From Finite-Valued Nondeterministic Transducers to Deterministic Two-Tape Automata

Anonymized Authors

Abstract—The question whether P equals NP revolves around the discrepancy between active production and mere verification by Turing machines. In this paper, we examine the analogous problem for finite transducers and automata. Every nondeterministic finite transducer defines a binary relation associating each input word with all output words that the transducer can successfully produce on the given input. Finite-valued transducers are those for which there is an finite upper bound on the number of output words that the relation associates with every input word. We characterize finite-valued, functional, and unambiguous nondeterministic transducers whose relations can be verified by a deterministic two-tape automaton, show how to construct such an automaton if one exists, and prove the undecidability of the criterion.

Index Terms—Finite-Valued Transducers, Two-Tape Automata, Production versus Verification by Finite Machines, Undecidable Verifiability Criterion.

Formal Verification

All major results presented in this paper were also formally proved in Isabelle/HOL [13], a theorem prover using higher-order logic, ensuring their correctness beyond any reasonable doubt. The formalizations are available at https://github.com/lics21-automata/automata, including an explanation of how to interpet them and details on how they connect to the individual definitions and theorems in the present paper.

I. Introduction

Automata. One of the simplest computation models is a finite automaton reading any given input word from left to right while deterministically updating its state according to a transition function that depends on the current state and symbol that is currently read; once the input word has been read completely, it is either accepted or rejected depending on the state in which the automaton ends up. We refer to such an automaton as a deterministic finite automaton (DFA). For a nondeterministic finite automaton (NFA), in contrast, the transition function may permit an arbitrary number of transitions from the current state instead of exactly one; the input word is then accepted if it is accepted for any of the possible choices of transitions. The set of words accepted by an automaton A is its recognized language $L(A)$. It is well-known that deterministic and nondeterministic finite automata both recognize the same class of *regular* languages [14]; that is, nondeterminism does not add any computational power to the finite automaton model.

Two-Tape Automata. One way to generalize finite automata is the multi-tape model. Here, an automaton has multiple tapes, each with one head that can read the word written on the tape from left to right. There are several different but equivalent ways to model multi-tape automata, most notably the Rabin-Scott model and the Turing machine model. In the Rabin-Scott model, only one of the heads is reading a symbol and advancing to the next one in each step; the current state determines on which tape this happens. The computation is either accepted or rejected once all heads have reached the end of the tape. In the Turing machine model, in contrast, the multi-tape automaton reads all symbols at the current head positions simultaneously and can then move forward any subset of the heads in each step. In this paper, we only examine deterministic finite automata with two tapes (2t-DFAs).

Transducers. *Transducers* were introduced by Elgot and Mezei [8] as a generalization of automata. A transducer reads the given input word symbol by symbol, just like an automaton, but additionally produces a separate output word while doing so. During each transition reading one input symbol, the transducer produces a possibly empty sequence of output symbols. The final output word is obtained by concatenating the outputs of all transitions. After transducing the entire input word into an output word, the transducer decides whether the computation is accepting or not depending on the current state. In this paper, we are mainly interested in nondeterministic finite transducers (NFTs). For further context, we will also consider deterministic finite transducers (DFTs) and an extension of NFTs that allows for λ-transitions (λ-NFTs). Allowing λ-transitions means that the transducer can produce output not only when reading an input symbol but also on the empty word λ, that is, at any time without reading any part of the input.

Functionality and Finite-Valuedness. A deterministic transducer associates any given input word with the output word produced on it if the corresponding computation is accepting. A nondeterministic transducer can have multiple accepting computations on an input word, associating it with all corresponding output words. Denoting the input and output alphabet by Σ and Γ, respectively, a transducer's language is a relation between Σ^* and Γ^*. For two-tape automata, we analogously consider the tapes as containing an input over Σ and an output over Γ, respectively. Thus, the language of a two-tape automaton is a relation $L \subseteq \Sigma^* \times \Gamma^*$ as well.

Given such a relation, we say that an input $a \in \Sigma^*$ is *associated* with an output $u \in \Gamma^*$ if $(a, u) \in L$. The relation L is *functional* if any input is associated with at most one output. Note that such an L describes a function that may be

only partial. Given a $k \in \mathbb{N}$, we say that L is *k-valued* if it associates at most k outputs with every input. If there is a $k \in \mathbb{N}$ such that L is k-valued, we call L finite-valued. We call two-tape automata and transducers functional or finite-valued if their languages are. We denote the restriction of machine model M to functional or finite-valued ones by f-M and fv-M, respectively. Moreover, we denote by $\mathcal{L}(M)$ the family of languages recognizable by machines of model M.

See Figure 1 for an overview of the hierachy of languages that are of interest to us. The vertical inclusions in it are trivial since functionality implies finite-valuedness. The strictness follows for instance from the relations $\{(0,0),(0,00),\ldots,(0,0^k)\}$—which is k-valued but not functional for any $k > 1$—and the not even finite-valued $\{\,0,(0,0^i) \mid i \in \mathbb{N}\,\}$ since they are recognizable by 2t-DFAs as well as NFTs.

It is worth pointing out that determinism implies functionality for transducers but not for two-tape automata.

Motivation. The main motivation underlying all computation models is to study their expressive power, that is, the set of languages recognized by machines in each model. For instance, DFTs are strictly less expressive than functional NFTs. A characterization of the languages that can be recognized in one given model but not the other is also of interest.

For example, a classical result by Choffrut shows that satisfying the so-called *twinning property (condition de jumelage)* is both a necessary and sufficient condition for a functional NFT to be determinizable [5]. Choffrut also introduced the notion of functions *with bounded variation (à variation bornée)*—called *uniformly bounded* by others [16]—which characterizes the functions recognized by determinizable NFTs. Later, this criterion of bounded variation was proved to be decidable in polynomial time [21].

Our goal is to develop an analogous property that characterizes functional NFTs having an equivalent 2t-DFA, to examine the decidability or undecidability of this criterion, and provide an algorithm that transforms an NFTs into an equivalent 2t-DFA whenever possible.

Inclusion Hierarchy. To provide the context for our main endeavor, we prove all inclusions in Figure 1, going from left to right. The inclusion $\mathcal{L}(DFT) \subseteq \mathcal{L}(2t\text{-}DFA)$ has been formally proved by Aho et al. [1, Thms. 3.1 and 3.2]. The intuition is simple: Given a DFT, we can construct an equivalent 2t-DFA that simply simulates the DFT on the first tape and checks whether the output of the DFT matches the content of the second tape. It is easy to construct an example, such as the following, showing that the inclusion is strict.

Example 1. *We present the concrete language that can be recognized by an f-2t-DFA but not by a DFT.*

Consider the relation that maps any binary string w to $0^{|w|}$ if the last symbol of w is 0 and to $1^{|w|}$ otherwise. It is easy to construct a functional 2t-DFA that verifies this relation as follows.

The automaton advances its two heads synchronously, always comparing the current output symbol with the previous one. If there is a discrepancy at any point, the automaton rejects; otherwise, the output word is guaranteed to be uniform.

$$
\begin{array}{ccc}
& [11] & \\
\mathcal{L}(2t\text{-}DFA) & \subsetneq & \mathcal{L}(\lambda\text{-}NFT) \\
\cup \wr & [\text{Thm. 23}] & \cup \wr \qquad [20] \\
\mathcal{L}'(fv\text{-}2t\text{-}DFA) & \subsetneq & \mathcal{L}(fv\text{-}NFT) \\
[5] \quad \cup \wr & [\text{Thm. 23}] & \cup \wr \qquad [17] \\
\mathcal{L}(DFT) \subsetneq \mathcal{L}'(f\text{-}2t\text{-}DFA) & \subsetneq & \mathcal{L}(f\text{-}NFT)
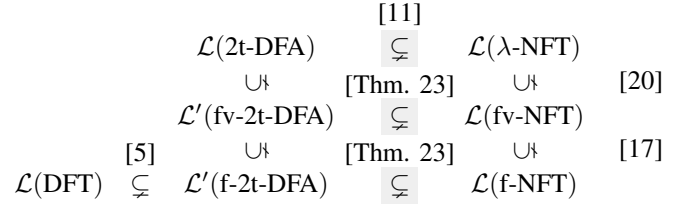\end{array}
$$

Fig. 1: The strict hierarchy of languages recognized by relevant machine models. The inclusions and their strictness are mostly trivial; they are briefly discussed in the introduction.

For each inclusion, there is the following decision problem: Given an element of the superset, is it contained in the subset? The shaded inclusions are exactly those associated with an undecidable problem. The references point to the proofs of decidability and undecidability, respectively. The decidability for the two vertical inclusions is an immediate consequence of the same result for the two right ones.

Let $\mathcal{L}'(fv\text{-}2t\text{-}DFA) = \{L \smallsetminus (\{\lambda\} \times \Gamma^*) \mid L \in \mathcal{L}\}$ denote the set of fv-2t-DFA-recognizable relations with all pairs (λ, x) associating the empty input λ to any output removed; $\mathcal{L}'(f\text{-}2t\text{-}DFA)$ is defined analogously. Writing $\mathcal{L}'$ instead of $\mathcal{L}$ is necessary in the two cases only for the inclusions to their right. The reasons for this are technical and discussed in the introduction.

If the two heads reach the end of the input and output word at the same time and the last two symbols match, then the automaton accepts; otherwise, it rejects.

A DFT cannot compute this relation, however, because it cannot output any symbol until it knows the last symbol of the word on the first tape, and by that time, the transducer cannot recall the length of the input word, which equals the number of copies of the last symbol that the transducer needs to produce.

To prove the inclusion $\mathcal{L}(2t\text{-}DFA) \subseteq \mathcal{L}(\lambda\text{-}NFT)$, we can construct a transducer that guesses the output and then verifies it by simulating the automaton. Here, we need to consider transducers with λ-transitions—i.e., transducers that can produce output without reading any input symbol—for the inclusion to hold because a general two-tape automaton can advance its output head arbitrarily many times while its input head is standing still.

In the two inclusions $\mathcal{L}(f\text{-}2t\text{-}DFA) \subseteq \mathcal{L}(f\text{-}NFT)$ and $\mathcal{L}(fv\text{-}2t\text{-}DFA) \subseteq \mathcal{L}(fv\text{-}NFT)$ intuitively the transducer just guesses the output and then verifies it by simulating the automaton. This works because the output head of a finite-valued (or functional) automaton can only advance by at most n positions, where n denotes the number of the automaton's states, while the input head stands still. Otherwise, one could pump in a non-empty output subword on which the automaton loops without changing the input word. However, one needs to exclude computations on the empty input word λ since the transducer cannot produce (thus guess) any output without reading an input symbol.

Question of Decidability. We now turn our attention to the corresponding decidability questions. Whether a machine is

finite-valued is decidable even in the case of nondeterministic transducers [20]. We can also decide whether the given fv-NFT is functional or not [17]. It follows from these two results that we can decide the finite-valuedness and functionality for two-tape automata as well.

Via a reduction from the Post correspondence problem, Fischer and Rosenberg proved that, for any $k \geq 2$, it is undecidable whether a k-tape NFA has any equivalent k-tape DFA [11]. In particular, given a 2t-NFA, it is undecidable whether there is an equivalent 2t-DFA. This can be reformulated with λ-NFTs, as they are equivalent to 2t-NFAs. Their proof relies on relations that are not finite-valued, however. In this paper, we extend Fischer and Rosenberg's undecidability result [11] to the finite-valued, functional, and even unambiguous case. The undecidability also implies the strictness of the corresponding inclusions.

For the remaining decision problem, we can construct from the given f-2t-DFA an equivalent NFT, as long as we consider automata with accepting relations with a non-empty input word, as described above. Finally, we can check whether the described NFT, which is equivalent to the given fv-2t-DFA, has an equivalent DFT via the decidable twinning property [6].

Alternative Terminology. We mention some alternative terminology that will not be relevant in presentation of our paper but can be found in the literature. Deterministic transducers are sometimes called *sequential* transducers. This is because their transductions—which are unique for every given input word—can be processed sequentially, symbol by symbol.

There is also the notion of *subsequential* transducers. These are sequential transducers with one additional feature: After reading the entire input word and having reached a final state, they can append to the output one more word that may depend on the final state.

The difference between sequentiality and subsequentiality is an overall rather insubstantial technical detail. It is easy to see that sequential and subsequential become equivalent under the assumption of a unique symbol marking the end of every input word.

A function is called sequential or subsequential if it is recognized as the relation of a sequential or subsequential transducer, respectively. Finally, a relation (and in particular a function) is called *rational* if it is recognized by a finite nondeterministic transducer.

We can now restate our motivating result by Choffrut in his own terminology: Rational functions with bounded variation are exactly the subsequential functions.

It is also possible to extend the notion of subsequentiality to nondeterministic transducers. However, the additional option to add a suffix to the output depending on the final state makes even less of a difference if nondeterministic transitions are allowed. This is because the transducer can always guess that the current symbol is the last with an additional transition option including the final suffix for every situation. Whenever the guess turns out to be wrong, the corresponding computation path can be aborted in the following step. This works on all input words except for the empty one: On the empty input word λ a transducer cannot perform even a single transition and is thus unable to produce any output at all without subsequentiality. However, even with subsequentiality an NFT can only associate finitely many output words with the input λ; the difference is thus negligible and we may just ignore it by excluding all pairs $\{ (\lambda, u) \mid u \in \Gamma^* \}$ in the relation from consideration.

Further Model Extensions. We remark that both multi-tape automata and transducers can be extended to the *two-way* model that allows the heads to move not only forward but also back, from right to left [3], [4]. Filiot et al. [10] showed that it is decidable whether a two-way DFT can be transformed into an equivalent one-way DFT and provide a characterization of the two-way DFTs for which this is possible. It is known that two-way DFTs compute exactly those relations that are expressible as monadic second-order logic string transductions [9]. Rabin and Scott [14] proved that two-way automata are as powerful as one-way automata for a single tape but recognize strictly more languages with multiple tapes.

Yao and Rivest have shown that increasing the number of heads on a single tape yields a strict hierarchy of languages [22]; see also the survey by Holzer et al. [12]. A more recent result by Raszyk et al. [15] proves that DFTs with multiple heads are strictly more expressive than f-NFTs. Asking whether multi-head two-way DFAs can simulate two-head one-way NFAs is equivalent to the famous question whether $L = NL$ [18].

In the present paper, we exclusively consider the one-way model with one head per tape.

Our Contributions. We compare the computational power of 2t-DFAs and fv-NFTs. To this end, we introduce the notion of transducers having *bounded trailing* and show that this property characterizes fv-NFTs whose relations are computable by a 2t-DFA. In other words: Any given fv-NFT (and thus in particular any functional NFT) has an equivalent 2t-DFA if and only if the fv-NFT has bounded trailing.

This result is analogous to the already mentioned classical result by Choffrut [6]. He introduced the notion of a transducer satisfying the *twinning property* and proved that this property characterizes functional NFTs having an equivalent DFT.

We show how our notion of bounded trailing can be seen as a relaxation of another property due to Choffrut [6], namely functions *with bounded variation*.

Furthermore, we prove the question whether a given transducer has bounded trailing to be undecidable. This stands in contrast to the twinning property, which was proven to be a polynomial-time decidable criterion by Weber and Klemm [21].

Finally, we provide a concrete construction that is guaranteed to transform any given fv-NFT into an equivalent 2t-DFA whenever possible, that is, if the fv-NFT has bounded trailing, despite this criterion being undecidable.

We have formally proved all major results of this paper (Lemma 10, Theorem 14, Claim 21, Claim 22, and Theorem 15 for functional NFTs) as well as Examples 13 and 17 in the proof assistant Isabelle [2] to ensure their correctness. We also provide pen-and-paper proofs for all of our results.

II. Preliminaries

We formally describe the automaton model and transducer model considered in this paper and then introduce some useful terminology.

There are many equivalent ways of extending the one-tape automaton model to multiple tapes. We base our definition on the Turing machine model.

Definition 2. *A* two-tape deterministic finite automaton *(2t-DFA), is a septuple* $A = (Q, \Sigma, \Gamma, {}_\sqcup, \delta, q_0, F)$ *consisting of*

- *a finite, nonempty set of states* Q,
- *a finite, nonempty input alphabet* Σ,
- *a finite, nonempty output alphabet* Γ,
- *a* blank symbol ${}_\sqcup \notin \Sigma \cup \Gamma$,
- *an initial state* $q_0 \in Q$,
- *a set of accepting states* $F \subseteq Q$, *and*
- *a transition function* δ *satisfying the following properties:*
    - $\delta\colon Q \times (\Sigma \cup \{{}_\sqcup\}) \times (\Gamma \cup \{{}_\sqcup\})$
      $\to Q \times \{\text{Stay}, \text{Advance}\} \times \{\text{Stay}, \text{Advance}\}$,
      $(q, \sigma, \gamma) \mapsto (q', m_1, m_2)$,
    - $\sigma = {}_\sqcup \implies m_1 = \text{Stay}$
      (i.e., the first head stops at the end delimiter ${}_\sqcup$*)*,
    - $\gamma = {}_\sqcup \implies m_2 = \text{Stay}$ *(i.e., the second head stops at the end delimiter* ${}_\sqcup$*), and*
    - $m_1 = \text{Advance} \vee m_2 = \text{Advance}$ *(i.e., at least one head advances in every step).*

The computation of a 2t-DFA proceeds as follows. There are two tapes that we call the *input* and *output* tape; the former contains a word $a \in \Sigma^*$, the latter a word $u \in \Gamma^*$. Both words are delimited by the blank symbol ${}_\sqcup$ at the end. The two tapes have one reading head each, which we refer to as the *input* and *output* head. The input and output head are initially positioned on the first symbol of a and u, respectively. Depending on the current state of the automaton and what symbols the two heads are reading, either the input or output head or both advance by one symbol in each step. As soon as a head reaches the blank symbol delimiting a word, it cannot move any further. The computation ends when both heads have reached the end of the input. A word pair (a, u) is in the language accepted by the automaton if and only if the computation on this pair of words ends in an accepting state. A *configuration* consists of the positions of the two heads and the current state. We write $q \xrightarrow{a}{}_{u} q'$ if the automaton can start in a state q and end up in a state q' by reading a word a with the input head and u with the output head.

We now formally define a nondeterministic finite transducer.

Definition 3. *A* nondeterministic finite transducer *(NFT) is a sextuple* $T = (Q, \Sigma, \Gamma, \delta, q_0, F)$, *where* Q, Σ, Γ, q_0, *and* F *are defined as for a 2t-DFA in Definition 2, but the* transition function δ *now is a function that maps each pair* $(q, \sigma) \in Q \times \Sigma$ *of a state and input symbol to a* finite subset *of* $Q \times \Gamma^*$, *describing the nondeterministic transition options.*

The computation of an NFT proceeds like the computation of an NFA except that the NFT produces in each step a possibly empty sequence of output symbols. The outputs from the single steps are concatenated to obtain the final output of the complete computation. Using the visualization of a two-tape machine, we can describe the computation of an NFT as reading the word on the input tape symbol-wise while writing to the initially empty output tape, appending each step's output. We write $q \xrightarrow{a}{}_{u} q'$ if the transducer transitions from state q to state q' with the input head reading $a \in \Sigma^*$ and the output head producing $u \in \Gamma^*$.

The computation ends once the entire input word has been read, that is, when the input head has reached the blank symbol. If the transducer is in an accepting state at this moment, then the word pair (a, u) on the two tapes is in the relation $L(T)$ computed by T. If the transition function does not offer any option for a step and thus forcibly ends the computation before the blank symbol on the input tape is reached, then this computation does not contribute to the relation $L(T)$.

A binary relation $R \subseteq \Sigma^* \times \Gamma^*$ is *finite-valued* if a constant bounds the number of outputs $u \in \Gamma^*$ per input $a \in \Sigma^*$, that is, if

$$\exists k \in \mathbb{N}\colon \forall a \in \Sigma^*\colon \quad |\{\, u \in \Gamma^* \mid (a, u) \in R \,\}| \leq k.$$

Definition 4. *An NFT* T *is a* finite-valued *nondeterministic finite transducer (fv-NFT) if* $L(T)$ *is a finite-valued relation. If* $L(T)$ *is even functional, then* T *is a* functional *nondeterministic finite transducer (f-NFT).*

We will use the following two notions related to NFTs many times in our proofs.

Definition 5 (Shortcut guarantee g)**.** *Let an NFT* $T = (Q, \Sigma, \Gamma, \delta, q_0, F)$ *be given. Let* $Q' \subseteq Q$ *denote the set of* all *useful* states, *that is, states from which an accepting state can be reached on some input. For every* $q \in Q'$, *let*

$$g_q = \min\{\, |x| \mid \exists u \in \Gamma^*, f \in F\colon q \xrightarrow{x}{}_{u} f \,\}$$

denote the length of a shortest word x *that leads from* q *into an accepting state. We call* $g(T) = \max_{q \in Q'} g_q$ *the* shortcut guarantee *of* T.

Definition 6 (Output speed s)**.** *Let an NFT* $T = (Q, \Sigma, \Gamma, \delta, q_0, F)$ *be given. We call*

$$s(T) = \max\{\, |\gamma| \mid \exists q, q' \in Q, \sigma \in \Sigma\colon (q', \gamma) \in \delta(q, \sigma) \,\}$$

the output speed *of* T. *Note that* $s(T)$ *is well-defined since* δ *only maps to finite subsets of* $Q \times \Gamma^*$.

III. Bounded Trailing

In this section, we introduce our core notion of *bounded trailing*, which we will prove to be an undecidable characterization of fv-NFTs having an equivalent fv-2t-DFA in the following three sections. We begin by recapitulating the notion of *bounded variation* introduced by Choffrut [6], which characterizes f-NFTs having an equivalent DFT.

For any two words v_1, v_2, we denote by $\text{lcp}(v_1, v_2)$ their longest common prefix and define the distance

$$d(v_1, v_2) = |v_1| + |v_2| - 2 \cdot |\text{lcp}(v_1, v_2)|.$$

Observe that prefixing any word u to both v_1 and v_2 does not change their distance; we have $d(uv_1, uv_2) = d(v_1, v_2)$.

Definition 7 (Bounded variation). *A relation $R \subseteq \Sigma^* \times \Gamma^*$ has* bounded variation *if*

$$\forall k \in \mathbb{N} \colon \exists t \in \mathbb{N} \colon \quad \forall (a_1, u_1), (a_2, u_2) \in R \colon$$
$$d(a_1, a_2) \leq k \quad \Longrightarrow \quad d(u_1, u_2) \leq t.$$

If a_1 and a_2 have a common prefix a, we can split them into $a_1 = ab_1$ and $a_2 = ab_2$, respectively. We have $d(b_1, b_2) \leq |b_1| + |b_2|$ and, if a is the longest common prefix of a_1 and a_2, indeed $d(b_1, b_2) = |b_1| + |b_2|$. Applying $d(a_1, a_2) = d(ab_1, ab_2) = d(b_1, b_2)$, we thus obtain the following equivalent definition of bounded variation.

Lemma 8 (Bounded variation). *A relation $R \subseteq \Sigma^* \times \Gamma^*$ has* bounded variation *if*

$$\forall k \in \mathbb{N} \colon \exists t \in \mathbb{N} \colon \quad \forall a, b_1, b_2 \in \Sigma^*, \ u_1, u_2 \in \Gamma^* \colon$$
$$(ab_1, u_1) \in R \ \wedge \ (ab_2, u_2) \in R \ \wedge \ |b_1| + |b_2| \leq k$$
$$\Longrightarrow \quad d(u_1, u_2) \leq t.$$

We can now instantiate the characterization of bounded variation described in Lemma 8 for the case of a relation recognized by a transducer. Specifically, we translate an input-output pair lying in the relation into the existence of a concrete computation by an NFT. Appropriately splitting u_1 and u_2 into uv_1w_1 and uv_2w_2, respectively, with a (not necessarily longest) common prefix u, and applying $d(u_1, u_2) = d(uv_1w_1, uv_2w_2) = d(v_1w_1, v_2w_2)$, this results in the following definition.

Definition 9 (Bounded variation for transducers). *An NFT $T = (Q, \Sigma, \Gamma, q_0, F, \delta)$ has* bounded variation *if*

$$\forall k \in \mathbb{N} \colon \exists t \in \mathbb{N} \colon \quad \forall q_1, q_2 \in Q, \ f_1, f_2 \in F,$$
$$a, b_1, b_2 \in \Sigma^*, \ u, v_1, v_2, w_1, w_2 \in \Gamma^* \colon$$
$$q_0 \xrightarrow[uv_1]{a} q_1 \xrightarrow[w_1]{b_1} f_1 \ \wedge \ q_0 \xrightarrow[uv_2]{a} q_2 \xrightarrow[w_2]{b_2} f_2 \ \wedge \ |b_1| + |b_2| \leq k$$
$$\Longrightarrow \quad d(v_1w_1, v_2w_2) \leq t.$$

We will now simplify Definition 9. It follows from the definition of d that

$$d(v_1, v_2) - \big||w_1| - |w_2|\big| \leq d(v_1w_1, v_2w_2)$$
$$\leq d(v_1, v_2) + |w_1| + |w_2|.$$

(These bounds are tight; consider $v_1 = 000$, $v_2 = 0$, $w_1 = 0$ and either $w_2 = 000$ or $w_2 = 100$.) Using the bounded output speed s of T (Definition 6), we obtain

$$\big||w_1| - |w_2|\big| \leq |w_1| + |w_2| \leq s(|b_1| + |b_2|) \leq sk.$$

Hence, the difference between $d(v_1, v_2)$ and $d(v_1w_1, v_2w_2)$ can be bounded by a term depending only on T and k and can thus be hidden within the choice of t. We can therefore simplify $d(v_1w_1, v_2w_2)$ to $d(v_1, v_2)$.

Using the shortcut guarantee g of T (Definition 5) for b_1 and b_2, we can now assume without loss of generality that $|b_1| + |b_2| \leq 2g$ because replacing b_1 and b_2 can only change the respective output words w_1 and w_2, which are

no longer referenced anywhere else. This finally allows us to drop the universally quantified variable $k \in \mathbb{N}$ altogether. We thus obtain the following equivalent definition of bounded variation.

Lemma 10. *An NFT $T = (Q, \Sigma, \Gamma, q_0, F, \delta)$ has bounded variation if*

$$\exists t \in \mathbb{N} \colon \quad \forall q_1, q_2 \in Q, \ f_1, f_2 \in F,$$
$$a, b_1, b_2 \in \Sigma^*, \ u, v_1, v_2, w_1, w_2 \in \Gamma^* \colon$$
$$q_0 \xrightarrow[uv_1]{a} q_1 \xrightarrow[w_1]{b_1} f_1 \ \wedge \ q_0 \xrightarrow[uv_2]{a} q_2 \xrightarrow[w_2]{b_2} f_2$$
$$\Longrightarrow \quad d(v_1, v_2) \leq t.$$

Let us call a computation *useful* if it can be extended to an accepting computation. Intuitively, an NFT has bounded variation if the outputs of any two useful computations on the same input prefix a only differ on suffixes of bounded length.

Our notion of bounded trailing relaxes the notion of bounded variation by only looking at certain pairs of useful computations on the same input prefix a, namely computation pairs such that the output uv of the first computation contains the output u of the second computation as a prefix and such that the second computation can be extended to an accepting one that *trails* the first computation, in the sense of catching up with its output, by first producing the missing suffix v.

Since we have only restricted the computation pairs for which we impose the condition on the output, we have indeed relaxed the notion of bounded variation. In particular, bounded variation implies bounded trailing.

We now formally define our notion of bounded trailing. Starting from the condition in Lemma 10, we set $v_2 = \lambda$, rename v_1 to v, and require that w_2 starts with the prefix v, which we then split off in the notation. Finally, we use $d(\lambda, v) = |v|$.

Definition 11 (Bounded trailing). *An NFT $T = (Q, \Sigma, \Gamma, q_0, F, \delta)$ has* bounded trailing *if*

$$\exists t \in \mathbb{N} \colon \quad \forall q_1, q_2 \in Q, \ f_1, f_2 \in F,$$
$$a, b_1, b_2 \in \Sigma^*, \ u, v, w_1, w_2 \in \Gamma^* \colon$$
$$q_0 \xrightarrow[uv]{a} q_1 \xrightarrow[w_1]{b_1} f_1 \ \wedge \ q_0 \xrightarrow[u]{a} q_2 \xrightarrow[vw_2]{b_2} f_2 \ \Longrightarrow \ |v| \leq t.$$

Example 13 describes a concrete transducer with bounded trailing.

The following three sections prove bounded trailing to be a necessary and sufficient condition for an fv-NFT to have an equivalent 2t-DFA and that bounded trailing is undecidable. In contrast, bounded variation is decidable since it is equivalent to the so-called *twinning property*, which is decidable in polynomial time [21]. We also point out that bounded trailing does not characterize having an equivalent 2t-DFA for NFTs that are not finite-valued. Constructing a counterexample is straightforward; we provide one in the following example.

Example 12. *We describe a not finite-valued transducer that has an equivalent 2t-DFA but nevertheless unbounded trailing.*
Consider the relation

$$\{(0^n, 0^m) \mid n \in \mathbb{N} \wedge m \in \mathbb{N} \wedge 0 \leq m \leq 2n\},$$

which is clearly not finite-valued. It is easy for a 2t-DFA to recognize this relation by repeatedly advancing first the output head twice and then the input head once. Any input-output pair is in the language if and only if the end of the output is reached before the end of the input and all read symbols before the end markers are 0. It is also simple to build an equivalent NFT with unbounded trailing as follows. Whenever reading one input symbol, the transducer writes a 0 to the output tape either zero, one, or two times.

This transducer does indeed have unbounded trailing since for every even $t \in \mathbb{N}$, it has the two computations

$$q_0 \xrightarrow[\lambda 0^t]{0^{t/2}} q_\triangle \xrightarrow[\lambda]{0^{t/2}} f_\triangle \quad \text{and} \quad q_0 \xrightarrow[\lambda]{0^{t/2}} q_\triangledown \xrightarrow[0^t \lambda]{0^{t/2}} f_\triangledown,$$

where λ is the empty word and

$$a = b_1 = b_2 = 0^{t/2}, u = w_1 = w_2 = \lambda, \text{ and } v = 0^t$$

in Definition 11.

IV. BOUNDED TRAILING IS SUFFICIENT

In this section, we show that any NFT that has bounded trailing can be transformed into an equivalent 2t-DFA. Let T be an NFT with a trailing bound $t \in \mathbb{N}$. We construct an equivalent 2t-DFA A that simulates all nondeterministic computations of transducer T that are compatible with the output seen so far. Automaton A uses its states to maintain a subword z of the output word with the following property. For the currently read prefix p of the input word, there is a prefix x of the output word such that for any accepting computation of T that starts in the initial state q_0 of T, reads p, reaches a state q of T, and produces an output w consistent with the given output tape, we can write w as xy for a prefix y of z. In other words: Whatever prefix p of its input word automaton A has read at any point, it has always stored a z such that for some x every accepting computation of T on p consistent with A's output word has the form $q_0 \xrightarrow[xy]{p} q$ for some prefix y of z.

Automaton A stores in its current state a representation of each such computation of T, namely the pair $(q, |y|)$. We denote the set of these pairs by P. We show that storing P is feasible with a finite set of states by maintaining a subword z of length at most $r = s + t$, where s and t are T's output speed and trailing bound, respectively. Initially, z is empty and the set P of pairs (q, n) stored in A's state contains only a single pair, $(q_0, 0)$, where q_0 is T's initial state. This reflects the fact that the only computation of T on the empty prefix of the input tape keeps T in its initial state and produces no output. If $L(T) = \emptyset$, then A immediately transitions into a rejecting sink state.

Automaton A now proceeds as follows. As long as the length of the subsequence z stays below r and the output tape has not been fully read, the next symbol on the output tape is read and appended to z. Moreover, A removes from the set P all representations of computations that cannot be extended to an accepting computation with the extended subsequence z of the output tape. If the set P becomes empty, then automaton A transitions into a rejecting sink state, because there is no accepting computation of T consistent with the given input

and output tape. Otherwise, A determines the minimum m such that $(q, m) \in P$ for some q and drops the first m output symbols from the subsequence z. This corresponds to cutting off from z a prefix of length m and appending it to x. Note that this is sound because none of the stored computations end before outputting the new x. This way, A maintains the invariant that there is some state q of T such that $(q, 0) \in P$.

Once the length of the subsequence z becomes r or the output tape has been fully read, automaton A reads in the next symbol from the input tape and then simulates every single step that is nondeterministically possible for every single stored computation and updates the set P accordingly to a new P'. The mentioned invariant guarantees that $(q, 0) \in P$ for some state q of T. The fact that T has bounded trailing with a trailing bound t implies that $n \leq t$ holds for every pair $(q, n) \in P$. Hence, performing one further nondeterministically possible step continuing T's computation represented by $(q, n) \in P$ yields a pair $(q', n') \in P'$ that satisfies $n' \leq t + s$ since s is the longest output that T can produce while reading a single symbol. This is just within the length limit $r = s + t$ that we set for z. As before, automaton A rejects if the set P' becomes empty, because this means there is no accepting computation of T consistent with the given input and output tape. Otherwise, automaton A performs on P' the normalization described in the previous paragraph to obtain a new set P that maintains the invariant that there is some q' for which $(q', 0) \in P$.

Finally, if both the input and output tape have been fully read, automaton A accepts if and only if there is some $q \in F$ with $(q, |z|) \in P$, that is, an accepting state q of T that some computation of T arrives at after producing an output that matches xz until the very end, meaning that the output is equal to the content of the output tape.

Example 13. *We now provide a concrete example of the general construction described above; that is, we use a given NFT with bounded trailing to construct an equivalent 2t-DFA.*

We consider the NFT T with the set of states $Q = \{q_0, q_1, q_2, q_3\}$, initial state $q_0 \in Q$, the set of accepting states $F = \{q_0\}$, and the transition function δ given in Figure 2a. Let us define the language

$$L_0 = \{(aa, ababa), (aa, ababab), (ab, ababaa)\}.$$

One can check that the language computed by T is the Kleene closure of L_0 (i.e., $L(T) = L_0^*$) and that the trailing of T is bounded by $t = 1$.

The computation of automaton A on the input-output pair $(aa, ababab)$ is summarized in Figure 2b. We now describe this computation in detail. Because $L(T) \neq \emptyset$, the initial state of automaton A is storing $z = \lambda$, where λ denotes the empty word, and $P = \{(q_0, 0)\}$. The output speed of T—that is, the length of a longest output that T can produce while reading one input symbol—is $s = 4$. Hence, the maximum length of the subsequence z maintained in A's state is $r = s + t = 5$.

Because the output tape consists of six symbols, the first $r = 5$ of them are read and appended to z. The nonempty output words in $L(T)$ all start with the same five symbols, and the only pair in P, namely $(q_0, 0)$, can be extended to an accepting

computation consistent with next five output symbols. These symbols are therefore successively output and appended to z, while $(q_0, 0)$ is being kept in P. After this, A's state is storing $z = ababa$ and $P = \{(q_0, 0)\}$.

Now that the length of z is $5 = r$, the first symbol a from the input tape is read. All nondeterministic steps from

$$\delta(q_0, a) = \{(q_1, aba), (q_2, abab), (q_3, abab)\}$$

are consistent with z, which leads to

$$P' = \{(q_1, 3), (q_2, 4), (q_3, 4)\}.$$

The minimum m for which $(q, m) \in P'$ for some q is $m = 3$. After performing the normalization of P' with this m, the state of automaton A is storing $z = ba$ and $P = \{(q_1, 0), (q_2, 1), (q_3, 1)\}$.

Since the length of z is now $2 < r$, automaton A reads the next output symbol b and appends it to z, which thus becomes $z = bab$. Then A removes the pair $(q_3, 1)$ from the set P because this pair can no longer be extended to an accepting computation consistent with the extended subsequence $z = bab$ since the only possible transition from q_3 produces the output aa, which is inconsistent with the suffix ab of $z = bab$. The remaining two pairs $(q_1, 0)$ and $(q_2, 1)$ are still consistent with the extended $z = bab$. As $(q_1, 0)$ stays in P, no normalization need be performed.

Now that the end of the output tape has been reached, A reads in the next symbol a from the input tape despite $|z| = 3 < r$. Performing one step that reads a on every pair in P yields $P' = \{(q_0, 3)\}$. Note that $(q_0, 2) \notin P'$ because each transition from q_0 starts with a b, the last symbol of z. After one more normalization, A's state is storing $z = \lambda$ and $P = \{(q_0, 0)\}$.

Finally, A has reached the end of both the input and output tape, thus it checks whether $(q, |z|) \in P$ for some $q \in F$. Because $(q_0, 0) \in P$, $q_0 \in F$, and $|z| = 0$, automaton A accepts the input-output pair $(aa, ababab) \in L(T)$.

We conclude this section by formally stating its main result.

Theorem 14. *Any NFT with bounded trailing has an equivalent 2t-DFA.*
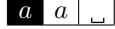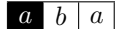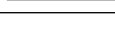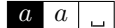
V. BOUNDED TRAILING IS NECESSARY
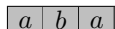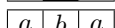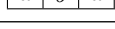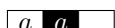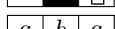
In this section, we prove the reverse of Theorem 14 for the case of finite-valuedness.

Theorem 15. *Any fv-NFT with an equivalent 2t-DFA has bounded trailing.*

Proof. Let an fv-NFT T and a 2t-DFA A with $L(T) = L(A)$ be given. (We may assume that T and A use a common input alphabet Σ and a common output alphab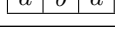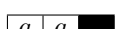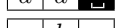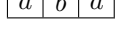et Γ.) Let k be an arbitrary integer such that T is k-valued. We assume that T has unbounded trailing and derive from this a contradiction to the k-valuedness of T, thus proving the theorem. We may assume without loss of generality that A moves exactly one head in each step because any step moving both heads at once can be simulated by two steps moving only one head at a time using one additional intermediate state.

	a	b
q_0	$\{(q_1, aba), (q_2, abab), (q_3, abab)\}$	$\emptyset$
q_1	$\{(q_0, ba), (q_0, bab)\}$	$\emptyset$
q_2	$\{(q_0, ab)\}$	$\emptyset$
q_3	$\emptyset$	$\{(q_0, aa)\}$

(a) The transition function δ of the given NFT T.

(b) The computation of the 2t-DFA A on the input $(aa, ababab)$. The positions of the heads are marked in black, the subsequence z of the output tape is shaded gray, and x consists of the white cells before z.

Fig. 2: Illustrations pertaining to Example 13.

Denote the state sets of transducer T and automaton A by Q_T, Q_A, respectively. Let g and s be the transducer's shortcut guarantee and output speed, respectively. Finally, we define a *homestretch length* $h = (g+1) \cdot (|Q_A| + 1)$ and a *trail length minimum* $t = h + t_1 + \cdots + t_k$, where t_i is recursively defined by

$$t_i = 2s + s(1 + i(t_0 + t_1 + \cdots + t_{i-1})) \cdot (1 + |Q_T| \cdot |Q_A|^i)$$

with the base case $t_0 = h + (g+1)s$. The reason for choosing exactly these values for h and t will become clear during the proof. For now, we note that they depend only on the given transducer T and the automaton A, hence they are well-defined and fixed within this proof.

Since T has unbounded trailing, it has two accepting computations

$$q_0 \xrightarrow{a}{uv} q_\triangle \xrightarrow{b_\triangle}{w_\triangle} f_\triangle \quad \text{and} \quad q_0 \xrightarrow{a}{u} q_\triangledown \xrightarrow{b_\triangledown}{vw_\triangledown} f_\triangledown$$

with $|v| > t$, where q_0 is the initial state and $f_\triangle$ and $f_\triangledown$ are accepting states of T. We decompose the *trail* v of length at least t as $\bar{v}v_k v_{k-1} \ldots v_2 v_1 \bar{v}$ with $|v_k| = t_k, \ldots, |v_1| = t_1$ and $|\bar{v}| = h$. We call $\bar{v}$ the *homestretch*. We consider the input-output prefix (a, uv) common to both computations of automaton A. The two-tape automaton model ensures that the two heads of A will eventually reach the end of a and v_1, respectively. Depending on which one does so first, we distinguish two cases.
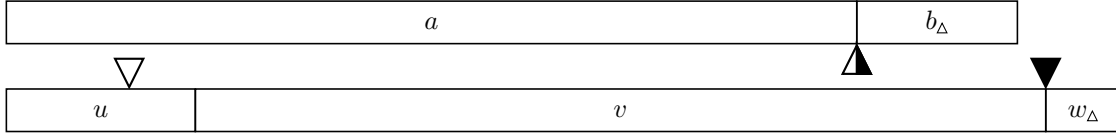
(a) Case $\triangle$: Automaton A's head positions when its input head has just reached the end of a and transducer T's head positions after its computation $q_0 \xrightarrow[uv]{a} q_\triangle$.
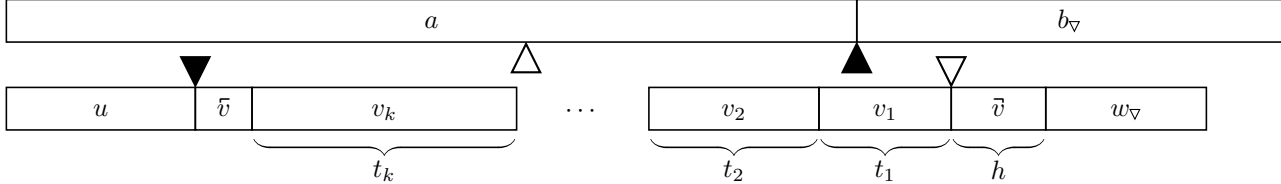
(b) Case $\triangledown$: Automaton A's head positions when the output head has just reached the end of v_1 and transducer T's head positions after computing $q_0 \xrightarrow[u]{a} q_\triangledown$.

Fig. 3: Two different input-output pairs in the relation $L(A) = L(T)$. The black and white triangles represent the heads of transducer T and automaton A, respectively. The second subfigure shows a decomposition $\bar{v}v_k v_{k-1} \ldots v_2 v_1 \bar{v}$ of v.

Case $\triangle$: Input head is first. In this case, we consider the input-output pair $(ab_\triangle, uvw_\triangle)$; see Figure 3a.

We begin by showing why we may assume without loss of generality that $|b_\triangle| \leq g$. For this, we consider transducer T's configuration after the computation $q_0 \xrightarrow[uv]{a} q_\triangle$; the corresponding head positions are indicated by the black triangles in Figure 3a. The shortcut guarantee g ensures the existence of a word pair $(b, w) \in \Sigma^* \times \Gamma^*$ and an accepting state $f \in F$ such that $q_\triangle \xrightarrow[w]{b} f$ and $|b| \leq g$; we can therefore substitute b, w, and f for $b_\triangle$, $w_\triangle$, and $f_\triangle$ if necessary and thus assume $b_\triangle$.

Now, we consider automaton A's computation on the same input-output pair $(ab_\triangle, uvw_\triangle)$. Let x denote the output word's suffix that has not yet been read by A when its input head has just reached the end of a; see the white triangles in Figure 3a for A's head positions. The input head has only $b_\triangle$ left to read but the output head all of x. We have already established that $|b_\triangle| \leq g$ using the shortcut guarantee, and we know that $|x| \geq |\bar{v}| = h$ since x contains the homestretch $\bar{v}$. In each step, A advances exactly one head by exactly one symbol, and a head does not move anymore once it has reached the end of the word written on its tape. Thus at most g movements remain for the input head but at least h for the output head. We call a step in which the input head moves an *input step* and a step in which the output head moves an *output step*. The input steps split the remaining computation into at most $g+1$ sequences of consecutive output steps. Since there are at least h output steps, there is at least one sequence of $h/(g + 1)$ uninterrupted output steps. Because of $h/(g+1) > |Q_A|$, this sequence contains at least two different output steps leading A into the same state. Choosing any two such steps, we can repeat a nonempty part of the output word arbitrarily often, namely, the part that starts at the position of the output head immediately after the first step and ends with the symbol at the position of the output head just before the second step. This results in arbitrarily many accepting computations, with the word on the input tape unmodified. Hence A associates infinitely many different output words with the same input word $ab_\triangle$, contradicting the k-valuedness of $L(A) = L(T)$.

Case $\triangledown$: Output head is first. In this case, automaton A's output head reaches the end of v_1 before its input head has finished reading a. This remains true for A's computation on the input-output pair $(ab_\triangledown, uvw_\triangledown)$; see Figure 3b.

In what follows, we establish an upper bound on the length of the remaining output $\bar{v}w_\triangledown$. The homestretch length $|\bar{v}| = h$ is already fixed. The length of $w_\triangledown$ can be bounded by combining the shortcut guarantee g and the output speed s as follows. Consider the transducer's computation

$$q_\triangledown \xrightarrow[vw_\triangledown]{b_\triangledown} f_\triangledown.$$

Since the output head can write at most s symbols per step, we can cut off from $b_\triangledown$ a prefix $\bar{b}_\triangledown$ such that

$$q_\triangledown \xrightarrow[v\bar{w}_\triangledown]{\bar{b}_\triangledown} q'$$

for some prefix $\bar{w}_\triangledown$ of $w_\triangledown$ with length $|\bar{w}_\triangledown| < s$ and some state q'. Denote by $\bar{b}_\triangledown$ and $\bar{w}_\triangledown$ the remaining suffixes such that $b_\triangledown = \bar{b}_\triangledown \bar{b}_\triangledown$ and $w_\triangledown = \bar{w}_\triangledown \bar{w}_\triangledown$. The state q' is useful as evidenced by the accepting computation

$$q_0 \xrightarrow[u]{a} q_\triangledown \xrightarrow[v\bar{w}_\triangledown]{\bar{b}_\triangledown} q' \xrightarrow[\bar{w}_\triangledown]{\bar{b}_\triangledown} f_\triangledown.$$

Using the definition of the shortcut guarantee g, we can therefore assume $|\bar{b}_\triangledown| \leq g$ by substituting a sufficiently short b' and some appropriate w' and f' for $\bar{b}_\triangledown$, $\bar{w}_\triangledown$, and $f_\triangledown$ if necessary. From this, we then obtain $|\bar{w}_\triangledown| \leq g \cdot s$ since the output speed s tells us how many symbols the transducer can output at most when reading one input symbol. We can thus assume that

$$|w_\triangledown| = |\bar{w}_\triangledown \bar{w}_\triangledown| < s + g \cdot s = (g + 1)s$$

without loss of generality. This finally yields the desired upper bound $|\bar{v}w_\triangledown| < h + (g + 1)s$.

Recall that $\bar{v}v_k v_{k-1} \ldots v_2 v_1 \bar{v}$ is the unique decomposition of v with $|v_k| = t_k, \ldots, |v_1| = t_1$ and $\bar{v} = h$. Since $t_0 = h + (g+1)s$, we immediately obtain

$$|v_i v_{i-1} \ldots v_1 \bar{v} w_\triangledown| \le t_0 + t_1 + \cdots + t_i.$$

Let $\bar{a}$ be a's suffix that is still unread at the moment when the output head of the automaton reaches the start of $\bar{v}$. Denote automaton A's initial state by $\hat{q}_0$ and choose any decomposition of a into $\bar{a}a_k a_{k-1} \ldots a_2 a_1 \hat{a}$ such that A's computation $\hat{q}_0 \xrightarrow{\quad ab_\triangledown \quad}{uvw_\triangledown} f_\triangledown$ splits into

$$\hat{q}_0 \xrightarrow{\bar{a}}{u\bar{v}} \hat{q}_{k+1} \xrightarrow{a_k}{v_k} \hat{q}_k \xrightarrow{a_{k-1}}{v_{k-1}} \ldots \xrightarrow{a_2}{v_2} \hat{q}_2 \xrightarrow{a_1}{v_1} \hat{q}_1 \xrightarrow{\hat{a}b_\triangledown}{\bar{v}w_\triangledown} f_\triangledown.$$

We will now prove the following claim by induction over $i \in \{0, 1, \ldots, k\}$:

Claim 16. *For every $i \in \{0, 1, \ldots, k\}$, there is an input word $b_{i+1} \in \Sigma^*$ and, for every $j \in \{1, 2, \ldots, i\}$, an output word $\tilde{v}_j \in \Gamma^*$ with $|\tilde{v}_j| < |v_j|$, such that transducer T has the accepting computation*

$$q_0 \xrightarrow{a}{u} q_\triangledown \xrightarrow{\quad b_{i+1} \quad}{\bar{v}v_k v_{k-1} \ldots v_{i+1} \tilde{v}_i \tilde{v}_{i-1} \ldots \tilde{v}_2 \tilde{v}_1 \bar{v}w_\triangledown} f_\triangledown$$

and such that automaton A has $i+1$ computations that all read the same input word ab_{i+1} overall, all begin with

$$\hat{q}_0 \xrightarrow{\quad \bar{a}a_k a_{k-1} \ldots a_{i+1} \quad}{u\bar{v}v_k v_{k-1} \ldots v_{i+1}} \hat{q}_{i+1},$$

and from $\hat{q}_{i+1}$ lead to accepting states while reading the remaining input $a_i \ldots a_2 a_1 \hat{a}b_i$ and the $i+1$ distinct outputs

$$v_i v_{i-1} \ldots v_3 v_2 v_1,$$
$$v_i v_{i-1} \ldots v_3 v_2 \tilde{v}_1,$$
$$v_i v_{i-1} \ldots v_3 \tilde{v}_2 \tilde{v}_1,$$
$$\ldots,$$
$$v_i \tilde{v}_{i-1} \ldots \tilde{v}_3 \tilde{v}_2 \tilde{v}_1, \text{ and}$$
$$\tilde{v}_i \tilde{v}_{i-1} \ldots \tilde{v}_3 \tilde{v}_2 \tilde{v}_1,$$

respectively.

Instantiated for $i = k$, this claim shows that the relation $\mathcal{L}(A) = \mathcal{L}(T)$ associates a single input word ab_{k+1} with $k+1$ distinct output words of pairwise different lengths, contradicting the k-valuedness and thus concluding the proof of Theorem 15.

We now briefly outline the proof of Claim 16; the full is provided afterward. Proving the induction basis is trivial: For $i = 0$, the claim coincides precisely with the situation shown in Figure 3b when setting $b_1 = b_\triangledown$. It remains to prove the induction step. Assuming the claim for $i - 1$ as our induction hypothesis, our goal is to find nonempty subwords $\mathring{b}_i$ of b_i and $\mathring{v}_i$ of v_i such that not only transducer T's accepting computation loops on them, but all of A's pairwise distinct accepting computations on the i different input-output pairs loop on the common input subword $\mathring{b}_i$ without advancing the output head at all. This is achievable since the length t_i of the subword v_i in the output is recursively defined to be sufficiently large—allowing us to force transducer T into producing plenty of output while reading the input b_i—and simultaneously small enough—ensuring that automaton A has

very few steps that advance the output head while reading b_i. $\qquad \square$

We now prove Claim 16 used in the proof above.

Proof of Claim 16. Establishing the induction basis is trivial: For $i = 0$, the claim coincides precisely with the situation shown in Figure 3b when setting $b_1 = b_\triangledown$. It remains to prove the induction step. We fix an arbitrary $i \in \{1, \ldots, k\}$, assume the claim for $i - 1$ as our induction hypothesis, and prove it for i. First consider T's computation given by the induction hypothesis for $i - 1$. Because T produces at most s output symbols when reading one input symbol, there are decompositions $b_i = \bar{b}_i \mathring{b}_i \bar{b}_i$ and $v_i = \bar{v}_i \mathring{v}_i \bar{v}_i$ with $|\bar{v}_i| < s$ and $|\bar{v}_i| < s$ such that T has a computation

$$q_0 \xrightarrow{a}{u} q_\triangledown \xrightarrow{\quad \bar{b}_i \quad}{\bar{v}v_k v_{k-1} \ldots v_{i+1} \bar{v}_i} \bar{q}_i \xrightarrow{\mathring{b}_i}{\mathring{v}_i} \mathring{q}_i$$
$$\xrightarrow{\quad \bar{b}_i \quad}{\bar{v}_i \tilde{v}_{i-1} \ldots \tilde{v}_1 \bar{v}w_\triangledown} f_\triangledown.$$

Note that

$$|\bar{v}_i| > t_i - 2s = s \cdot (1 + i(t_0 + t_1 + \cdots + t_{i-1})) \cdot (1 + |Q_T| \cdot |Q_A|^i).$$

Thus $\bar{q}_i \xrightarrow{\mathring{b}_i}{\mathring{v}_i} \mathring{q}_i$ contains more than

$$(1 + i(t_0 + t_1 + \cdots + t_{i-1})) \cdot (1 + |Q_T| \cdot |Q_A|^i)$$

steps during which T produces nonempty output. We call the positions of $\mathring{b}_i$ at which T produces nonempty output during its computation T-*productive*.

Now consider the i different computations of A given by the induction hypothesis for $i - 1$. They all start with a common prefix that splits into

$$\hat{q}_0 \xrightarrow{\quad \bar{a}a_k a_{k-1} \ldots a_{i+1} \quad}{u\bar{v}v_k v_{k-1} \ldots v_{i+1}} \hat{q}_{i+1} \xrightarrow{a_i}{v_i} \hat{q}_i.$$

Note that in the remainder of these computations, automaton A always scans the same input word $a_{i-1} \ldots a_2 a_1 b_i$ but i different output words. These output words are indeed pairwise different since their lengths are

$$\begin{aligned}
|\tilde{v}_i \tilde{v}_{i-1} \tilde{v}_{i-2} \ldots \tilde{v}_3 \tilde{v}_2 \tilde{v}_1 \bar{v}w_\triangledown| &< |v_i \tilde{v}_{i-1} \tilde{v}_{i-2} \ldots \tilde{v}_3 \tilde{v}_2 \tilde{v}_1 \bar{v}w_\triangledown| \\
&< |v_i v_{i-1} \tilde{v}_{i-2} \ldots \tilde{v}_3 \tilde{v}_2 \tilde{v}_1 \bar{v}w_\triangledown| \\
&< \ldots \\
&< |v_i v_{i-1} v_{i-2} \ldots v_3 \tilde{v}_2 \tilde{v}_1 \bar{v}w_\triangledown| \\
&< |v_i v_{i-1} v_{i-2} \ldots v_3 v_2 \tilde{v}_1 \bar{v}w_\triangledown| \\
&< |v_i v_{i-1} v_{i-2} \ldots v_3 v_2 v_1 \bar{v}w_\triangledown| \\
&\le t_0 + t_1 + \cdots + t_i.
\end{aligned}$$

Since these are i different computations, each of which has at most $t_0 + t_1 + \cdots + t_{i-1}$ output symbols, there are within the remaining $a_{i-1} \ldots a_2 a_1 b_i$ input suffix at most $i(t_0 + t_1 + \cdots + t_{i-1})$ positions such that, for at least one of the i computations by A, the input head of A stays put in said position while its output head advances. We call these positions A-*productive* and the others A-*unproductive*. It follows in particular that the subword $\bar{b}_i$ in $b_i = \bar{b}_i \mathring{b}_i \bar{b}_i$ contains at most

$$i(t_0 + t_1 + \cdots + t_{i-1})$$

positions that are A-productive. They delimit at most

$$1 + i(t_0 + t_1 + \cdots + t_{i-1})$$

sequences of consecutive A-unproductive positions in $\bar{b}_i$. Using the lower bound on the number of T-productive positions in $\bar{b}_i$ derived above, we conclude that at least one of these A-unproductive sequences contains more than

$$\frac{(1 + i(t_0 + t_1 + \cdots + t_{i-1})) \cdot |Q_T| \cdot |Q_A|^i}{1 + i(t_0 + t_1 + \cdots + t_{i-1})} = |Q_T| \cdot |Q_A|^i$$

T-productive positions. Thus $\bar{b}_i$ has a decomposition $\bar{b}'_i \mathring{b}_i \bar{b}'_i$ such that, on the one hand, all i computations of A given by the induction hypothesis for $i - 1$ loop through $\mathring{b}_i$ simultaneously without advancing the output head at all and, on the other hand, T's partial computation on $\bar{b}_i$ decomposes into

$$\bar{q}_i \xrightarrow{\frac{\bar{b}'_i}{\bar{v}'_i}} \mathring{q}_i \xrightarrow{\frac{\mathring{b}_i}{\mathring{v}_i}} \mathring{q}_i \xrightarrow{\frac{\bar{b}'_i}{\bar{v}'_i}} \bar{q}_i$$

with a loop that starts at some state $\mathring{q}_i$ and, while reading the input sequence $\mathring{b}_i$, produces some output $\mathring{v}_i$ that is nonempty due to at least one T-productive position in $\mathring{b}_i$. Removing the nonproductive loop $\mathring{b}_i$ from all i of A's computations given by the induction hypothesis for $i - 1$, we obtain all but the last of A's computations in the claim for i. Moreover, deleting the loop on $\mathring{b}_i$ from T's computation in the induction hypothesis for $i - 1$ and defining $b_{i+1} = \bar{b}_i \bar{b}'_i \bar{b}'_i \bar{b}_i$ and $\tilde{v}_i = \bar{v}_i \bar{v}'_i \bar{v}'_i \bar{v}_i$ yields the computation for T in the claim for i. Finally, we use $\mathcal{L}(A) = \mathcal{L}(T)$ to obtain from this new computation of T the missing last computation of A. All mentioned computations of A start with the same prefix

$$\hat{q}_0 \xrightarrow{\frac{\bar{a} a_k a_{k-1} \ldots a_{i+1}}{u \bar{v} v_k v_{k-1} \ldots v_{i+1}}} \hat{q}_{i+1}$$

since A is deterministic and all input and output in this prefix has remained unchanged. This concludes the induction step and thus the proof of Claim 16. □

We now provide an illustrative example of an fv-NFT with unbounded trailing. According to Theorem 15, it has no equivalent fv-2t-DFA, implying the strictness of the inclusion $\mathcal{L}(\text{fv-2t-DFA}) \subseteq \mathcal{L}(\text{fv-NFT})$; see Figure 1.

Example 17. *The transducer's alphabets are $\Sigma = \Gamma = \{0, 1\}$ and its relation is*

$$\{ (0^i 10^j 10, 0^i) \mid i, j \in \mathbb{N} \} \cup \{ (0^i 10^j 11, 0^j) \mid i, j \in \mathbb{N} \}.$$

This union contains exactly the pairs of the form $(0^i 10^j 1\beta, 0^k)$, where β is a bit valued 0 or 1 and $k = i$ if $\beta = 0$ and $k = j$ if $\beta = 1$. This relation is easily computed by an fv-NFT that nondeterministically guesses β, then copies either 0^i or 0^j to the output tape, and finally accepts or rejects depending on whether the guess was correct.

No 2t-DFA can compute this relation, however, for the following intuitive reason. Consider an input-output pair $(0^i 10^j 1\beta, 0^k)$ with sufficiently large i, j, and k. By the time the input head reaches the first 1, the output head has either read most of 0^k already or has a large part of it still lying ahead. In the first case, the automaton may have potentially checked whether $k = i$, but cannot remember how many zeroes of 0^k the output head has already passed making it impossible to check whether $k = j$ if the input ends with $\beta = 1$. In the second case, the automaton can potentially still check whether $k = j$, but cannot remember how many zeroes the

input head has already passed, that is, the value of i. This makes it impossible to check whether $k = i$ if the input ends with $\beta = 0$. Therefore, the automaton fails on inputs with $\beta = 1$ in the first case and on inputs with $\beta = 0$ in the second case.

To show formally that the described transducer has unbounded trailing we use the variable names of Definition 11. For any given $t \in \mathbb{N}$, we can choose $u = w_1 = w_2 = \lambda$, where λ denotes the empty word, $a = v = 0^t$, $b_1 = 10^t 10$, and $b_2 = 10^t 11$. This yields two computations showing that the trailing bound must be at least t, namely

$$q_0 \xrightarrow{\frac{0^t}{\lambda 0^t}} q_\triangle \xrightarrow{\frac{10^t 10}{\lambda}} f_\triangle \text{ and } q_0 \xrightarrow{\frac{0^t}{\lambda}} q_\triangledown \xrightarrow{\frac{10^t 11}{0^t \lambda}} f_\triangledown.$$

VI. Bounded Trailing is Undecidable

In this section, we prove that determining whether an fv-NFT has bounded trailing is undecidable. This is achieved by reducing the halting problem on the empty input, which is known to be undecidable, to the problem of determining whether an fv-NFT has bounded trailing. We present a reduction via a third problem, namely determining whether a Turing machine reaches infinitely many configurations on the empty input.

We begin by formally defining the standard model of a deterministic Turing machine with a single tape that is unbounded in both directions.

Definition 18. *A Turing machine is a sextuple $M = (Q, \Gamma, \textvisiblespace, q_0, F, \delta)$ consisting of*

- *a finite, nonempty set of states Q,*
- *a finite, nonempty alphabet Γ,*
- *a blank symbol $\textvisiblespace \notin \Gamma$,*
- *an initial state $q_0 \in Q$,*
- *a set $F \subseteq Q$ of accepting states, and*
- *a (partial) transition function*
  $\delta : Q \times (\Gamma \cup \{\textvisiblespace\}) \to Q \times \Gamma \times \{\text{Left}, \text{Right}\}.$

A *configuration* of a Turing machine consists of its current state, the content of the tape, and the position of the head on the tape. We will only consider the computations of a Turing machine on the empty input, the initial configuration thus always consists of the initial state q_0, a tape containing only blank symbols, and the head scanning one of them. A configuration is called *accepting* if its current state q is accepting, i.e., if $q \in F$. A configuration is called *halting* if it is accepting or the transition function is undefined for the current state $q \in Q$ and the symbol $a \in \Gamma$ currently scanned by the head. If a configuration is not halting, the next configuration reached in one step of the Turing machine's computation is obtained by updating the current state, writing a non-blank symbol to the tape's cell scanned by the head, and moving the head either one cell to the left or one cell to the right.

Any configuration reached during the Turing machine's computation on the empty input consists of a finite contiguous sequence of non-blank symbols and the position of the head scanning either any symbol within this sequence or one of the two blank symbols delimiting it. Hence, we can represent a configuration of this machine as a finite sequence of cells of

two types: Every configuration contains exactly one cell of the first type, namely the one currently scanned by the head. In our representation, this type of cell contains some potentially blank symbol and the current state. The second type contains a non-blank symbol only. The initial configuration is represented by a single cell containing the blank symbol and the initial state—recall that we consider only the computation on the empty input.

A Turing machine halts on the empty input if it reaches a halting configuration during its computation starting in the initial configuration. The undecidability of determining whether a halting configuration can be reached is well known [19]. A straightforward reduction shows that it is also undecidable whether a given Turing machine reaches infinitely many different configurations during its computation on the empty input.

Lemma 19. *The problem of determining whether a Turing machine reaches infinitely many different configurations during its computation on the empty input is undecidable.*

Proof. We prove the lemma by contradiction. Suppose that there is an algorithm A_∞ that decides for every given Turing machine M whether it reaches infinitely many different configurations on the empty input. We will use A_∞ to design an algorithm A_H that decides for any Turing machine M whether it reaches a halting configuration on the empty input. The latter problem is known to be undecidable, yielding the desired contradiction.

Given a Turing machine M, algorithm A_H first invokes A_∞ to decide whether M reaches infinitely many different configurations on the empty input. If it does, then A_H outputs "No" because reaching a halting configuration implies reaching only finitely many configurations in total. Otherwise, A_H simulates M's deterministic computation on the empty input step by step, remembering all configurations, until either a halting or a previously encountered configuration is reached. Then A_H outputs "Yes" in the former case and "No" in the latter. □

Because both the set of states and the alphabet are finite, the set of all configurations of a bounded length is necessarily finite. It follows that a Turing machine M reaches infinitely many configurations if and only if it reaches configurations of arbitrary length.

Corollary 20. *The problem of determining whether a Turing machine reaches configurations of arbitrary length on the empty input is undecidable.*

We now show how to construct from a given deterministic Turing machine M an fv-NFT T that has unbounded trailing if and only if M reaches configurations of arbitrary length.

A valid input for T is a sequence of M's configurations followed by one of two special symbols that we call *mode indicators*. The configurations are represented by finite sequences of cells as described in the previous section and separated from each other by a dedicated symbol not occurring anywhere else. The two mode indicators are represented by a cell containing either of the two words in $\{\mathrm{copy}, \mathrm{step}\}$. The transducer T starts its computation by nondeterministically guessing the

mode indicator and then operates in the corresponding mode described below. If the guess turns out to be wrong or if the input is invalid in any way, the computation is aborted and the transducer rejects the input. The two possible types of input-output pairs after an accepting computation are depicted in Figure 4a.

Copy Mode
> The input is output symbol by symbol, omitting the mode indicator in the end.

Step Mode
> In the first step, output M's entire fixed initial configuration c_0 while reading and remembering the first input symbol. Then read from the sequence in the input one configuration c after the other. While reading c, compute and output the successor configuration c' that M reaches from c in one computation step. Of course, this all assumes that such a configuration c' exists; otherwise, T aborts the computation as it does for invalid inputs.

To see that T can in fact realize these computations, observe that the changes necessary to turn a configuration c into its successor configuration c' do, on the one hand, only depend on the single type-one cell of c containing the currently scanned symbol and the current state and, on the other hand, only affect the immediate proximity of this cell.

Hence, the successor configuration c' can indeed be computed from c by a finite transducer that essentially is still copying each configuration symbol by symbol, but letting the input head run ahead by one cell while keeping the contents of the three most recent input cells in a buffer. This allows the transducer to modify the configuration in the right place to produce the successor configuration even for configuration changes of the Turing machine that move its only head to the left. Transducer T can be effectively computed from any given Turing machine M; we omit the technicalities.

The described transducer T is *unambiguous*; that is, there is at most one accepting computation for every input word. This is easily checked as follows. On the one hand, T rejects all invalid inputs anyway. On a valid input, on the other hand, T takes only a single nondeterministic decision to choose the operating mode and then the computation proceeds deterministically, being accepting for only one of the two choices, depending on the mode indicator at the end of the input word. Transducer T's being unambiguous implies that the undecidability stated in Theorem 23 remains valid even when we are restricted to unambiguous NFTs instead of fv-NFTs. Namely, unambiguity implies functionality and thus finite-valuedness—but of course not vice versa—which trivially means that Theorems 14 and 15 hold for functional and unambiguous NFTs too. Nevertheless, unambiguous transducers are just as expressive as functional ones [7]. We state T's relevant properties formally.

Claim 21. *T is unambiguous and thus also functional and finite-valued; that is, T is an f-NFT and an fv-NFT.*

We will now sketch the proof of the crucial connection between the length of M's configurations and T's trailing.

(a) The two types of valid inputs for T and the corresponding accepted outputs. Each c_i with $i > 0$ encodes an arbitrary configuration of M, c_0 is its initial configuration, and c_i' is c_i's successor configuration under a single computation step of M.

(b) Two accepting computation patterns of T that make unbounded trailing inevitable if M reaches arbitrarily long configurations. Here, c_0 is still the initial configuration of M on the empty input, but c_{i+1} is now the successor configuration of c_i under a single computation step of M.

Fig. 4: Accepting computation patterns of transducer T, which depends on Turing machine M.

Claim 22. *T has unbounded trailing if and only if Turing machine M reaches configurations of arbitrary length during its computation on the empty input.*

Proof. The transducer T takes only one nondeterministic decision during any computation, namely to operate in either copy or step mode, the rest of the computation is deterministic. According to Definition 11, the only way for any trailing to occur is therefore a pair of two computations, one in copy mode and one in step mode, producing consistent output words. One of these two output words must be a prefix of the other; we call it the common prefix of the two computations. See Figure 4b for an example of the arising situation.

In the following paragraph we argue why the configurations within the common prefix represent a valid computation of M and why the current trail length is equal to the length of the configuration currently read, up to a constant.

Since transducer T always starts by outputting the initial configuration c_0 in step mode, this has to be the first configuration on the output tape in copy mode as well. In copy mode, T can only write this configuration c_0 to the beginning of the output tape if c_0 is also the first configuration on the input tape. The step mode behavior now ensures that the second configuration on the output tape is the successor of c_0, we call it c_1. Iterating this argument, we see that the common prefix indeed contains a valid computation $c_0, c_1, \ldots, c_k$, where each configuration is the successor of the previous one under one computation step of M. Finally, the trail is always as long as the configuration that is currently being read, up to the size of the one cell by which the input head is leading. Thus the current trail length is indeed always, up to constant, equal to the length of the current configuration in the simulation of the

computation of M on the empty input word. This concludes the proof of Claim 22. □

Finally, we obtain the main result of this section by combining Corollary 20 and Claims 21 and 22.

Theorem 23. *Whether a given fv-NFT has bounded trailing is an undecidable problem.*

Proof. We prove the theorem by contradiction. Suppose that there is an algorithm A deciding whether an fv-NFT has bounded trailing. We construct an algorithm A_∞ deciding whether a Turing machine M reaches configurations of arbitrary length. The latter problem is undecidable by Corollary 20, yielding a contradiction.

Given a Turing machine M, algorithm A_∞ constructs the fv-NFT T and uses algorithm A to decide whether T has bounded trailing. If it does, then A_∞ outputs "No," otherwise it outputs "Yes." The correctness of algorithm A_∞ follows from Claim 22. □

VII. Conclusion

We have introduced the notion of bounded trailing, characterized fv-NFTs having an equivalent fv-2t-DFA as those with bounded trailing, proved that it is undecidable whether an fv-NFT has bounded trailing—thereby proving fv-NFTs to be more powerful than fv-2t-DFAs—and showed how to construct an equivalent fv-2t-DFA if there is one. Furthermore, all of these results hold true for functional and unambiguous transducers as well because Theorems 14 and 15 prove the characterization for any fv-NFT—which includes the functional and unambiguous finite transducers—and because we have shown our criterion's undecidability using an unambiguous finite transducer, which is the most restrictive.

On the one hand, our results mirror Choffrut's classical characterization of determinizable NFTs as those with the twinning property [6]. One the other hand, the undecidability of our bounded-trailing criterion contrasts with the twinning property being decidable in polynomial time [21]. Despite this undecidability, we provide a uniform construction that transforms an fv-NFT into an fv-2t-DFA whenever possible.

Fischer and Rosenberg [11] have shown already more than half a century ago that deciding whether a 2t-NFA (or equivalently, a λ-NFT) can be transformed into an equivalent 2t-DFA is an undecidable problem; however, their proof requires relations that are not finite-valued and thus does not yield any result for fv-NFTs. Moreover, neither a characterization of transducers with an equivalent deterministic automaton nor a method to construct an equivalent deterministic automaton if it exists has been described so far. The present paper addresses all of these issues for the case of fv-NFTs.

We emphasize again that the bounded trailing property defined in this paper does not characterize NFTs having an equivalent 2t-DFA unless the relation is finite-valued; a corresponding counterexample of a transducer with both an equivalent 2t-DFA and unbounded trailing is found in Example 12. Finding a reasonable characterization of NFTs with an equivalent 2t-DFA and a method to transform the former into the latter if possible remains as an open research opportunity.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. A general theory of translation. *Math. Systems Theory*, 3:193–221, 1969.

[2] Anonymous Authors. Formalization associated with this paper, 2021. https://github.com/lics21/automata.

[3] O. Carton. Two-way transducers with a two-way output tape. In *Developments in Language Theory*, pages 263–272. Springer Berlin Heidelberg, 2012.

[4] O. Carton, L. Exibard, and O. Serre. Two-way two-tape automata. In *Developments in Language Theory - 21st International Conference, DLT 2017*, volume 10396 of *Lecture Notes in Computer Science*, pages 147–159. Springer, 2017.

[5] C. Choffrut. Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles. *Theoretical Computer Science*, 5(3):325–337, 1977.

[6] C. Choffrut. A generalization of Ginsburg and Rose's characterization of G-S-M mappings. In *Automata, Languages and Programming*, pages 88–103. Springer Berlin Heidelberg, 1979.

[7] C. Choffrut and S. Grigorieff. Uniformization of rational relations. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 59–71. Springer, 1999.

[8] C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, 1965.

[9] J. Engelfriet and H. J. Hoogeboom. Two-way finite state transducers and monadic second-order logic. In *Automata, Languages and Programming*. Springer, 1999.

[10] E. Filiot, O. Gauwin, P.-A. Reynier, and F. Servais. From two-way to one-way finite state transducers. In *LICS 2013*, pages 468–477. IEEE Computer Society, 2013.

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

[12] M. Holzer, M. Kutrib, and A. Malcher. Complexity of multi-head finite automata: Origins and directions. *Theoretical Computer Science*, 412(1):83–96, 2011.

[13] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

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

[15] M. Raszyk, D. A. Basin, and D. Traytel. From nondeterministic to multi-head deterministic finite-state transducers. In *ICALP 2019*, volume 132 of *LIPIcs*, pages 127:1–127:14, 2019.

[16] C. Reutenauer and M.-P. Schützenberger. Minimization of rational word functions. *SIAM J. Comput.*, 20(4):669–685, 1991.

[17] M. P. Schützenberger. Sur les relations rationnelles. In *Automata Theory and Formal Languages*, pages 209–213, Berlin, Heidelberg, 1975.

[18] I. H. Sudborough. On tape-bounded complexity classes and multihead finite automata. *Journal of Computer and System Sciences*, 10(1):62–76, 1975.

[19] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[20] A. Weber. On the valuedness of finite transducers. *Acta Informatica*, 27(8):749–780, 1989.

[21] A. Weber and R. Klemm. Economy of description for single-valued transducers. *Information and Computation*, 118(2):327–340, 1995.

[22] A. C. Yao and R. L. Rivest. $k + 1$ heads are better than k. *J. ACM*, 25(2):337—-340, 1978.