall q_{|Q|}) : (q)([\varphi v][u \varphi])^n[\varphi v]\), for all \(n\).

If \(q \in \bar{Q}\), switch \([\varphi v]\) and \([u \varphi]\) in the above formulas. In the special case where \(u\) is the empty word the result simplifies: \(A\) accepts the configuration \(qv\) iff \((q)[\rightarrow v]\) evaluates to \(\text{true}\) when the elements of \(Q - F\) are set to \(\text{false}\) and the elements of \(F\) are set to \(\text{true}\).

Notation. Recall that we compose functions from left to right.

By definition, a configuration \((q, i)\) on input \(w\) is accepting iff \(q \in F\) (in \(\bar{Q}\)) and \(i = |w|\).

The important consequence of the two facts is that \([u]\) and \([v]\) together determine \([uv]\), and that \(q, [u], [v]\) together determine whether the configuration \(uv\) is accepted.
Proof (Outline) for Facts (5.1) and (5.2). We only consider the formula for $[\rightarrow uv]$ in Fact 5.1. The other formula and Fact 5.2 are proved in a similar way.

We take the computation tree of $A$ with initial configuration $quv$ and subdivide the tree into successive $u$-regions and $v$-regions. A $u$-region consists of vertices whose corresponding configurations have the form $(q, i)$, where $i < |u|$, or where $i = |u|$ and $q \in \mathcal{Q}$. A $v$-region consists of vertices whose corresponding configurations have the form $(q, i)$, where $i > |u|$, or where $i = |u|$ and $q \in \overline{\mathcal{Q}}$ (see Fig. 1). We also require that on a path between vertices of a same $u$-region (resp. $v$-region) there are no vertices belonging to a different region. Recall that $\overline{\mathcal{Q}} \cap \mathcal{Q} = \emptyset$.

![Fig. 1. Computation tree for the initial configuration $quv$, after $k$ cross-overs from $u$ into $v$.](image-url)
Since the initial configuration is $qvw$, we start with the first $u$-region. The boundary of the first $u$-region bounds a tree which is the parse tree of the boolean expression $(q)[\rightarrow u \rightarrow ]$ (over the set of boolean variables $Q \cup Q$). Here $Q$ is identified with $Q \times \{|u|\}$; $q$ is used to label the configuration $(q, |u|)$. The computation tree could be infinite; this problem is handled as seen in the definition of $[\rightarrow u]$.

The computation now proceeds into the first $v$-region. The boundary of the first $u$-region together with the first $v$-region, bounds the parse tree for $(q)[\rightarrow u \rightarrow ][\overline{z}v]$. Next we go into the second $u$-region. The regions seen so far delimit a parse tree for $(q)[\rightarrow u \rightarrow ][\overline{z}v][u\overline{z}]$.

The computation goes on; in the limit the boolean expression (over $Q$) obtained is $\lim s_k$. To see that this limit is equal to $(q)[\rightarrow uv]$ we observe that if the number of $u$- and $v$-regions tends to infinity, the depth of computation tree tends to infinity, and conversely. Moreover, assume that the variables of the boolean expressions, i.e., the leaves of the tree, have been given a truth-value assignment; then the truth-value returned by one limiting process (truncating the tree at progressively greater depths) stabilizes to TRUE iff the value returned by the other limiting process (letting the number of $u$- and $v$-regions tend to infinity) stabilizes to TRUE. (When a limiting process stabilizes, that means that we do not need to consider the infinite tree, but just a finite truncation.)

Remark. As we saw in the proof, the role of $Q$ is to represent configurations of the form $(q, |u|)$, that occur at the boundary between $u$ regions and $v$-regions. On the other hand, $\overline{Q}$ and $\overline{Q}$ represent $\overline{Q} \times \{0\}$, respectively $\overline{Q} \times \{|uw|\}$ in the boolean expressions; so they should be kept in the end result (while $Q$ should be eliminated in the end result).

6. One-Tape Turing Machines with Bounded-Length Crossing Sequences

In [He] F. C. Hennie proves that deterministic one-tape linear-time Turing machines have crossing sequences of bounded length; and he proves that Turing machines with bounded-length crossing sequences accept only regular languages (and his proof of the latter carries over to the non-deterministic case). We shall show that such Turing machines can also be positionally simulated by deterministic two-way finite automata; we then extend this result to alternating one-tape linear-time Turing machines (this shows in particular that Hennie’s regularity result for bounded crossing sequences generalizes to the alternating case).

In [BH] M. Blum and C. Hewitt show that one-pebble two-way finite automata accept only regular languages; their proof uses Hennie’s result. Again, we show that this model can be positionally simulated by a two-way finite automaton.

Definitions

A non-deterministic Turing machine is said to be $k$-visiting (for some constant integer $k$) iff for every input $w$ that is accepted there exists an accepting computation during which every position of $w$ is visited at most $k$ times.
An alternating Turing machine is said to be *k-visiting* iff for every input $w$ that is accepted there exists a computation subtree (obtained by making just one choice at each $\exists$-node) which returns a stable value *TRUE* for all truncation depths $\geq k \cdot |w|$; moreover, we require that along every computation path of this subtree, truncated at depth $k \cdot |w|$, every position on $w$ is visited at most $k$ times.

We first prove Theorem 3.4: a *non-deterministic* one-tape $k$-visiting machine is positionally simulated by some *deterministic* two-way finite automaton ("2DFA"). The proof is done in two steps:

1. From the Turing machine $T$ (with input alphabet $\Sigma$), we build a deterministic two-way finite automaton $A$ (with input alphabet $\Delta$, to be defined) and a length-preserving surjective homomorphism $\phi: \Delta^* \rightarrow \Sigma^*$. Moreover, for every $y \in \Sigma^*$ and every position $i$ (with $0 \leq i \leq |y|$) we have: $T$, started on input $y$ at position $i$, accepts iff there exists $w \in \Delta^*$ with $y = \phi(w)$ such that $A$, started on an input $w$ at position $i$, accepts. (Recall $|w| = |\phi(w)|$, since $\phi$ is length-preserving.) Moreover, the movement of $A$ on an input $w \in \Delta^*$ is exactly the same as the movement of $T$ on $y = \phi(w) \in \Sigma^*$. (So $T$ is "almost positionally simulated" by $A$, except that the input alphabets are different, but related by the length-preserving morphism $\phi$.)

2. From $A$ we obtain another deterministic two-way finite automaton $B$ (this time with the same input alphabet as $T$) as described in the following lemma:

**Lemma 6.1.** For every two-way finite automaton $A$ (deterministic or non-deterministic) with input alphabet $\Delta$ and for every length-preserving surjective homomorphism $\phi: \Delta^* \rightarrow \Sigma^*$ there exists a *deterministic* two-way finite automaton $B$ with input alphabet $\Sigma$, such that for every state $q_1$ of $A$ there exists a state $q_2$ of $B$ for which we have: For all $y \in \Sigma^*$ and all positions $i$ (with $0 \leq i \leq |y|$), there exists $x \in \Delta^*$ with $\phi(x) = y$, such that: $A$ started on $x$ in state $q_1$ at position $i$ accepts iff $B$ started on $y$ in state $q_2$ at position $i$ accepts.

This lemma is a generalization of Theorem 3.1 (we obtain Theorem 3.1 if in the above lemma we let $\phi$ be the identity morphism and $\Sigma = \Delta$); it says that every non-deterministic two-way finite automaton can be positionally simulated by a deterministic two-way finite automaton, even if the two automata are related only via a length-preserving homomorphism. This is related to the following (vague) question (from [B2]): If $A$ is a two-way finite automaton accepting the language $L$ in $\Delta^*$ and if $\phi: \Delta^* \rightarrow \Sigma^*$ is a homomorphism, can one find a two-way finite automaton $B$ accepting $\phi(L)$ "without going all the way to one-way finite automata"?

If $A$ is obtained from $T$ as described before, and if $B$ is obtained from $A$ as in Lemma 6.1, then $B$ will positionally simulate $T$.

We will first show how $A$ (a 2DFA) is obtained from $T$ (a one-tape non-deterministic $k$-visiting Turing machine). Afterwards we will prove Lemma 6.1. This will complete the proof of Theorem 3.4, concerning non-deterministic $k$-visiting Turing machines.
Construction of a $2$DFA $A$ from a non-deterministic one-tape $k$-visiting TM $T$. The idea is to associate to each computation of $T$ ($T$ is a TM and hence, can print) a $k$-track picture of this computation; the $2$DFA $A$ takes such a $k$-track picture as an input and checks whether this computation is valid and accepting. The state set of $A$ will be $Q \times \{1, \ldots, k\}$ (where $Q$ is the state set of $T$); so $A$ remembers the state in which $T$ would be (in the computation that $A$ is checking) and also the current track number. The accept states of $A$ are $F \times \{1, \ldots, k\}$. Each time $T$ makes a turn (or “reversal”) in its computation, the $k$-track picture shows the continuation of the computation on the “next” track. In order to represent the computation of $T$ on an input $w$ by a $k$-track picture one needs to squeeze the computation into a $k \times |w|$ rectangle. Now a straight sweep of the head over a portion of $w$ will not necessarily be represented on one track but it will move on the “landscape” formed by the earlier sweeps and reversals. See Fig. 2 for an example. This idea is similar to the hint in [HU, Exercise 3.19, p. 73] (where a slightly different problem is considered).

Explanation of the various arrows in the track picture: $\downarrow$, indicates that the automaton moves over the cell from left to right and then goes to track $i$ (where $i$ is larger than the number of the current track); $\uparrow$, etc. have similar meanings. $\rightarrow$ indicates that the cell is read from left to right and that the previous track

|   | $a_1, q_1$ | $a_2, q_2$ | $a_3, q_3$ | $a_4, q_4$ | $a_5, q_5$ | $a_6, q_6$ | $a_7, q_7$ | $a_8, q_8$ | $a_9, q_9$ | $a_{10}, q_{10}$ | $a_{11}, q_{11}$ | $a_{12}, q_{12}$ | $a_{13}, q_{13}$ | $a_{14}, q_{14}$ | $a_{15}, q_{15}$ | $a_{16}, q_{16}$ | $a_{17}, q_{17}$ | $a_{18}, q_{18}$ | $a_{19}, q_{19}$ | $a_{20}, q_{20}$ | $a_{21}, q_{21}$ | $a_{22}, q_{22}$ | $a_{23}, q_{23}$ | $a_{24}, q_{24}$ | $a_{25}, q_{25}$ | $a_{26}, q_{26}$ | $a_{27}, q_{27}$ |
| 1 | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ |
| 2 | $b_2, q_{15}$ | $b_3, q_{10}$ | $b_4, q_9$ | $b_5, q_8$ | $b_6, q_7$ | $b_7, q_{24}$ | $b_8, q_{23}$ | $b_9, q_{22}$ | $b_{10}, q_{21}$ | $b_{11}, q_{20}$ | $b_{12}, q_{19}$ | $b_{13}, q_{18}$ | $b_{14}, q_{17}$ | $b_{15}, q_{16}$ | $b_{16}, q_{15}$ | $b_{17}, q_{14}$ | $b_{18}, q_{13}$ | $b_{19}, q_{12}$ | $b_{20}, q_{11}$ | $b_{21}, q_{10}$ | $b_{22}, q_{9}$ | $b_{23}, q_8$ | $b_{24}, q_7$ | $b_{25}, q_6$ | $b_{26}, q_5$ | $b_{27}, q_4$ | $b_{28}, q_3$ |
| 3 | $c_2, q_{16}$ | $c_3, q_{11}$ | $c_4, q_{12}$ | $c_5, q_{19}$ | $c_6, q_{20}$ | $c_7, q_{25}$ | $c_8, q_{26}$ | $c_9, q_{27}$ | $c_{10}, q_{26}$ | $c_{11}, q_{25}$ | $c_{12}, q_{24}$ | $c_{13}, q_{23}$ | $c_{14}, q_{22}$ | $c_{15}, q_{21}$ | $c_{16}, q_{20}$ | $c_{17}, q_{19}$ | $c_{18}, q_{18}$ | $c_{19}, q_{17}$ | $c_{20}, q_{16}$ | $c_{21}, q_{15}$ | $c_{22}, q_{14}$ | $c_{23}, q_{13}$ | $c_{24}, q_{12}$ | $c_{25}, q_{11}$ | $c_{26}, q_{10}$ | $c_{27}, q_9$ |
| 4 | $d_3, q_{14}$ | $d_4, q_{13}$ | $d_5, q_{12}$ | $d_6, q_{11}$ | $d_7, q_{10}$ | $d_8, q_{9}$ | $d_9, q_8$ | $d_{10}, q_7$ | $d_{11}, q_6$ | $d_{12}, q_5$ | $d_{13}, q_4$ | $d_{14}, q_3$ | $d_{15}, q_2$ | $d_{16}, q_1$ | $\uparrow$ | $\downarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ |
| 5 | $e_3, q_{17}$ | $e_4, q_{18}$ | $\downarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ | $\rightarrow$ |

Fig. 2. The $k$-track picture (here $k = 5$) of a computation of $T$, and the corresponding movement of the head.
was \(i\) (where \(i\) is less than the number of the current track). \(\downarrow_{j}^{i}\) indicates that the cell is read from left to right, that the previous track was \(i\), and that the next track is \(j\) (with \(i > \) current \( \text{track} > j\)). \(\uparrow_{i}^{j}\) is similar, except that here both \(i\) and \(j\) are less than the number of the current track.

The alphabet \(\Delta\) of \(\mathbb{A}\) is the cartesian product \((\Sigma \times Q \times D) \times (\Gamma \times Q \times D \cup \{\varepsilon\})^{k-1}\) where

\[
D = \{\rightarrow, \leftarrow, \downarrow_{i}, \uparrow_{i}, \downarrow_{i}^{j}, \uparrow_{i}^{j}, \downarrow_{i}^{j}, \uparrow_{i}^{j}\}
\]

\(\Gamma\) is the working alphabet of \(\mathcal{T}\) and \(Q\) is the state set of \(\mathcal{T}\). A letter of \(\Delta\) is thus a \(k\)-track column (each cell of which contains an element of \(\Gamma \times Q \times D\) or is empty).

We define the homomorphism \(\varphi: A^{*} \rightarrow \Sigma^{*}\) by mapping every element of \(A\) to the \(Z\)-coordinate (the first component of the cartesian product).

It is clear from Fig. 2 how \(\mathbb{A}\) operates. The details would be tedious, and are omitted.

**Proof of Lemma 6.1.** The proof is a straightforward application of Main Lemma 3.2. Let \(S_{1} = \{[w] | w \in A^{*}\}\) be the monoid of the two-way automaton \(\mathbb{A}\) and let \(S = \mathbb{P}(S_{1})\) be the power monoid over \(S_{1}\) (the elements of \(S\) are the subsets of \(S_{1}\), and the multiplication is \(X \cdot Y = \{x \cdot y | x \in X, y \in Y\}\).

We consider the homomorphism \(\pi: \Sigma^{*} \rightarrow S\) defined by \(\pi(u) = [\varphi^{-1}(u)] = \{[w] | w \in S_{1}, \varphi(w) = u\} \in S\). This is indeed a homomorphism, since \(\varphi: A^{*} \rightarrow \Sigma^{*}\) is length-preserving.

By Main Lemma 3.2, there exists a deterministic two-way finite automaton which, when started on input \(uv \in \Sigma^{*}\) at position \(|u|\), computes the pair \(([\varphi^{-1}(u)]), ([\varphi^{-1}(v)])\) \(\in S \times S\). The knowledge of \([\varphi^{-1}(u)]\), \([\varphi^{-1}(v)]\), and of the starting state \(q_{1}\) of \(\mathbb{A}\), is sufficient to determine whether there exists \(u, v \in A^{*}\) with \(\varphi(u) = u, \varphi(v) = v\), such that \(\mathbb{A}\) accepts \(uvq_{1}v\).

Now, just as in the proof of Theorem (3.1), we can construct a 2DFA \(\mathbb{B}\) with the properties required in Lemma 6.1. 

**Positional simulation of a One-pebble Finite Automaton by a 2DFA**

In [BH, p. 159] M. Blum and C. Hewitt show that a 2NFA with one pebble (which it can leave as a marker, pick up, and transport around) only recognizes regular languages. The proof they give (due to Albert Meyer), is easily modified to show positional simulation by a 2DFA (without a pebble). First, one positionally simulates the one-pebble 2NFA by a 2-visited one-tape non-deterministic Turing machine (which "prints tables on the tape" as in [BH], except that it also prints a special symbol at the initial position of the reading head, so as to be able to again
find its initial position). Next one positionally simulates this Turing machine by a 2DFA as above. This proves part 2 of Theorem 3.4.

**Alternating One-Tape Turing Machines with Bounded-Length Crossing Sequences**

We outline a proof of Theorem 3.5. Let $T$ be an alternating one-tape Turing machine which is $k$-visiting, for some $k \geq 1$.

First, we represent each complete computation tree of $T$ by a $k$-track "many-sheeted" picture: a 2AFA $A_1$ takes such a picture as an input and checks (using alternation, but no printing) whether the corresponding computation tree is valid and accepting. Each time $T$ makes a turn in a computation path, the $k$-track image shows the continuation of the computation on the "next" track (just as previously, see Fig. 2). If $T$ does an $\exists$- or $\forall$-branching, the picture uses a new sheet for each branching choice; at every position on the input tape and in each track, there are at most $|Q| \cdot |\Gamma|$ overlaid cells, each with a different content $\in Q \times \Gamma$ (here $Q$ is the state set and $\Gamma$ is the work alphabet of $T$). So at every position of the input tape, we have a pile of $|Q \times \Gamma|^k$ columns of length $k$. In Fig. 2, we described the movement of $T$ by arrows in the cells. Now, however, because of the branching, we must draw arrows from a cell to every other cell that $T$ can go to in one step. The alphabet $A$ of $A_1$ is thus obtained by taking $|Q \times \Gamma|^k$ and concatenating it with the set of all directed graphs on $2 \cdot |Q \times \Gamma|^k$ vertices. See Fig. 3 for an example. This yields a 2AFA $A_1$ which accepts the picture of the computation tree of $T$ on an input $w$ iff the corresponding tree is accepting. The 2AFA $A_1$ can be positionally simulated by a 2DFA $B$ (using Theorem 3.3).

Finally, we define a homomorphism $\varphi: \Sigma^* \rightarrow \Sigma^*$ (in a similar way as in the case when $T$ was just non-deterministic) and apply Lemma 6.1 to obtain 2DFA $B$ which positionally simulates $T$.

---

**Fig. 3.** Two piles of columns of the $k$-track picture (with $|Q \times \Gamma|^k$-sheeted cells) of the computation tree of $T$ (here $k = 3$); the two piles are connected by a directed graph, indicating the movement.
7. MISCELLANEOUS

Automata for Which the Appropriate Generalization of Kannan's Conjecture Does not hold

Here are two examples where the non-deterministic automaton is more powerful than the deterministic one (for language recognition, and hence, positional simulation):

Four-way finite automata on a two-dimensional tape; here positional simulation is a very natural concept, but non-determinism is more powerful (see [BH, p. 1561]).

Two-way finite automata on a tape which is infinite in both directions; one easily constructs examples where non-determinism is more powerful than determinism. (Here a two-sidedly infinite word is accepted if there are infinitely many occurrences of accept states at different positions in the left and in the right directions, for some computation path.)

Decidability of Positional Simulation

The following question is decidable: Given any two 2DFA's \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \), is \( \mathcal{A}_1 \) positionally simulated by \( \mathcal{A}_2 \) ?

Indeed, \( \mathcal{A}_1 \) is not positionally simulated by \( \mathcal{A}_2 \) iff one can choose a start state of \( \mathcal{A}_1 \) such that for every possible starting state of \( \mathcal{A}_2 \) there exists an input \( w \) of length \( \leq 2 \cdot n_1^1 \cdot n_2^2 \) and a starting position \( \leq n_1^1 \cdot n_2^2 \) on that input, for which \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) differ (regarding acceptance). This is because on a longer word there would be repeated pairs of crossing sequences on the same side of the starting position (a pair of crossing sequences consists of a crossing sequence of \( \mathcal{A}_1 \) together with a crossing sequence of \( \mathcal{A}_2 \), at the same tape position); so this long word on which \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) disagree can be replaced by a shorter word on which \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) disagree.

The same question is also decidable for \( \text{2NFAS} \), \( \text{2AFAS} \), etc., since by our previous results all these can be replaced by \( \text{2DFAS} \).

Complexity. These problems are P-SPACE hard (since the emptiness problem for \( \text{2DFAS} \) is already P-SPACE complete; see H. B. Hunt [H, GJ, p. 265]).

Open Problem

(Kannan) Improve the lower bound given in the second theorem of Section 1. This paper gives \( 2^{O(n^3)} \) as an upper bound. I would guess that the best lower bound is fairly close to this upper bound.

Acknowledgments and Personal Remarks

Douglas Albert first told me about Kannan's conjecture and suggested that I work on it. My first impression was that is was "obviously wrong," and I looked for counterexamples of the following form:
a 2NFA non-deterministically chooses to go right or left from its starting position; going left it looks for a pattern (say aaa), going right it looks for another pattern (say bbb). The 2NFA never reverses its motion after its initial choice of a direction; it accepts if one of the choices leads to the appropriate pattern. At first it seems that a 2DFA is unable to positionally simulate this 2NFA, because the 2DFA cannot remember the initial position: if it finds the pattern aaa or bbb one wonders how it could know whether the pattern is to the left or to the right of the initial position. However (exercise), a 2DFA can positionally simulate this 2NFA—and this made me change my opinion. One might also say that if Kannan's conjecture had been wrong it would not have been so difficult to find the lower bounds of \([K]\) (see Section 1). I worked out more examples of the above type (where the 2NFA does non-deterministic choices at the start and then runs deterministically and without reversal). David Klarner suggested that I first prove the conjecture for that type of 2NFAs in general. I did that, and the proof was very similar to the proof of what became Main Lemma 3.2. I had written to Ravi Kannan, and he kindly sent me some notes with examples (done with J. Hopcroft); they were of a very different type but one could recognize special cases of the idea of the "right boundary." It still took me a while to realize that the special case of the conjecture that I had proved could be turned into a semigroup result (namely, Main Lemma 3.2) and that this would then directly prove Kannan's conjecture, in general (using the monoid of \([B.2]\) associated to a 2NFA).

REFERENCES


