A Survey of Multi-Tape Automata

Carlo A. Furia

May 2012

Abstract

This paper summarizes the fundamental expressiveness, closure, and decidability properties of various finite-state automata classes with multiple input tapes. It also includes an original algorithm for the intersection of one-way nondeterministic finite-state automata.

Contents

1 Overview

This paper is a survey of the known results about the expressiveness, closure, and decidability properties of finite-state automata that read multiple input tapes. The theoretical study of these computing devices began within the classic work of Rabin and Scott [16], Elgot and Mezei [5], and Rosenberg [17, 18, 6]. In recent years, multi-tape automata have found some applications such as automatic structures [13, 20, 2, 19], querying string databases [7], and weighted automata for computational linguistic [12, 3].

Section 2 gives a general definition of multi-tape automata that allows for nondeterministic, asynchronous, and two-way movements of the input heads. The following sections analyze the expressiveness, closure properties, and decidability for increasing levels of generality: Section 3 briefly recalls the well-known properties of single-tape automata; Section 4 discusses synchronous automata, where different input heads are not allowed to be in arbitrary portions of their respective tapes; and Section 5 considers the most general case of fully asynchronous heads. Section 6 gives a synopsis of the fundamental properties surveyed.

Finally, Section 7 presents an original algorithm for the intersection of asynchronous one-way multi-tape automata. Since this class is not closed under intersection, the algorithm may not terminate in the general case.

2 Definitions

2.1 Alphabets, Words, and Tuples

$\mathbb{Z}$ is the set of integer numbers, and $\mathbb{N}$ is the set of natural numbers $0, 1, \ldots$. For a (finite) set S, $\wp(S)$ denotes its powerset. For a finite nonempty *alphabet* Σ, Σ^* denotes the set of all finite sequences $\sigma_1 \ldots \sigma_n$, with $n \geq 0$, of symbols from Σ (also called *words* over Σ); when $n = 0$, $\epsilon \in \Sigma^*$ is the *empty* word. $|s| \in \mathbb{N}$ denotes the length n of a word $s = \sigma_1 \ldots \sigma_n$. An *n-word* is an n-tuple $\langle x_1, \ldots, x_n \rangle \in (\Sigma^*)^n$ of words over Σ, where each $x_i = \sigma_{1,i} \ldots \sigma_{n_i,i}$.

The *convolution* of n words $\langle x_1, \ldots, x_n \rangle$ is a word over the padded alphabet $(\Sigma \cup \{\Box\})^n$:

$$ x_1 \otimes \cdots \otimes x_n = \begin{bmatrix} y_1^1 \\ \vdots \\ y_n^1 \end{bmatrix} \cdots \begin{bmatrix} y_1^m \\ \vdots \\ y_n^m \end{bmatrix}, $$

where m is the maximum length $\max\{|x_1|, \ldots, |x_n|\}$, and

$$ y_k^h = \begin{cases} \text{the } h\text{-th component of } x_{h,k} \text{ of } x_k & h \leq |x_k|, \\ \Box & \text{otherwise}. \end{cases} $$

Given two words $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_m$, $x \circ y$ denotes their *concatenation* $x_1 \ldots x_n \, y_1 \ldots y_m$, and $rev(x)$ denotes the *reversal* $x_n \ldots x_1$ of x. These operations on words are naturally lifted to sets of words and to n-words; for example, $W \circ X = \{w \circ x \mid w \in W, x \in X\}$ denotes the concatenation of the two sets of words W and X; $rev(()x) = \langle rev(x_1), \ldots, rev(x_n) \rangle$ denotes the reversal of the n-word $x = \langle x_1, \ldots, x_n \rangle$. Also, W^* denotes the *Kleene closure* with respect to finite self-concatenation, that is $W^* = \bigcup_{k \in \mathbb{N}} W^k$, where

$$ W^k = \overbrace{W \circ W \circ \cdots \circ W}^{k}. $$

Given a set W of n-words over Σ, $\overline{W}$ is the *complement* set $(\Sigma^*)^n \setminus W$; $\exists_k W$ is the *projection* set of $n-1$-words obtained by projecting out W's k-th component, that is

$$\exists_k W = \left\{ \langle y_1, \ldots, y_{n-1} \rangle \mid \begin{array}{l} \exists y \langle x_1, \ldots, x_{k-1}, y, x_{k+1}, \ldots x_n \rangle \in W, \\ x_1 = y_1, \ldots, x_{k-1} = y_{k-1}, \\ x_{k+1} = y_k, \ldots, x_n = y_{n-1} \end{array} \right\};$$

$\forall_k W$ is the *generalization* set of $n-1$-words obtained for every value of W's k-th component, that is

$$\forall_k W = \left\{ \langle y_1, \ldots, y_{n-1} \rangle \mid \begin{array}{l} \forall y \langle x_1, \ldots, x_{k-1}, y, x_{k+1}, \ldots x_n \rangle \in W, \\ x_1 = y_1, \ldots, x_{k-1} = y_{k-1}, \\ x_{k+1} = y_k, \ldots, x_n = y_{n-1} \end{array} \right\};$$

and the convolution of W is the set of words

$$W^{\otimes} = \{ x_1 \otimes \cdots \otimes x_n \mid \langle x_1, \ldots, x_n \rangle \in W \}.$$

2.2 Multi-Tape Finite Automata

A two-way finite-state automaton with $n \geq 1$ tapes scans n read-only input tapes, each with an independent head. At every step, the transition function determines the possible next states and head movements, based on the current state and the symbols currently under each head. Two special symbols $\triangleright, \triangleleft$ respectively mark the left and right ends of each input tape; $\Sigma_{\triangleright\triangleleft}$ denotes the extended alphabet $\Sigma \cup \{\triangleright, \triangleleft\}$.

Definition 1. A *two-way nondeterministic finite-state automaton with n tapes* is a tuple $\langle \Sigma, Q, \delta, q_0, F \rangle$, where:

- Σ is the input alphabet, such that $\triangleright, \triangleleft \notin \Sigma$;

- Q is the finite set of states;

- $\delta : Q \times \Sigma_{\triangleright\triangleleft}^n \to \wp(Q \times \{-1, 0, 1\}^n)$ is the transition function that maps current state and input to a set of next states and head movement directions, with the restriction that the head does not move past the end markers;

- $q_0 \in Q$ is the initial state;

- $F \subseteq Q$ is the set of accepting states.

The semantics of two-way nondeterministic finite-state automata relies on the notion of *configuration*. Given a two-way nondeterministic finite-state automaton A as in Definition 1, a configuration of A is a $(2n+1)$-tuple

$$\langle x_1, \ldots, x_n, q, i_1, \ldots, i_n \rangle \in (\triangleright \Sigma^* \triangleleft)^n \times Q \times \mathbb{N}^n,$$

where $q \in Q$ is the current state, and, for $1 \leq k \leq n$, x_k is the content of the k-th tape and $0 \leq i_k \leq |x_k| + 1$ is the position of the k-th head; when $i_k = 0$ (resp., $i_k = |x_k| + 1$) the head is on the left marker $\triangleright$ (resp., right marker $\triangleleft$).

The transition relation $\vdash$ between configurations is defined as:

$$\langle x_1, \ldots, x_n, q, i_1, \ldots, i_n \rangle \vdash \langle x_1, \ldots, x_n, q', i'_1, \ldots, i'_n \rangle$$

if and only if, for each $1 \le k \le n$, $\sigma_{i_k,k}$ is the symbol at position i_k in input word x_k, $\delta(q, \sigma_{i_1,1}, \ldots, \sigma_{i_n,n})$ includes a tuple $\langle q', d_1, \ldots, d_n \rangle$, and $i'_k = i_k + d_k$ for each $1 \le k \le n$.

A *run* ρ of A on input $x = \langle x_1, \ldots, x_n \rangle$ is a sequence of configurations $\chi_0 \ldots \chi_m$ such that $\chi_0 = \langle x, q_0, 0^n \rangle$ and, for all $1 \le k \le m$, $\chi_{k-1} \vdash \chi_k$. A run ρ of A on input x is *accepting* if $\chi_m = \langle x, q, i_1, \ldots, i_n \rangle$ for some $q \in F$ and the i_k-th character of the k-th tape is $\lhd$ (that is, every head has reached the end of its tape). Correspondingly, A accepts an input word x if there is an accepting run ρ of A on x. The *language* accepted (or recognized) by A is the set of n-words

$$\mathcal{L}(A) = \{\, x \in (\Sigma^*)^n \mid A \text{ accepts } x \,\} \ .$$

Definition 2. An n-tape automaton A is:

- *deterministic* if $|\delta(q, \sigma_1, \ldots, \sigma_n)| \le 1$ for any $q, \sigma_1, \ldots, \sigma_n$;

- *s-synchronized* for $s \ge 0$ if every run of A, accepting or not, is such that any two heads that are not on the right-end marker $\lhd$ are no more than s positions apart (as measured from the left-end marker $\rhd$);

- *synchronized* if it is s-synchronized for *some* $s \in \mathbb{N}$;

- *asynchronous* if it is *not* synchronized;

- *synchronous* if it is 0-synchronized;

- *r-reversal bounded* for $r \ge 0$ if every run of A, accepting or not, is such that any head never inverts its direction of motion more than r times;

- *reversal-bounded* if it is r-reversal bounded for *some* $r \in \mathbb{N}$;

- *one-way* if it is 0-reversal bounded, that is δ never moves any head left;

- *with rewind* if it only makes a special type of reversals where, after all heads reach the right-end marker $\lhd$, they are simultenously rewinded to the left-end marker $\rhd$, where the computation continues;

- *r-rewind bounded* for $r \ge 0$ if it is $2r$-reversal bounded with rewind;

- *rewind-bounded* if it is r-rewind bounded for *some* $r \in \mathbb{N}$.

Correspondingly, $\mathsf{NFA}(n, s, r)$ denotes the class of s-synchronized r-reversal bounded n-tape two-way nondeterministic finite-state automata, $\mathsf{DFA}(n, s, r)$ the corresponding deterministic class, and when $s = \infty$ (resp. $r = \infty$) the class includes asynchronous (resp. two-way reversal-unbounded) automata as well. Finally, for $\mathsf{R} \in \mathbb{N} \cup \{\infty\}$, $\mathsf{NFA}\!\left(n, s, \widehat{\mathsf{R}}\right)$ and $\mathsf{DFA}\!\left(n, s, \widehat{\mathsf{R}}\right)$ denote the classes of nondeterministic and deterministic s-synchronized R-rewind bounded (or unbounded if $\mathsf{R} = \infty$) n-tape two-way finite-state automata.

With a little abuse of notation, the notation for automata classes also denotes to the corresponding class of accepted languages; for example, $\mathsf{NFA}(3, \infty, 4)$ is also the class of languages recognized by nondeterministic 3-tape machines with at most 4 reversals and with no synchronization restrictions.

2.3 Weak Inclusions

A number of weak inclusions are a direct consequence of the definitions. In the following, let $n \geq 1$ be a number of tapes, $s, r \in \mathbb{N}$ be nonnegative integers, and $\text{S}, \text{R} \in \mathbb{N} \cup \{\infty\}$ be nonnegative integers or infinity. Determinism is a syntactic restriction, thus:

$$\text{DFA}(n, \text{S}, \text{R}) \subseteq \text{NFA}(n, \text{S}, \text{R}) \; ; \tag{1}$$

and similarly, bounding the number of reversals reduces generality:

$$\text{DFA}(n, \text{S}, r) \subseteq \text{DFA}(n, \text{S}, r+1) \subseteq \text{DFA}(n, \text{S}, \infty) \; , \tag{2}$$
$$\text{NFA}(n, \text{S}, r) \subseteq \text{NFA}(n, \text{S}, r+1) \subseteq \text{NFA}(n, \text{S}, \infty) \; ; \tag{3}$$

and so does limiting the degree of synchrony:

$$\text{DFA}(n, s, \text{R}) \subseteq \text{DFA}(n, s+1, \text{R}) \subseteq \text{DFA}(n, \infty, \text{R}) \; , \tag{4}$$
$$\text{NFA}(n, s, \text{R}) \subseteq \text{NFA}(n, s+1, \text{R}) \subseteq \text{NFA}(n, \infty, \text{R}) \; . \tag{5}$$

Rewinds are a restricted form of reversals, hence:

$$\text{DFA}\left(n, \text{S}, \overset{\frown}{\text{R}}\right) \subseteq \text{DFA}(n, \text{S}, 2\text{R}) \; , \tag{6}$$
$$\text{NFA}\left(n, \text{S}, \overset{\frown}{\text{R}}\right) \subseteq \text{NFA}(n, \text{S}, 2\text{R}) \; . \tag{7}$$

2.4 Decision Problems

Given a class $\mathcal{C}$ of languages, and n-tape automata A, B, consider the following decision problems:

Emptiness: does $\mathcal{L}(A) = \emptyset$?

Universality: does $\mathcal{L}(A) = (\Sigma^*)^n$?

Finiteness: is $\mathcal{L}(A)$ finite?

Disjointness: does $\mathcal{L}(A) \cap \mathcal{L}(B) = \emptyset$?

Inclusion: does $\mathcal{L}(A) \subseteq \mathcal{L}(B)$?

Equivalence: does $\mathcal{L}(A) = \mathcal{L}(B)$?

Class membership: does $\mathcal{L}(A) \in \mathcal{C}$?

3 Single-Tape Automata

This section recalls the well-known properties of single-tape automata (that is, with $n = 1$ tapes), which define the class of *regular* languages [16, 10, 14]. Clearly, the notion of synchronization is immaterial with a single tape:

$$\text{DFA}(1, 0, 0) = \text{NFA}(1, 0, 0) = \text{DFA}(1, \text{S}, \text{R}) = \text{NFA}(1, \text{S}', \text{R}') \; . \tag{8}$$

Also, languages defined by one-tape automata are closed under complement, intersection, union, concatenation, Kleene closure, projection, generalization, and reversal. The emptiness, universality, finiteness, disjointness, inclusion, equivalence, and class membership problems are all decidable for regular languages.

4 Synchronized Multi-Tape Automata

This section studies several properties of synchronized multi-tape automata $\mathsf{NFA}(n, s, \textsc{r})$ and $\mathsf{DFA}(n, s, \textsc{r})$, where s denotes a generic value in $\mathbb{N}$, and $\textsc{r}$ a generic value in $\mathbb{N} \cup \{\infty\}$. As we will see, synchronization essentially makes multi-tape automata defining regular languages over convolutions, hence synchronized multi-tape automata define a very robust class of languages.

4.1 Expressiveness

This section shows that:

$$\mathsf{DFA}(n, 0, 0) = \mathsf{NFA}(n, 0, 0) = \mathsf{DFA}(n, s, \textsc{r}) = \mathsf{NFA}(n, s, \textsc{r}) \ . \tag{9}$$

Since rewinds are a special case of reversals, it also follows that:

$$\mathsf{DFA}(n, 0, 0) = \mathsf{NFA}(n, 0, 0) = \mathsf{DFA}\left(n, s, \overset{\frown}{\textsc{r}}\right) = \mathsf{NFA}\left(n, s, \overset{\frown}{\textsc{r}}\right) \ . \tag{10}$$

4.1.1 Increasing Synchronization

Every synchronized n-tape automaton can be transformed to an equivalent synchronous n-tape automaton:

$$\mathsf{NFA}(n, s, \textsc{r}) = \mathsf{NFA}(n, 0, \infty) \ . \tag{11}$$

The construction to translate an $A \in \mathsf{NFA}(n, s, \textsc{r})$ into an $A' \in \mathsf{NFA}(n, 0, \textsc{r})$ generalizes the one in [11] to two-way nondeterministic automata; to our knowledge, it is original to the present survey. It works as follows: additional states in the synchronous automaton A' keep track of the "neighbour" s symbols on each tape scanned during a synchronous reading of the tapes.

Precisely, the states Q' of A' are of the form $[q, b_1, \ldots, b_n]$, where each b_k is a word of at most $s + 1$ characters over $\Sigma_{\rhd\lhd} \cup \{\downarrow\}$, with exactly one occurrence of the special symbol $\downarrow$. For any such words b, $\downarrow(b)$ denotes the character immediately to the right of $\downarrow$, or ϵ if $\downarrow$ is the right-most character in b.

Then, a configuration of A' where the symbol currently under the k-th head is σ_k and the current state is $[q, b_1, \ldots, b_n]$, corresponds to a configuration of A where the symbol currently under the k-th head is $\downarrow(b_k \cdot \sigma_k)$—that is the one immediately to the right of $\downarrow$ in $b_k \cdot \sigma_k$—and the current state is q.

The initial state q_0' of A' is $[q_0, \downarrow^n]$, where all heads are on the left marker. During any computation over a given input, the (common) position of the heads in A' corresponds to the position of the right-most head in A; more precisely, the states of A' plus the input describe a $n \times (k + 1)$ tape portion:

$$
\begin{array}{cccccc}
\alpha_1 & x_1 & \downarrow & y_1 & \beta_1 & \sigma_1' \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\alpha_k & x_k & \downarrow & y_k & \beta_k & \sigma_k' \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\alpha_n & x_n & \downarrow & y_n & \beta_n & \sigma_n' \ ,
\end{array}
$$

where $\alpha_k \in \Sigma_{\rhd\lhd}^*$ is the left-most stored portion of the k-th tape, $x_k \in \{\epsilon\} \cup \Sigma_{\rhd\lhd}$ is the character immediately to the left of the marker, $y_k \in \{\epsilon\} \cup \Sigma_{\rhd\lhd}$ is the character

immediately to the right of the marker, $\beta_k \in \Sigma^*_{\triangleright\triangleleft}$ is the right-most stored portion of the k-th tape, and $\sigma'_k \in \Sigma_{\triangleright\triangleleft}$ is the charater currently under the k-th head. The hypothesis that A' is s-bounded entails that $2 \le |\alpha_k\, x_k \downarrow y_k\, \beta_k\, \sigma'_k| \le s + 2$. Finally, when a head is on the right marker $\triangleleft$, we call it "inactive", and we do not update the corresponding portion of the state. Write $\mathcal{I}$ to denote the set of inactive heads in the current state (that is, the set of heads h such that $a_h \cdot \sigma_h \in \Sigma^* \triangleleft$ and $a_h x_h \ne \epsilon$), and $\mathcal{H} = \{1, \ldots, n\} \setminus \mathcal{I}$ to denote the other active heads.

Let us now see how to construct the transitions of A'. For each of A's transitions

$$\langle q', d_1, \ldots, d_n \rangle \in \delta(q, \sigma_1, \ldots, \sigma_n),$$

consider all states of A' of the form $[q, a_1, \ldots, a_n]$. A' should make a transition from such current state according to the characters currently marked by $\downarrow$, corresponding to the σ_i, and then adjust the state to reflect the head movements in A. If all *active* heads move left in A (that is, $\max\{d_j \mid j \in \mathcal{H}\} = -1$) and there exist an active tape $\overline{k}$ whose s-size buffer encoded in the state has the marker $\downarrow$ in the leftmost position ($\alpha_{\overline{k}} = x_{\overline{k}} = \epsilon$), then it is not sufficient to rearrange the information in $[q, a_1, \ldots, a_n]$ to determine the next state A' moves to, because the new character the head moves is to $s + 1$ characters away from the current position of the head in A'. Hence, in these situations, A' enters a sequence of *lookup* states $\ell_1^-, \ldots, \ell_{s+1}^-$ while moving all its active heads left, until it reads the $(s+1)$-th character to the left, so that it can update the state accordingly. Precisely, the following transitions in A' define a lookup, for any characters $\mu_1, \ldots, \mu_n$ read while moving heads.

- Enter the lookup state ℓ_1^- and start moving left:

  $$\langle [\ell_1^-, a_1, \ldots, a_n], \lambda_1^{-1/0}, \ldots, \lambda_n^{-1/0} \rangle \in \delta'([q, a_1, \ldots, a_n], \mu_1, \ldots, \mu_n),$$

  where

  $$\lambda_j^{a/b} = \begin{cases} a & j \in \mathcal{H}, \\ b & \text{otherwise}, \end{cases}$$

  hence inactive heads are not moved.

- Traverse all lookup states, for $1 \le h \le s$, while moving left:

  $$\langle [\ell_{h+1}^-, a_1, \ldots, a_n], \lambda_1^{-1/0}, \ldots, \lambda_n^{-1/0} \rangle \in \delta'([\ell_h^-, a_1, \ldots, a_n], \mu_1, \ldots, \mu_n)$$

- Include the current characters $\mu_1, \ldots, \mu_n$ in the state and start moving right on the "duplicate" lookup state ℓ_s^+:

  $$\langle [\ell_s^+, \nu_1, \ldots, \nu_n], \lambda_1^{1/0}, \ldots, \lambda_n^{1/0} \rangle \in \delta'([\ell_{s+1}^-, a_1, \ldots, a_n], \mu_1, \ldots, \mu_n),$$

  where ν_k is $\lambda_k^{\mu_k/\epsilon} \cdot a_k$, with the last character removed if $|\lambda_k^{\mu_k/\epsilon} \cdot a_k| > s + 1$.

- Move right, for $s \ge h > 1$, until you reach ℓ_1^+:

  $$\langle [\ell_{h-1}^+, \nu_1, \ldots, \nu_n], \lambda_1^{1/0}, \ldots, \lambda_n^{1/0} \rangle \in \delta'([\ell_h^+, \nu_1, \ldots, \nu_n], \mu_1, \ldots, \mu_n)$$

- Move to the original position on q:

  $$\langle [q, \nu_1, \ldots, \nu_n], \lambda_1^{1/0}, \ldots, \lambda_n^{1/0} \rangle \in \delta'([\ell_1^+, \nu_1, \ldots, \nu_n], \mu_1, \ldots, \mu_n).$$

8

Notice that, after the lookup, the tuple $\nu_1, \ldots, \nu_n$ is certainly different than the starting tuple $a_1, \ldots, a_n$, because we have added a left-most character to the left of the marker $\downarrow$ in at least one of $a_1, \ldots, a_n$. Hence, the lookup does not introduce nondeterminism.

After setting up the lookup when necessary, let us now describe normal transitions of A'. Let $[q, a_1, \ldots, a_n]$ be the current state; add to A' all transitions of the form

$$\langle [q', a'_1, \ldots, a'_n], \lambda_1^{d'/0}, \ldots, \lambda_n^{d'/0} \rangle \in \delta'([q, a_1, \ldots, a_n], \sigma'_1, \ldots, \sigma'_n)$$

such that, for $1 \leq k \leq n$:

- $d' = \max\{d_j \mid j \in \mathcal{H}\}$;

- $\sigma_k = \downarrow(a_k \cdot \sigma'_k)$ or, if $\downarrow$ does not appear in a_k (this may only happen after a lookup when the $\downarrow$ occurred at the last position in a_k), $\sigma_k = \sigma'_k$;

- Let γ_k be $a_k \cdot \sigma'_k$ with $\downarrow$ moved right, left, or not moved, according to whether d_k is 1, −1, or 0; formally:

$$\gamma_k = \begin{cases} \alpha_k \, x_k \, y_k \downarrow \beta_k \, \sigma'_k & d_k = 1 \text{ and } y_k \neq \epsilon \,, \\ \alpha_k \, x_k \, \sigma'_k \downarrow & d_k = 1 \text{ and } y_k = \epsilon = \beta_k \,, \\ \alpha_k \, x_k \downarrow y_k \, \beta_k \, \sigma'_k & d_k = 0 \,, \\ \alpha_k \downarrow x_k \, y_k \, \beta_k \, \sigma'_k & d_k = -1 \text{ and } \downarrow \text{ appears in } a_k \cdot \sigma'_k \,, \\ a_k \downarrow \sigma'_k & d_k = -1 \text{ and } \downarrow \text{ doesn't appear in } a_k \cdot \sigma'_k \,. \end{cases}$$

Then, a'_k is:

$$a'_k = \begin{cases} \gamma_k & |\gamma_k| \leq s+1 \text{ and } d' = 1 \,, \\ \gamma_k \text{ without the first character} & |\gamma_k| > s+1 \text{ and } d' = 1 \,, \\ \gamma_k \text{ without the last character } \sigma'_k & d' \leq 0 \,. \end{cases}$$

Remarks.

- With this construction the number $|Q'|$ of states of A' is $O(|Q| \cdot |\Sigma|^{n \cdot s} \cdot s|\Sigma|)$, because there are at most $|\Sigma|^{n \cdot s}$ n-tuples of words of length at most s for each state in Q, plus a multiplicative factor $s \cdot |\Sigma|$ to account for the lookup states ($2s + 1$, for any possible left-most characters $\mu_k \in \Sigma_{\triangleright\triangleleft}$).

- The lookups may introduce additional reversals, therefore the construction does not, in general, preserve the original value R of reversals.

- Automaton A' is deterministic if and only if A is. Therefore, the same construction proves:
$$\mathsf{DFA}(n, s, \mathrm{R}) = \mathsf{DFA}(n, 0, \infty) \,. \tag{12}$$

4.1.2 Removing Reversals

The classic constructions to turn two-way automata into equivalent one-way automata (such as Shepherdson's [21] or Vardi's [22]) are applicable to multi-tape synchronous automata as well. The only minimal difference is that a multi-tape one-way automaton is not forced to always move its heads right, because the heads reaching the right-hand marker $\triangleleft$ cannot move right at all. In all, the following result holds:

$$\mathsf{NFA}(n, 0, \mathrm{R}) = \mathsf{NFA}(n, 0, 0) \,. \tag{13}$$

4.1.3 Determinization

The usual construction to determinize nondeterministic finite-state automata works for multi-tape synchronous automata too:

$$\mathsf{DFA}(n,0,0) = \mathsf{NFA}(n,0,0) \ . \tag{14}$$

4.1.4 Convolution

The computations of synchronous n-tape one-way automata can be regarded as computations of single-tape automata over n-track alphabets. More precisely, let $\mathcal{L}(A)$ be the language over alphabet Σ accepted by some synchronous n-tape one-way automaton $A \in \mathsf{NFA}(n,0,0)$. Then, there exists a single-tape one-way automaton B that accepts the language $\mathcal{L}(A)^{\otimes}$.

4.2 Closure Properties

Since synchronized automata essentially define regular languages via convolution, they enjoy the same closure properties. Thus, the usual constructions [10] show that $\mathsf{NFA}(n,0,0)$ is closed under under complement, intersection, union, concatenation, Kleene closure, projection, generalization, and reversal.

4.3 Decidability

Decidability of most decision problems considered in Section 2.4 is also a straightforward consequence of the regular nature of synchronized automata: for example, *emptiness* is decidable: construct the automaton recognizing $\mathcal{L}(A)^{\otimes}$ and determine if it accepts some word; alternatively, directly test reachability of an accepting state. Similarly, it is easy to see that universality, finiteness, disjointness, inclusion, and equivalence are decidable for synchronized automata.

Next, we consider some class membership problems. Ibarra and Tran [11, Th. 15] show that, for generic two-way automaton $A \in \mathsf{NFA}(n,\infty,\infty)$ and $s \geq 0$, it is decidable to determine whether $A \in \mathsf{NFA}(n,s,\infty)$, that is whether A is s-synchronized. The idea is to build an automaton M that simulates the computations of A on a single tape that stores the convolution of the input; whenever M detects a computation where a pair of heads would "separate" more than s cells apart, it accepts, and otherwise it rejects. Since M accepts the empty language if and only if A is not s-synchronized, and emptiness for M is decidable (as M is a single-tape automaton), it follows that we have an effective procedure to determine whether $A \in \mathsf{NFA}(n,s,\infty)$. The same procedure is applicable to automata with bounded reversals, hence it is also decidable whether $A \in \mathsf{NFA}(n,s,\text{R})$ for given $s \in \mathbb{N}$ and $\text{R} \in \mathbb{N} \cup \{\infty\}$.

Ibarra and Tran [11, Th. 14] also show—via reduction from the halting problem of 2-counter machines—that, for a generic two-way automaton $A \in \mathsf{NFA}(n,\infty,\infty)$, it is undecidable to determine whether A is synchronized (for some $s \in \mathbb{N}$). However, if $A \in \mathsf{DFA}(n,\infty,0)$ is one-way deterministic, then the same problem is decidable, because A is asynchronous unless it is $(q-1)$-synchronized, where q is the number of A's states. Since we notice that the determinization construction of single-tape automata is applicable to deterministic one-way multi-tape automata as well, we have that the problem of deciding whether a generic $A \in \mathsf{NFA}(n,\infty,0)$ is synchronized is also decidable.

On the other hand, the following problem is undecidable (for $n > 1$) [11, Th. 11]: given $A \in \mathsf{NFA}(n, \infty, 0)$, determine whether $\mathcal{L}(A) \in \mathsf{NFA}(n, s, 0)$ with s given or not (the two problems are both undecidable because of the result (9): every s-synchronous language is also 0-synchronous), that is whether A recognizes a synchronous language. Undecidability immediately extends to the two-way case (also with bounded reversals). The decidability of the corresponding problems for deterministic one-way automata is currently open.

5 Asynchronous Multi-Tape Automata

This section studies several properties of asynchronous multi-tape automata $\mathsf{NFA}(n, \infty, \mathsf{R})$ and $\mathsf{DFA}(n, \infty, \mathsf{R})$, where R denotes a generic value in $\mathbb{N} \cup \{\infty\}$, and $n \geq 2$. Asynchrony significantly increases the expressiveness of multi-tape automata, but it also significantly restricts the decidability and closure properties.

5.1 Expressiveness

This section shows that:

$$\mathsf{NFA}(n, s, \mathsf{R}) \subset \mathsf{DFA}(n, \infty, 0) \subseteq \mathsf{DFA}\left(n, \infty, \widehat{r}\right) \subset \mathsf{DFA}\left(n, \infty, \widehat{r+1}\right) , \quad (15)$$

$$\mathsf{DFA}\left(n, \infty, \widehat{r}\right) \subset \mathsf{NFA}\left(n, \infty, \widehat{r}\right) \subset \mathsf{NFA}\left(n, \infty, \widehat{r+1}\right) , \quad (16)$$

$$\mathsf{DFA}\left(n, \infty, \widehat{0}\right) = \mathsf{DFA}(n, \infty, 0) \subset \mathsf{NFA}(n, \infty, 0) = \mathsf{NFA}\left(n, \infty, \widehat{0}\right) , \quad (17)$$

and, for $r \geq 1$,

$$\mathsf{NFA}\left(n, \infty, \widehat{r}\right) \subset \mathsf{DFA}(n, \infty, 2r) \subseteq \mathsf{NFA}(n, \infty, 2r) , \quad (18)$$

$$\mathsf{NFA}(n, \infty, 0) \text{ and } \mathsf{DFA}\left(n, \infty, \widehat{r}\right) \text{ are incomparable.} \quad (19)$$

5.1.1 Asynchronous More Expressive Than Synchronous

There exist asynchronous languages that no synchronous automaton recognizes, thus:

$$\mathsf{NFA}(n, s, \mathsf{R}) = \mathsf{DFA}(n, s, \mathsf{R}) \subset \mathsf{DFA}(n, \infty, 0) . \quad (20)$$

In fact, consider the language over pairs of strings

$$L_{n,2n} = \{\langle a^n, a^{2n} \rangle \mid n \in \mathbb{N}\} .$$

It is clear that $L_{n,2n} \in \mathsf{DFA}(n, \infty, 0)$, because the automaton in Figure 1 recognizes it; notice that the automaton is indeed asynchronous, as the second head makes twice as many moves as the first one, and both scan the entire tape upon acceptance. On the contrary,

$$L_{n,2n}^{\otimes} = \left\{ \left[\begin{array}{c} a \\ a \end{array} \right]^n \left[\begin{array}{c} \square \\ a \end{array} \right]^n \mid n \in \mathbb{N} \right\}$$

is not regular, hence $L_{n,2n}$ is not accepted by any synchronous 2-tape automaton because of the results of Section 4.1.4. Since we can always extend $L_{n,2n}$ to an n-word language with all components empty but the first two, the separation of synchronized and asynchronous holds for every $n \geq 2$.

5.1.2 Expressiveness Increases With Rewinds

The power of asynchronous multi-tape automata increases with the number of rewinds:

$$\mathsf{DFA}\Big(n, \infty, \overset{\frown}{r}\Big) \subset \mathsf{DFA}\Big(n, \infty, r \overset{\frown}{+} 1\Big) \ , \tag{21}$$

$$\mathsf{NFA}\Big(n, \infty, \overset{\frown}{r}\Big) \subset \mathsf{NFA}\Big(n, \infty, r \overset{\frown}{+} 1\Big) \ . \tag{22}$$

Collectively, rewind-bounded automata are instead as expressive as rewind-unbounded ones:

$$\bigcup_{r \in \mathbb{N}} \mathsf{DFA}\Big(n, \infty, \overset{\frown}{r}\Big) = \mathsf{DFA}\Big(n, \infty, \overset{\frown}{\infty}\Big) \ , \tag{23}$$

$$\bigcup_{r \in \mathbb{N}} \mathsf{NFA}\Big(n, \infty, \overset{\frown}{r}\Big) = \mathsf{NFA}\Big(n, \infty, \overset{\frown}{\infty}\Big) \ . \tag{24}$$

These results have been proved by Chan [4, Ch. 4]. Consider first deterministic rewind-bounded automata. Given any 2^r distinct positive integers

$$0 < k_1 < k_2 < \cdots < k_{2^r} \ ,$$

consider the language

$$L_{2^r} \ = \ \{\langle a^m, a^{k_i \cdot m}\rangle \mid m \in \mathbb{N}, 1 \le i \le 2^r\} \ .$$

It is not difficult to show that $L_{2^r} \in \mathsf{DFA}\Big(2, \infty, \overset{\frown}{r}\Big)$: each language $\{\langle a^m, a^{k \cdot m}\rangle \mid m \in \mathbb{N}\}$ is recognizable without rewinds, and the r rewinds can implement a binary search among all 2^r different values of $k_1, k_2, \ldots, k_{2^r}$. The standard argument that binary search of 2^r elements requires, in the worst case, r comparisons, carries over to this setting of r-rewind automata, hence allowing us to prove that r rewinds are necessary to recognize L_{2^r}. The argument clearly extends to an arbitrary number of tapes, thus separating $\mathsf{DFA}\Big(n, \infty, \overset{\frown}{r}\Big)$ from $\mathsf{DFA}\Big(n, \infty, r \overset{\frown}{+} 1\Big)$ for all finite r's.

Let us now consider nondeterministic rewind-bounded automata. The argument for deterministic ones breaks down, as we can use nondeterministic parallelism to try out all 2^r values of k. More generally, every language over *unary* alphabet that is accepted by a reversal-bounded n-tape automaton is also accepted by some one-way n-tape nondeterministic automaton (that can be constructed effectively, see Chan [4, Th. 4.1]). However, inclusion is still strict for larger alphabets. For $m \ge 2$, consider the 2-word language

$$L_{0/1,m} \ = \ \{\langle x_1 \# x_2 \# \cdots \# x_m, x_m \# x_{m-1} \# \cdots \# x_1\rangle \mid x_1, \ldots, x_m \in \{0, 1\}^*\} \ .$$
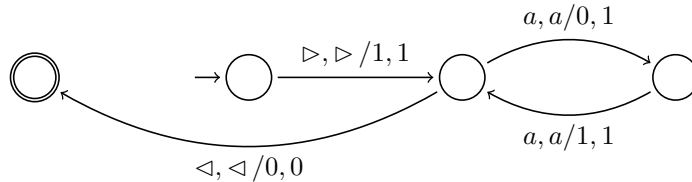

Figure 1: A 2-tape asynchronous automaton recognizing the language $L_{n,2n}$.

It is clear that $L_{0/1,m}$ is accepted by a deterministic $(m-1)$-rewind bounded automaton. Chan proves, using Yao and Rivest's technique [23], that no nondeterministic automaton with only $m-2$ rewinds can accept $L_{0/1,m}$. Since a deterministic automaton is a special case of a nondeterministic one, we have a separation of $\mathsf{NFA}(2,\infty,m-2)$ and $\mathsf{NFA}(2,\infty,m-1)$ (which, as usual, extends to a generic number of tapes).

Finally, Rosenberg first observed [18] that bounding the number of rewinds is without loss of generality for deterministic automata: if, in some computation, an automaton re-enters the same state after two rewinds, it has entered an infinite loop (and hence will not accept the input). The pigeonhole principle implies that an automaton must re-enter the same state after a number of rewinds equal to the number of its internal states. Thus, a finite bound on the number of rewinds (equal to the number of internal states) does not limit the expressiveness of deterministic automata. A very similar argument works for nondeterministic automata too (as observed by Chan): an input can be accepted only if there exists some computation where a different state is entered after each rewind; therefore, every accepting computation has bounded rewinds. These arguments prove (23) and (24).

5.1.3 Expressiveness and Reversals

It is currently unknown whether a hierarchy on to the number of reversals exists, similar to the hierarchy on the number of rewinds discussed in the previous section. In particular, it is unknown whether the inclusions $\mathsf{DFA}(n,\infty,r) \subseteq \mathsf{DFA}(n,\infty,r+1)$ and $\mathsf{NFA}(n,\infty,r) \subseteq \mathsf{NFA}(n,\infty,r+1)$ are strict for every r. The standard conjecture is that they are, but the capabilities of reversals are quite difficult to capture exactly for asynchronous automata. In particular, Chan [4, Sec. 4.3] showed that the hierarchy collapses for nondeterministic automata over unary alphabet, since these automata precisely define the set of unary encodings of Presburger relations, regardless of the number of reversals; thus, a separation in the general case requires languages over binary alphabets.

5.1.4 Nondeterminism Not Replaceable by Rewinds

Nondeterminism is not replaceable by rewinds in general. Consider the language L_{2^r} introduced in Section 5.1.2; a deterministic automaton needs r rewinds to recognize it, but it is clear that a nondeterministic automaton needs no rewinds, as it can just guess the right value of k (if it exists). This shows that $\mathsf{NFA}(n,\infty,0) \setminus \mathsf{DFA}\left(n,\infty,\widehat{r}\right)$ is not empty, for each $r \geq 0$ and $n \geq 2$.

Chan [4, Th. 4.6] proves a stronger result: there exist languages accepted by one-way nondeterministic automata that no deterministic automaton with (arbitrarily many) rewinds accepts. The language

$$
L_{\star,m} = \left\{ \left\langle \begin{array}{c} u_1 \star v_1 \# \cdots \# u_m \star v_m, \\ x_1 \star y_1 \# \cdots \# x_n \star y_n \end{array} \right\rangle \;\middle|\; \begin{array}{c} \left(\begin{array}{c} u_1, \ldots, u_m, \\ v_1, \ldots, v_m, \\ x_1, \ldots, x_n, \\ y_1, \ldots, y_n \end{array} \right) \in \{0,1\}^+, \\ \exists i,j : u_i = x_j \;\wedge\; v_i \neq y_j \end{array} \right\}
$$

is an example that belongs to the non-empty difference

$$
\mathsf{NFA}(n,\infty,0) \setminus \bigcup_{r \in \mathbb{N}} \mathsf{DFA}\left(n,\infty,\widehat{r}\right) . \tag{25}
$$

13

Rosenberg [18, Th. 9] shows a simpler language that belongs to the set difference (25). Consider the two languages

$$
\begin{aligned}
B &= \{\langle xa, \epsilon \rangle \mid x \in \{a, b\}^* \}, \\
C &= \{\langle b^m, b^m \rangle \mid m \in \mathbb{N} \},
\end{aligned}
$$

and their concatenation

$$
A = B \circ C = \quad \{\langle xab^m, b^m \rangle \mid m \in \mathbb{N}, x \in \{a, b\}^* \}.
$$

It is clear that both B and C are in $\mathsf{DFA}(2, \infty, 0)$. On the contrary, A is not in $\mathsf{DFA}\left(2, \infty, \widehat{r}\right)$ for any $r \geq 0$ because a deterministic automaton cannot "guess" where the tail of b's starts on the first tape; rewinds do not help, because the tail can be arbitrarily long, while the number of rewinds is bounded by the finite number of states of the automaton (as we observed in Section 5.1.2). For the same reason, A is in $\mathsf{NFA}(2, \infty, 0)$, where a nondeterministic choice guesses where the tail starts.

5.1.5 Rewinds Not Replaceable by Nondeterminism

Rewinds are not replaceable by nondeterminism in general, as first proved by Rosenberg [18, Th. 9]. We can easily show it using the language

$$
L_{m,m}\{\langle a^m b a^m, a^m b a^m \rangle \mid m \in \mathbb{N} \},
$$

discussed later in Section 5.2.1. $L_{m,m}$ is in $\mathsf{DFA}\left(2, \infty, \widehat{1}\right)$: a deterministic automaton can make a pass to check that the two input strings are identical and in the form $a^* b a^*$; then, it rewinds and makes another pass to check that the sequence of a's before the b on the first tape has the same length as the sequence of a's after the b on the second tape. However, $L_{m,m}$ is not in $\mathsf{NFA}(2, \infty, 0)$, because nondeterminism does not help here (Section 5.2.1 gives a more rigorous characterization of why this is the case). Hence, the difference

$$
\bigcup_{r \in \mathbb{N}} \mathsf{DFA}\left(n, \infty, \widehat{r}\right) \setminus \mathsf{NFA}(n, \infty, 0) \tag{26}
$$

is not empty.

5.1.6 Reversals Not Replaceable by Rewinds

Reversals are not replaceable by rewinds in general, not even when combined with nondeterminism. The 2-word language

$$
L_\rho = \{\langle x, rev(x) \rangle \mid x \in \{0, 1, \#\}^* \}
$$

is clearly recognizable with a 1-reversal bounded deterministic automaton. No automaton with rewinds and nondeterminism, however, can accept L_ρ. Otherwise, let M be such an automaton, and let μ be the number of its rewinds; we could then use M to recognize a language similar to $L_{0/1,m}$ for $m = \mu + 2$, but where the word on the second tape is the reversal of the one on the first; this is impossible with μ rewinds, as shown by same argument that showed that $L_{0/1,m}$ is not accepted with less than $m - 1$ rewinds. The contradiction shows that the difference $\mathsf{DFA}(n, \infty, 1) \setminus \mathsf{NFA}\left(n, \infty, \widehat{R}\right)$ is not empty.

5.1.7 Determinism vs. Nondeterminism

Whether reversals can replace nondeterminism is an open problem in general; in particular, it is unknown whether the inclusion $\mathsf{DFA}(n, \infty, \mathsf{R}) \subseteq \mathsf{NFA}(n, \infty, \mathsf{R})$ is strict or not. Some related results for single-tape multi-head automata might suggest that it is. For example, Bebják and Štefáneková [1] show some functions $f(m)$ (where m is the length of the input) such that $f(m)$-reversal bounded n-head nondeterministic automata are strictly more expressive than their deterministic counterparts. The general question for n-head automata is likely hard to settle, as it corresponds [9] to the open problem of whether deterministic logarithmic space (L) equals nondeterministic logarithmic space (NL).

5.2 Closure Properties

This section presents the closure properties of multi-tape finite automata. We start with one-way automata, and then introduce rewinds and reversals.

5.2.1 Deterministic One-Way Automata

Let us consider the closure properties of the class $\mathcal{D}_n = \mathsf{DFA}(n, \infty, 0)$, for $n \geq 2$.

Rabin and Scott observed [16, Th. 17] that $\mathcal{D}_2$ is closed under complement, with the usual construction that complements the accepting states (with a transition function that is total). The same construction carries over to show that $\mathcal{D}_n$ is closed under complement for any n.

$\mathcal{D}_n$ is not closed under projection, but we have the weaker property that if $L \in \mathcal{D}_n$ then $\exists_k L \in \mathcal{N}_{n-1}$. Rabin and Scott proved this [16, Th. 16] for $n = 2$, and other authors [5, 6, 15] generalized it. The idea of the proof is the following: given an n-tape automaton A, build an $n-1$-tape automaton B that replicates A's behavior on all tapes except the k-th (which is projected out), where it behaves nondeterministically (that is, it performs any computation A may perform on the k-th tape).

Since $\mathcal{D}_n$ is closed under complement but not under projection, it is also not closed under generalization: the projection $\exists_k L$ of a language $L \in \mathcal{D}_n$ is equivalently expressed as $\overline{\forall_k \overline{L}}$, hence if $\mathcal{D}_n$ were closed under generalization, it would also be closed under projection—a contradiction.

$\mathcal{D}_n$ is not closed under intersection. For example, Rabin and Scott [16, Th. 17] suggest the 2-word language:

$$L_{m,m} = \{\langle a^m b a^m, a^m b a^m \rangle \mid m \in \mathbb{N}\}$$
$$= L_m \cap L_{x,x} = \{\langle a^m b a^h, a^k b a^m \rangle \mid m, h, k \in \mathbb{N}\} \cap \{\langle x, x \rangle \mid x \in \{a, b\}^*\}.$$

Clearly $L_m \in \mathcal{D}_2$ and $L_{x,x} \in \mathcal{D}_2$; however, their intersection $L_{m,m}$ is not in $\mathcal{D}_2$, because its projection $\exists_1 L_{m,m} = \{a^m b^m\}$ is not regular (i.e., in $\mathcal{D}_1 = \mathcal{N}_1$), contradicting the closure under projection established above. Obviously, the proof generalizes to any $n \geq 2$. Notice that $L_{x,x}$ is synchronous, hence $\mathcal{D}_2$ is not closed under intersection even with synchronous languages.

Since $\mathcal{D}_n$ is closed under complement but not under intersection, it is also not closed under union by De Morgan's laws: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. However, if $L_1, L_2 \in \mathcal{D}_n$ then $L_1 \cup L_2 \in \mathcal{N}_n$, because a nondeterministic automaton can recognize L_1 and L_2 in parallel.

Fischer and Rosenberg showed [6, Sec. 3] that $\mathcal{D}_n$ is not closed under concatenation, Kleene closure, or reversal. The proof uses the three simple 2-word languages $E = \{\langle a, a\rangle, \langle b, b\rangle\}$, $G = \{\langle c, c\rangle\}$, $H = \{\langle a, \epsilon\rangle, \langle b, \epsilon\rangle\}$. Clearly,

$$E^*GH^* = \{\langle xcy, xc\rangle \mid x, y \in \{a, b\}^*\} \in \mathcal{D}_2 \, ;$$

because recognizing E^*GH^* amounts to comparing the words on the two tapes up to the character c; however, the concatenation

$$E^*GH^* \circ E^*GH^* = \{\langle x_1cy_1x_2cy_2, x_1cx_2c\rangle \mid x_1, x_2, y_1, y_2 \in \{a, b\}^*\}$$

is not in $\mathcal{D}_2$, intuitively because a deterministic automaton cannot "guess" when y_2 ends and x_2 begins (formally, a counting argument similar to the pumping lemma of regular languages shows that a deterministic automaton must misclassify some words). This implies that $\mathcal{D}_2$, and hence $\mathcal{D}_n$, is not closed under concatenation. A similar arguments that considers arbitrary concatenations of E^*GH^* proves that $\mathcal{D}_2$ is not closed under Kleene closure, either.

To show non-closure under reversal, take the language $H^*E^* = \{xy, y \mid x, y \in \{a, b\}^*\}$, which is accepted by a nondeterministic automaton in $\mathcal{N}_2$ that guesses when x ends in xy. A modification of the argument used for $E^*GH^* \circ E^*GH^* \notin \mathcal{D}_2$ proves that H^*E^* is not recognizable with a deterministic automaton. However, $rev(H^*E^*) = E^*H^*$ clearly is in $\mathcal{D}_2$; in all $\mathcal{D}_2$, and $\mathcal{D}_n$, is not closed under reversal.

5.2.2 Nondeterministic One-Way Automata

Let us consider the closure properties of the class $\mathcal{N}_n = \mathsf{NFA}(n, \infty, 0)$, for $n \geq 2$.

$\mathcal{N}_n$ is closed under projection, using the same construction discussed in Section 5.2.1 for deterministic automata: given an n-tape automaton A, build an $n - 1$-tape automaton B that replicates A's behavior on all tapes except the k-th (which is projected out), where it behaves nondeterministically (that is, it performs any computation A may perform on the k-th tape). Hence if $L \in \mathcal{N}_n$ then $\exists_k L \in \mathcal{N}_{n-1}$.

$\mathcal{N}_n$ is closed under union; Elgot and Mezei gave [5] the first proof of this fact (using a different formalism, ultimately corresponding to one-way multi-tape automata), which generalizes the construction for 1-tape automata: a nondeterministic automaton checks all components of a finite union in parallel through nondeterministic choice, and accepts if and only if at least one parallel computation accepts. Similarly, the classic constructions for 1-tape automata are applicable to n-tape automata to prove that $\mathcal{N}_n$ is closed under concatenation, Kleene closure, and reversal.

Like $\mathcal{D}_n$, $\mathcal{N}_n$ is not closed under intersection. The proof is the very same as in the deterministic case: L_m and $L_{x,x}$ (Section 5.2.1) are in $\mathcal{N}_2$ (even in $\mathcal{D}_2$), but their intersection $L_{m,m}$ is not in $\mathcal{N}_2$ because its projection on either component is not regular, whereas $\mathcal{N}_n$ is closed under projection.

Closure under union and non-closure under intersection imply, through De Morgan's laws, that $\mathcal{N}_n$ is not closed under complement.

Finally, $\mathcal{N}_n$ is not closed under generalization. The proof uses the following lemma proved by Monks [15, Th. 4]: for every n-word language L in $\mathcal{N}_n$ there exists an $(n + 1)$-word language L' in $\mathcal{D}_n$ such that $L = \exists_1 L'$. The proof gives an effective construction that turns an automaton accepting L into a deterministic automaton accepting L' over an extended alphabet, which can be projected out to obtain the original set L of n-words. Now, consider L_m and $L_{x,x}$ again; both are in $\mathcal{D}_2$ and, since $\mathcal{D}_2$ is

closed under complement, their complements $\overline{L_m}$ and $\overline{L_{x,x}}$ also are in $\mathcal{D}_2$, hence in $\mathcal{N}_2$ *a fortiori*. $\mathcal{N}_2$ is closed under union, thus the language

$$L_\cup \;=\; \overline{L_m} \cup \overline{L_{x,x}}$$

is also in $\mathcal{N}_2$. Therefore, the lemma we just stated ensures the existence of another language $\widetilde{L_\cup}$ in $\mathcal{D}_3$ such that $\exists_1 \widetilde{L_\cup}$ equals $L_\cup$; since $\widetilde{L_\cup} \in \mathcal{D}_3$, its complement $\overline{\widetilde{L_\cup}}$ is in $\mathcal{D}_3$ as well. Now, notice that the complement $\overline{L_\cup}$ of $L_\cup$ is equivalent to

$$\overline{L_\cup} \;=\; \overline{\overline{L_m} \cup \overline{L_{x,x}}} \;=\; L_m \cap L_{x,x} \;=\; L_{m,m}\,,$$

which we noted is not in $\mathcal{N}_2$. If $\mathcal{N}_n$ were closed under generalization,

$$\forall_1 \overline{\widetilde{L_\cup}} \;=\; \overline{\exists_1 \widetilde{L_\cup}} \;=\; \overline{L_\cup}$$

would be in $\mathcal{N}_2$, which we noted is not the case. In all, $\mathcal{N}_n$ is not closed under generalization.

5.2.3 Deterministic Rewind-Bounded Automata

Let us consider the closure properties of the class $\mathcal{RD}_n = \mathsf{DFA}\Big(n, \infty, \widehat{\infty}\Big)$, for $n \geq 2$.

$\mathcal{RD}_n$ is closed under complement, since each pass of a rewind automaton on the input tapes corresponds to a computation of an automaton in $\mathcal{D}_n$, which is closed under complement. Hence, it is sufficient to adapt the usual construction for complementation in the following way: after every pass, accept if the original automaton rejects, reject if the original automaton accepts, and rewind if the original automaton rewinds.

$\mathcal{RD}_n$ is closed under union: given an automaton $A_1 \in \mathcal{RD}_n$ accepting L_1 and another automaton $A_2 \in \mathcal{RD}_n$ accepting L_2, an automaton A accepting $L_1 \cup L_2$ operates as follows: simulate A_1; if A_1 accepts A also accepts; otherwise, A rewinds, simulates A_2, and accepts iff A_2 does. Notice that termination is not a problem because the final part of Section 5.1.2 discussed why we can assume that the number of rewinds is always bounded.

Closure under union and complement imply closure under intersection for $\mathcal{RD}_n$ through De Morgan's laws. Alternatively, we can produce a direct construction similar to the one for union, where the simulation of two automata A_1, A_2 accepts iff both simulations accept.

Not only is $\mathcal{RD}_n$ closed under all Boolean operations; Rosenberg proves [18, Th. 8] that every language in $\mathcal{RD}_n$ is expressible as a finite Boolean combination of languages in $\mathcal{D}_n$ (essentially, each of finitely many rewinds represents a one-way deterministic computation over the input), hence $\mathcal{RD}_n$ is the *Boolean closure* of $\mathcal{D}_n$.

$\mathcal{RD}_n$ is not closed under concatenation, because we have already observed in Section 5.1.4 that the languages B and C are in $\mathcal{D}_2$ (hence in $\mathcal{RD}_2$ *a fortiori*), but their concatenation $A = B \circ C$ is not in $\mathcal{RD}_2$ because accepting it requires nondeterminism. (As usual, the result immediately carries over to $\mathcal{RD}_n$).

Since, however, $rev(A) = \{\langle b^m a x, b^m \rangle \mid m \in \mathbb{N}, x \in \{a,b\}^*\}$ clearly is in $\mathcal{RD}_2$, we conclude that $\mathcal{RD}_n$ is not closed under reversal either.

$\mathcal{RD}_n$ is not closed under Kleene closure; Rosenberg's proof [18, Th. 8] works as follows. Consider the language

$$D \;=\; \{\langle b^m c, b^m \rangle \mid m \in \mathbb{N}\}\;;$$

clearly $D \in \mathcal{RD}_2$ (actually, even $D \in \mathcal{D}_2$). Then, the same argument showing $B \circ C$ not in $\mathcal{RD}_2$ proves that $B \circ D \notin \mathcal{RD}_2$. Assume by contradiction that $\mathcal{RD}_2$ is closed under Kleene closure; then, since $\mathcal{RD}_2$ is closed under union and intersection, the language

$$A' = (B \cup D)^* \cap \left(B \circ (\{a, b\}^*)^n \circ \{\langle c, \epsilon \rangle\}\right) \tag{27}$$

also is in $\mathcal{RD}_2$. This is a contradiction, because A' equals $B \circ D$—as the B in the right-most argument of $\cap$ in (27) forces B to appear first, and the closing $\langle c, \epsilon \rangle$ forces D to appear second and last—but we know that $B \circ D \notin \mathcal{RD}_2$.

We can prove that $\mathcal{RD}_n$ is not closed under projection using the language $L_{m,m} = L_m \cap L_{x,x}$ defined in Section 5.2.1. We know that both L_m and $L_{x,x}$ are in $\mathcal{D}_2$, hence their intersection $L_{m,m}$ is in $\mathcal{RD}_2$ (it is easy to build an automaton in $\mathcal{RD}_2$ that accepts $L_{m,m}$ with one rewind). The projection $\exists_1 L_{m,m} = \{a^m, b^m\}$ is, however, not in $\mathcal{RD}_1$ because it is not regular and $\mathcal{RD}_1 = \mathcal{N}_1$.

Finally, $\mathcal{RD}_n$ is not closed under generalization, because it is closed under complement but not under projection.

5.2.4 Nondeterministic Rewind-Bounded Automata

Let us consider the closure properties of the class $\mathcal{RN}_n = \mathsf{NFA}\left(n, \infty, \widehat{\infty}\right)$, for $n \geq 2$.

Using rewinds, the same constructions that show $\mathcal{RD}_n$ closed under intersection and union are applicable to show $\mathcal{RN}_n$ is closed under intersection and union: use one rewind to execute the two computations for L_1 and L_2 in parallel; accept if both accept, when recognizing $L_1 \cap L_2$; accept if at least one accepts, when recognizing $L_1 \cup L_2$.

$\mathcal{RN}_n$ is not closed under projection, with the same proof we have used in several other cases: $L_{m,m}$ is in $\mathcal{RN}_2$, but $\exists_1 L_{m,m} = \{a^m b^m\}$ is not in $\mathcal{RN}_1$ because it is not regular and $\mathcal{RN}_1 = \mathcal{N}_1$.

$\mathcal{RN}_n$ is not closed under complement either. The proof is a modification of Chan [4, Th. 4.6]: consider the language

$$L_{\neg\star,m} = \left\{ \left\langle \begin{array}{c} u_1 \star v_1 \# \cdots \# u_m \star v_m, \\ x_1 \star y_1 \# \cdots \# x_n \star y_n \end{array} \right\rangle \; \middle| \; \begin{array}{c} \left(\begin{array}{c} u_1, \ldots, u_m, \\ v_1, \ldots, v_m, \\ x_1, \ldots, x_n, \\ y_1, \ldots, y_n \end{array}\right) \in \{0,1\}^+, \\ \forall i, j : u_i = x_j \; \Rightarrow \; v_i \neq y_j \end{array} \right\}.$$

$L_{\neg\star,m}$ essentially is the complement of $L_{\star,m}$, but we can show that $L_{\neg\star,m} \notin \mathcal{RN}_2$ whereas $L_{\star,m} \in \mathcal{RN}_2$. Intuitively, rewinds do not help in recognizing $L_{\star,m}$ or $L_{\neg\star,m}$, as m and n can be arbitrarily large, whereas a finite-state automaton can only "remember" a finite number of position to jump to after any rewind; nontdeterminism works to recognize $L_{\star,m}$ but is still insufficient for $L_{\neg\star,m}$ where "universal" nondeterminism is needed. Formally, assume by contradiction that $L_{\neg\star,m} \in \mathcal{RN}_2$ for some automaton A with r rewinds. Then, modify A into A' so that A' works on inputs of the form

$$\langle x_1 \# x_2 \# \cdots \# x_{r+2}, y_1 \# y_2 \# \cdots \# y_{r+2} \rangle$$

as A would work on inputs of the form

$$\langle a \star x_1 \# a^2 \star x_2 \# \cdots \# a^{r+2} \star x_{r+2}, a^{r+2} \star y_1 \# a^{r+1} \star y_2 \# \cdots \# a \star y_{r+2} \rangle,$$

by having enough states to count a's up to $r + 2$. Therefore A' recognizes $L_{0/1,m}$ with r rewinds, contradicting the fact that $L_{0/1,m}$ is not in $\mathsf{NFA}(2, \infty, r)$. Since m is

generic, the contradiction shows that $L_{\neg\star,m} \notin \mathcal{RN}_2$, then $\mathcal{RN}_n$ is not closed under complement.

$\mathcal{RN}_n$ is closed under concatenation, Kleene closure, and reversal, through the usual constructions that exploit nondeterminism mentioned in Section 5.2.2.

Finally, whether $\mathcal{RN}_n$ is closed under generalization is an open problem.

5.2.5 Reversal-Bounded Automata

The proof that establishes the non-closure under projection of automata with rewinds (by reduction to the 1-tape case) entails the non-closure under projection of automata with generic reversals in $\mathsf{NFA}(n, \infty, \mathtt{R})$ and $\mathsf{NFA}(n, \infty, \mathtt{R})$, for any $n \geq 2$ and $\mathtt{R} > 0$. Since non-closure under projection of a class of automata is tantamount to undecidability of emptiness for automata of that class, we do not proceed further with the investigation of the closure properties of two-way automata (which have not been studied in the literature either).

5.3 Decidability

This section discusses the decidability of various problems for multi-tape automata.

5.3.1 Deterministic One-Way Automata

Let us consider a generic member A of the class $\mathcal{D}_n = \mathsf{DFA}(n, \infty, 0)$, for $n \geq 2$.

Emptiness is decidable, as a corollary of the fact that the projection of $A \in \mathcal{D}_n$ is in $\mathcal{N}_{n-1}$, which is closed under projection. Thus, $\mathcal{L}(A)$ is empty if and only if $\exists_{t_1} \exists_{t_2} \cdots \exists_{t_{n-1}} \mathcal{L}(A)$ is empty, where $t_1, \ldots, t_{n-1}$ is any subset of $\{1, \ldots, n\}$ with $n-1$ elements; the latter problem is decidable because emptiness of regular languages is decidable. Notice that the projection automaton is effectively constructible from A. Another proof of the same decidability results uses an analogue of the pumping lemma for multi-tape automata: Rosenberg [17] shows that, given a partition $Q_1 \cup \cdots \cup Q_n = Q$ of A's set Q of states, $\mathcal{L}(A) \neq \emptyset$ if and only if A accepts some input $\langle x_1, \ldots, x_n \rangle$ such that $|x_i| \leq |Q_i|$ for all $1 \leq i \leq n$. Since $\mathcal{D}_n$ is closed under complement, universality is also decidable for $A \in \mathcal{D}_n$.

The property of projection also shows that finiteness is decidable for $A \in \mathcal{D}_n$: $\mathcal{L}(A)$ is finite if and only if $\exists_{t_1} \exists_{t_2} \cdots \exists_{t_{n-1}} \mathcal{L}(A)$ is finite for *every* subset $t_1, \ldots, t_{n-1}$ of $\{1, \ldots, n\}$ with $n-1$ elements; the latter problem is decidable because finiteness of regular languages is decidable.

The problem of whether equivalence is decidable for $A \in \mathcal{D}_n$ has been open for several years; Harju and Karhumäki [8] have finally shown it to be decidable. Their proof is quite technical, and it is basically a corollary of the related problem of "multiplicity equivalence" for nondeterministic one-way multi-tape automata. Two automata in $\mathcal{N}_n$ are *multiplicity equivalent* if they accept the same n-words exactly the same number of times (that is with the same number of distinct computations). Using some group-theoretic techniques, Harju and Karhumäki show that the multiplicity equivalence problem is decidable for automata in $\mathcal{N}_n$, hence also for automata in $\mathcal{D}_n$. The multiplicity equivalence problem reduces, however, to (ordinary) equivalence for deterministic automata in $\mathcal{D}_n$, which have at most one computation on each input.

Rabin and Scott [16, Th. 18] have shown that disjointness is undecidable for automata in $\mathcal{D}_2$; as usual, the proof immediately generalizes to $\mathcal{D}_n$ with $n \geq 2$. The

proof is a simple reduction from Post's correspondence problem: given a finite set of 2-words over Σ (with $|\Sigma| \geq 2$)

$$\{\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \ldots, \langle x_m, y_m \rangle\} \, ,$$

determine if there exists a sequence $i_1, i_2, \ldots, i_k$ of indices from $1, \ldots, m$ (possibly with repetitions) such that

$$x_{i_1} x_{i_2} \cdots x_{i_k} \;=\; y_{i_1} y_{i_2} \cdots y_{i_k} \, .$$

The reduction is as follows: consider the (finite) 2-word languages

$$
\begin{aligned}
X &= \{\langle x_1, 1 \rangle, \ldots, \langle x_m, m \rangle\} \, , \\
Y &= \{\langle y_1, 1 \rangle, \ldots, \langle y_m, m \rangle\} \, .
\end{aligned}
$$

Obviously, X^* and Y^* are both in $\mathcal{D}_2$, but the given instance of Post's correspondence problem has a solution if and only if $X^* \cap Y^* \neq \emptyset$. Since Post's correspondence problem is undecidable, disjointness for $\mathcal{D}_2$ (and $\mathcal{D}_n$) is also undecidable.

The undecidability of the inclusion problem for $\mathcal{D}_n$ follows from the fact that $\mathcal{D}_n$ is closed under complement but disjointness is undecidable for it; in fact, $A \subseteq B = \emptyset$ if and only if $A \subseteq \overline{B}$.

5.3.2 Nondeterministic One-Way Automata

Let us consider a generic member A of the class $\mathcal{N}_n = \mathsf{NFA}(n, \infty, 0)$, for $n \geq 2$.

The proof of decidability of emptiness and finiteness for $\mathcal{D}_n$ does not depend on the restriction on deterministic automata, and in fact establishes that the same problems are decidable for $\mathcal{N}_n$ as well.

Conversely, undecidability results for $\mathcal{D}_n$ immediately extend to its superclass $\mathcal{N}_n$; thus disjointness and inclusion are undecidable for $\mathcal{N}_n$ as well.

Universality is undecidable for $\mathcal{N}_n$; Fischer and Rosenberg [6, Th. 7] give another proof using reduction from Post's correspondence problem (introduced in Section 5.3.1: $X^* \cap Y^* = \emptyset$ if and only if $\overline{X^* \cup Y^*} = \emptyset$ if and only if $\overline{X^*} \cup \overline{Y^*} = (\Sigma^*)^2$, which is the universality problem for a language in $\mathcal{N}_2$ (remember that $\mathcal{N}_n$ is closed under union).

Now, notice that there exist an automaton A_U in $\mathcal{D}_n$ (hence also in $\mathcal{N}_n$) that accepts the universe language $(\Sigma^*)^n$. Since deciding the equivalence $\mathcal{L}(A) = \mathcal{L}(A_U)$ is equivalent to deciding the universality problem for a generic $A \in \mathcal{N}_n$, it follows that equivalence is also undecidable for $\mathcal{N}_n$.

Finally, deciding whether a generic language $L \in \mathcal{N}_n$ is deterministically recognizable (i.e., $L \in \mathcal{D}_n$) is also undecidable, with a more complex reduction from Post's correspondence problem [6, Th. 9].

5.3.3 Deterministic Rewind-Bounded Automata

All decision problems considered in this paper are undecidable for the class $\mathcal{RD}_n = \mathsf{DFA}\!\left(n, \infty, \widehat{\infty}\right)$, for $n \geq 2$. This is a consequence of the non-decidability of some problems for the class $\mathcal{D}_n$, of which $\mathcal{RD}_n$ is a strict superset, combined with the closure of $\mathcal{RD}_n$ under all Boolean operations.

Consider, for example, the emptiness problem for $\mathcal{RD}_n$; if it were decidable, then the disjointness problem for generic $A, B \in \mathcal{D}_n$ would be decidable by reducing it to

testing the emptiness of $A \cap B \in \mathcal{RD}_n$. But Section 5.3.1 showed that disjointness is undecidable for $\mathcal{D}_n$. Since $\mathcal{RD}_n$ is closed under complement, universality is also undecidable for $\mathcal{RD}_n$.

Emptiness is reducible to finiteness as follows. Assume, *a contrario* that finiteness is decidable for $\mathcal{RD}_n$; then, we can decide whether $\mathcal{L}(A) = \emptyset$ for a generic $A \in \mathcal{RD}_n$. First, determine if A accepts a finite or infinite language; if it accepts an infinite language, we conclude that $\mathcal{L}(A)$ is not empty; thus, let $\mathcal{L}(A)$ be finite. Modify A into A' by adding a loop on every accepting state of A; the loop reads some $\sigma \in \Sigma$ and moves the head right on every tape. It should be clear that, for every n-word $\langle x_1, \ldots, x_n \rangle$ accepted by A, A' accepts all (infinitely many) words in

$$\{\langle x_1 \sigma^m, x_2 \sigma^m, \ldots, x_n \sigma^m \rangle \mid m \in \mathbb{N}\} \ .$$

We have that A' accepts a finite language if and only if $\mathcal{L}(A)$ is empty, but emptiness is undecidable for a generic $A \in \mathcal{RD}_n$.

Disjointness is undecidable for $\mathcal{RD}_n$ because the universe language $(\Sigma^*)^n$ is in $\mathcal{RD}_n$, and deciding disjointness of $\mathcal{L}(A)$ and $(\Sigma^*)^n$ is equivalent to deciding emptiness of $\mathcal{L}(A)$. Similarly, $\mathcal{L}(A) \subseteq \emptyset$ if and only if $\mathcal{L}(A) = \emptyset$, hence inclusion is also undecidable for $\mathcal{RD}_n$. Finally, equivalence is a generalization of emptiness, thus it is undecidable for $\mathcal{RD}_n$.

5.3.4 Other Two-Way Automata

The undecidability of all decision problems for $\mathcal{RD}_n$ propagates up the hierarchy: all decision problems are undecidable for $\mathcal{RN}_n$, as well as for generic two-way automata.

6 Summary of Properties

Figure 2 summarizes the known inclusion between classes of multi-tape automata. Solid lines denote strict inclusions and dotted lines denote weak inclusions; namely, it is unknown whether nondeterminism increases the expressive power of asynchronous multi-tape automata with unbounded reversals.

Table 1 summarizes the known closure properties of synchronized multi-tape automata (**syn**), deterministic asynchronous one-way (**da1**), nondeterministic asynchronous one-way (**na1**), deterministic asynchronous with reversals (**dar**), and non deterministic asynchronous with reversals (**nar**), with respect to complement $\neg$, intersection $\cap$, union $\cup$, projection $\exists$, generalization $\forall$, concatenation $\circ$, Kleene star $*$, and reversal rev. "Y" means closure, "N" means non-closure, "n" denotes closure within the next class, and "?" means open problem.

Table 2 shows which problems are decidable ("Y") and which undecidable ("N") for the same classes of multi-tape automata.

	syn	da1	na1	dar	nar
¬	Y	Y	N	Y	N
∩	Y	N	N	Y	Y
∪	Y	N	Y	Y	Y
∃	Y	n	Y	N	N
∀	Y	N	N	N	?
∘	Y	N	Y	N	Y
*	Y	N	Y	N	Y
rev	Y	N	Y	N	Y

Table 1: Closure properties of multi-tape automata.
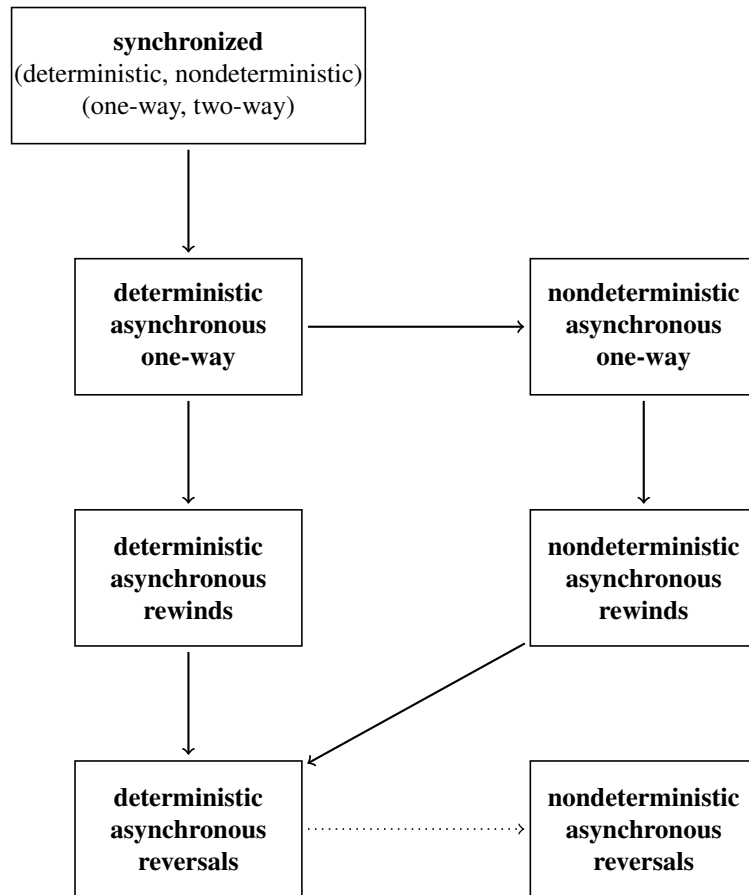
Figure 2: Relative expressiveness of multi-tape automata classes.

	syn	da1	na1	dar	nar
Emptiness	Y	Y	Y	N	N
Universality	Y	Y	N	N	N
Finiteness	Y	Y	Y	N	N
Disjointness	Y	N	N	N	N
Inclusion	Y	N	N	N	N
Equivalence	Y	Y	N	N	N

Table 2: Decidability properties of multi-tape automata.

7 Intersection of Asynchronous Automata

This section describes an algorithm for the intersection of multi-tape nondeterministic asynchronous one-way finite-state automata. Since these are not closed under intersection, the algorithm may not terminate (or, equivalently, it may define an infinite-state automaton as result). We use a slightly different definition of multi-tape automaton, which is easily seen equivalent to Definition 1 in the one-way case.

Definition 3. An *n-tape finite-state automaton* A is a tuple $\langle \Sigma, T, Q, \tau, \delta, Q_0, F \rangle$ where: Σ is the input alphabet, with $\lhd \notin \Sigma$; $T = \{t_1, \ldots, t_n\}$ is the set of tapes; Q is the finite set of states; $\tau : Q \to T$ assigns a tape to each state; $\delta : Q \times \Sigma_\lhd \to \wp(Q)$ is the (nondeterministic) transition function; $Q_0 \subseteq Q$ are the initial states; $F \subseteq Q$ are the accepting (final) states.

Whenever convenient we will represent the transition function δ as a relation, that is the set of triples (q, σ, q') such that $q' \in \delta(q, \sigma)$.

Consider two asynchronous automata $A = \langle \Sigma, Q^A, \delta^A, Q_0^A, F^A, T^A, \tau^A \rangle$ and $B = \langle \Sigma, Q^B, \delta^B, Q_0^B, F^B, T^B, \tau^B \rangle$, such that A has m tapes $T^A = \{t_1^A, \ldots, t_m^A\}$ and B has n tapes $T^B = \{t_1^B, \ldots, t_n^B\}$. We now describe an algorithm that computes the intersection $C = A \cap B$ of A and B, where $C = \langle \Sigma, Q, \delta, Q_0, F, T, \tau \rangle$; C's tapes T are the union of T^A and T^B. To describe the algorithm, we introduce repeated operations as separate routines. All components of the algorithm have access to the definitions of A and B and to a global stack s where new states of the composition are pushed (when created) and popped (when processed).

Routine *async_next* takes a t-tape automaton D (i.e., A or B) and one of its states q, and returns a set of tuples $\langle q', h_1, \ldots, h_t \rangle$ of all next states reachable from q either directly or by accumulating delayed transitions $h_i \in (\delta^D)^*$ in tape t_i, for $1 \leq i \leq t$. We call *delayed states* such tuples of states with delayed transitions. The search for states reachable from q stops at the first occurrences of states associated with a certain tape. Figure 3 shows the pseudo-code for *async_next*.

Consider now a pair of delayed states $\langle p, h_1, \ldots, h_m \rangle$ and $\langle q, k_1, \ldots, k_n \rangle$, respectively for automata A and B. The two delayed states can be composed only if the delays on the synchronized tapes are pairwise *consistent*, that is the sequence of input symbols of one is a prefix (proper or not) of the other's; $cons(h_i, k_i)$ denotes that the sequences h_i, k_i of delayed transitions are consistent. Routine *new_states* (in Figure 4) takes two sets P, Q of delayed states and returns all consistent states obtained by composing them. *new_states* also pushes onto the stack s all composite states that have not already been added to the composition. For convenience, *new_state* also embeds the tape t of each new composite state within the state itself.

It is often convenient to add arbitrary prefixes to the delays of delayed states generated by *new_states*. To this end, routine *compose_transition* (in Figure 5) takes two sets P, Q of delayed states and an $(m + n)$-tuple of delays, and calls *new_states* on the modified states obtained by orderly adding the delays to the states in P and Q. It also adds all transitions to the newly generated states to the transition function δ of the composite C.

We are ready to show the main routine *intersect* which builds C from A and B. Since the intersection may have infinite states, *intersect* takes as arguments a bound on the maximum number of states and on the maximum delay (measured in number of transitions) accumulated in the states.

1 *async_next* (D, q): *SET* $[\langle q', h_1, \ldots, h_t \rangle]$
2   −− q is always reachable from itself
3   **Result** := $\{\langle q, \epsilon, \ldots, \epsilon \rangle\}$
4   −− for every tape other than q's
5   **for each** $t_i \in \{t_1^D, \ldots, t_t^D\} \setminus \tau^D(q)$ **do**
6   P := all shortest paths p from q to some $\overline{q}$ such that:
7             $\tau^D(\overline{q}) = t_i$ and no state $\widetilde{q}$ with $\tau^D(\widetilde{q}) = t_i$ appears in p before $\overline{q}$
8   −− each element in P is a sequence of transitions
9   **for each** $e_1 \cdots e_m \in P$ **do**
10             $h_1, \ldots, h_t := \epsilon$
11   −− each transition is a triple (source, input, target)
12   **for each** $(q_1, \sigma, q_2) \in e_1 \cdots e_m$ **do**
13   −− add the transition to the sequence corresponding
14   −− to its source's tape
15                 $h_{\tau^D(q_1)} := h_{\tau^D(q_1)} + (q_1, \sigma, q_2)$
16   −− $q_2(e_m)$ is the target state of the last transition e_m
17   **Result** := **Result** $\cup \langle q_2(e_m), h_1, \ldots, h_t \rangle$

Figure 3: Routine *async_next*.

1 *new_states* (P: *SET*$[\langle p, h_1, \ldots, h_m \rangle]$, Q: *SET*$[\langle q, k_1, \ldots, k_n \rangle]$): S
2     $S := \emptyset$
3   **for each** $\langle p, h_1, \ldots, h_m \rangle \in P$, $\langle q, k_1, \ldots, k_n \rangle \in Q$ **do**
4   −− if delays on synchronized tapes are consistent
5   **if** $\forall i \in T^A \cap T^B : cons(h_i, k_i)$ **then**
6   **for each** $t \in T$ **do** $S := S \cup \{\langle p, q, t, h_1, \ldots, h_m, k_1, \ldots, k_n \rangle\}$ **end**
7   −− Here Q denotes C's set of states, not the input argument
8   **for each** $r \in S$ **do if** $r \notin Q$ **then** *s.push*(r) **end**

Figure 4: Routine *new_states*.

1 *compose_transition* (P: *SET*$[(p, h_1, \ldots, h_m)]$, Q: *SET*$[(q, k_1, \ldots, k_n)]$,
2   d: $(h_1, \ldots, h_m, k_1, \ldots, k_n), \sigma, r$)
3     $J_A := \{(p, h_1\, h_1', \ldots, h_m\, h_m') \mid (p, h_1', \ldots, h_m') \in P\}$
4     $J_B := \{(q, k_1\, k_1', \ldots, k_n\, k_n') \mid (q, k_1', \ldots, k_n') \in Q\}$
5     $S := new\_states\,(J_A, J_B)$
6   **for each** $r' \in S$ **do** $\delta := \delta \cup \{r, \sigma, r'\}$ **end**

Figure 5: Routine *compose_transition*.

```
 1  intersect  (max_states ,  max_delay)
 2    Q := ∅ ; s := ∅
 3    −− A's  initially  reachable   states
 4    J_A := ⋃_{i∈I_A}  async_next (A, i)
 5    −− B's  initially  reachable   states
 6    J_B := ⋃_{i∈I_B}  async_next (B, i)
 7    S := new_states (J_A, J_B)
 8    −− mark these  states  as   initial
 9    I := S
10    until  s = ∅ or |Q| ≥ max_states loop
11      r := (q_a, q_b, t, h_1, …, h_m, k_1, …, k_n) = s.pop
12      if  ∀d ∈ {h_1, …, k_n} : |d| ≤ max_delay then
13        Q := Q ∪ {r}
14      else  continue
15      Q := Q ∪ {r}
16      if  t ∈ T^A ∩ T^B then
17        −− event on shared  tape
18        if  h_t = (u_a, σ, u'_a)‾h_t‾ and k_t = (u_b, σ, u'_b)‾k_t‾ then
19          −− delayed  transition  on both  A and B
20          P := async_next (A, q_a)
21          Q := async_next (B, q_b)
22          d := (h_1, …, ‾h_t‾, …, h_m, k_1, …, ‾k_t‾, …, k_n)
23          compose_transition  (P, Q, d, σ, r)
24        elseif  h_t = (u_a, σ, u'_a)‾h_t‾ and k_t = ε then
25          −− delayed  transition  on A
26          P := async_next (A, q_a)
27          −− normal  transition  on B
28          Q := { async_next (B, q'_b) | (q_b, σ_b, q'_b) ∈ δ^B ∧ σ = σ_b ∧ τ^B(q_b) = t}
29          d := (h_1, …, ‾h_t‾, …, h_m, k_1, …, k_n)
30          compose_transition  (P, Q, d, σ, r)
31        elseif  h_t = ε and k_t = (u_b, σ, u'_b)‾k_t‾ then
32          −− delayed  transition  on B
33          Q := async_next (B, q_b)
34          −− normal  transition  on A
35          P := { async_next (A, q'_a) | (q_a, σ_a, q'_a) ∈ δ^A ∧ σ = σ_a ∧ τ^A(q_a) = t}
36          d := (h_1, …, h_m, k_1, …, ‾k_t‾, …, k_n)
37          compose_transition  (P, Q, d, σ, r)
38        elseif  h_t = k_t = ε then
39          for  each σ ∈ Σ do
40            −− normal  transition  on both  A and B
41            P := { async_next (A, q'_a) | (q_a, σ_a, q'_a) ∈ δ^A ∧ σ_a = σ ∧ τ^A(q_a) = t}
42            Q := { async_next (B, q'_b) | (q_b, σ_b, q'_b) ∈ δ^B ∧ σ_b = σ ∧ τ^B(q_b) = t}
43            d := (h_1, …, h_m, k_1, …, …, k_n)
44            compose_transition  (P, Q, d, σ, r)
45      elseif  t ∈ T^A \ T^B then
46        −− event on A's  non−shared  tape
47        if  h_t = (u_a, σ, u'_a)‾h_t‾ then
48          −− delayed  transition  on A, B stays
```

26

```
49            P := async_next (A, q_a)
50            Q := {(q_b, ϵ, . . . , ϵ)}
51            d := (h_1, . . . , h_t, . . . , h_m, k_1, . . . , k_n)
52            compose_transition  (P, Q, d, σ, r)
53        elseif  h_t = ϵ then
54            −− normal transition  on A, B stays
55              Q := {(q_b, ϵ, . . . , ϵ)}
56              for  each σ ∈ Σ do
57                P := { async_next (A, q'_a) | (q_a, σ_a, q'_a) ∈ δ^A ∧ σ_a = σ ∧ τ^A(q_a) = t}
58                d := (h_1, . . . , h_m, k_1, . . . , k_n)
59                 compose_transition  (P, Q, d, σ, r)
60      elseif  t ∈ T^B ∖ T^A then
61          −− event on B's non−shared tape
62        if  k_t = (u_b, σ, u'_b) k_t then
63            −− delayed  transition  on B, A stays
64            P := {(q_a, ϵ, . . . , ϵ)}
65            Q := async_next (B, q_b)
66            d := (h_1, . . . , h_m, k_1, . . . , k_t, . . . , k_n)
67            compose_transition  (P, Q, d, σ, r)
68        elseif  k_t = ϵ then
69            −− normal transition  on B, A stays
70              P := {(q_a, ϵ, . . . , ϵ)}
71              for  each σ ∈ Σ do
72                Q := { async_next (B, q'_b) | (q_b, σ_b, q'_b) ∈ δ^B ∧ σ_b = σ ∧ τ^B(q_b) = t}
73                d := (h_1, . . . , h_m, k_1, . . . , k_n)
74                 compose_transition  (P, Q, d, σ, r)
```

References

[1] Andrej Bebják and Ivana Štefáneková. Nondeterminism is essential for reversal-bounded two-way multihead finite automata. *Kybernetika*, 24(1):65–71, 1988.

[2] Achim Blumensath and Erich Grädel. Automatic structures. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26–29, 2000*, pages 51–62. IEEE Computer Society, 2000.

[3] Jean-Marc Champarnaud, Franck Guingne, André Kempe, and Florent Nicart. Algorithms for the join and auto-intersection of multi-tape weighted finite-state machines. *International Journal of Foundations of Computer Science*, 19(2):453–476, 2008.

[4] Tat-hung Chan. *Reversal-Bounded Computation*. PhD thesis, Cornell University, 1980.

[5] C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965.

[6] Patrick C. Fischer and Arnold L. Rosenberg. Multitape one-way nonwriting automata. *Journal of Computer and System Sciences*, 2(1):88–101, 1968.

[7] Gösta Grahne, Matti Nykänen, and Esko Ukkonen. Reasoning about strings in databases. *Journal of Computer and System Sciences*, 59(1):116–162, 1999.

[8] Tero Harju and Juhani Karhumäki. The equivalence problem of multitape finite automata. *Theoretical Computer Science*, 78(2):347–355, January 1991.

[9] Markus Holzer, Martin Kutrib, and Andreas Malcher. Multi-head finite automata: Characterizations, concepts and open problems. In Turlough Neary, Damien Woods, Anthony Karel Seda, and Niall Murphy, editors, *Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6–7th December 2008*, volume 1 of *EPTCS*, pages 93–107, 2009.

[10] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 3rd edition, 2006.

[11] Oscar H. Ibarra and Nicholas Q. Tran. On synchronized multitape and multi-head automata. In Markus Holzer, Martin Kutrib, and Giovanni Pighizzini, editors, *Descriptional Complexity of Formal Systems – 13th International Workshop, DCFS 2011*, volume 6808 of *Lecture Notes in Computer Science*, pages 184–197, Berlin, Heidelberg, 2011. Springer Berlin / Heidelberg.

[12] Andre Kempe, Franck Guingne, and Florent Nicart. Algorithms for weighted multi-tape automata. Technical Report 031, XRCE, 2004.

[13] Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In Daniel Leivant, editor, *Logical and Computational Complexity. Selected Papers. Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, Indiana, USA, 13–16 October 1994*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer, 1995.

[14] Dexter C. Kozen. *Automata and Computability*. Springer, 1997.

[15] Maria Monks. Closure properties of predicates recognized by deterministic and non-deterministic asynchronous automata. `http://arxiv.org/abs/1010.3039`, 2010.

[16] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 39(2):114–125, 1959.

[17] Arnold L. Rosenberg. *Nonwriting extensions of finite automata*. PhD thesis, Harvard University, 1965. Technical report BL-39.

[18] Arnold L. Rosenberg. Multitape finite automata with rewind instructions. *Journal of Computer and System Sciences*, 1(3):299–315, 1967.

[19] Sasha Rubin. Automata presenting structures: A survey of the finite string case. *The Bulletin of Symbolic Logic*, 14(2):169–209, 2008.

[20] Michael Shapiro. Deterministic and non-deterministic asynchronous automatic structures. *International Journal of Algebra and Computation*, 2(3):297–305, 1992.

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

[22] Moshe Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30(5):261–264, March 1989.

[23] Andrew Chi-Chih Yao and Ronald L. Rivest. $k+1$ heads are better than k. *Journal of the ACM*, 25(2):337–340, 1978.