arXiv:2205.15631v2 [cs.FL] 4 Jun 2022

Computational and Descriptional Power of Nondeterministic Iterated Uniform Finite-State Transducers[*]

Martin Kutrib[†], Andreas Malcher

Institut für Informatik, Universität Giessen

Arndtstr. 2, 35392 Giessen, Germany

{kutrib, andreas.malcher}@informatik.uni-giessen.de

Carlo Mereghetti, Beatrice Palano

Dip. di Informatica "G. degli Antoni", Università degli Studi di Milano

via Celoria 18, 20133 Milano, Italy

{carlo.mereghetti, beatrice.palano}@unimi.it

Abstract. An iterated uniform finite-state transducer (IUFST) runs the same length-preserving transduction, starting with a sweep on the input string and then iteratively sweeping on the output of the previous sweep. The IUFST accepts the input string by halting in an accepting state at the end of a sweep. We consider both the deterministic (IUFST) and nondeterministic (NIUFST) version of this device. We show that constant sweep bounded IUFSTs and NIUFSTs accept all and only regular languages. We study the state complexity of removing nondeterminism as well as sweeps on constant sweep bounded NIUFSTs, the descriptional power of constant sweep bounded IUFSTs and NIUFSTs with respect to classical models of finite-state automata, and the computational complexity of several decidability questions. Then, we focus on non-constant sweep bounded devices, proving the existence of a proper infinite nonregular language hierarchy depending on the sweep complexity both in the deterministic and nondeterministic case. Though NIUFSTs are "one-way" devices we show that they characterize the class of context-sensitive languages, that is, the complexity class DSpace(lin). Finally, we show that the nondeterministic devices are more powerful than their deterministic variant for a sublinear number of sweeps that is at least logarithmic.

Keywords: Iterated transducers, nondeterminism, descriptional complexity, sweep complexity, language hierarchies.

1.　Introduction

The notion of an iterated uniform finite-state transducer (IUFST) has been introduced in [2] (see [3] for the journal version) and can be described as a finite transducer that iteratively sweeps from left to right over the input tape while performing the same length-preserving transduction. In particular, the output of the previous sweep is taken as input for every new sweep. This model is motivated by typical applications of transducers or cascades of transducers, where the output of one transducer is used as the input for the next transducer. For example, finite state transducers are used for the lexical analysis of computer programs and the produced output is subsequently processed by pushdown automata for the syntactical analysis. In [4], cascades of finite-state transducers are used in natural language processing. Another example is the Krohn-Rhodes decomposition theorem which shows that every regular language is representable as the cascade of several finite state transducers, each one having a "simple" algebraic structure [5, 6]. Finally, it is shown in [7] that cascades of deterministic pushdown transducers lead to a proper infinite hierarchy in between the deterministic context-free and the deterministic context-sensitive languages with respect to the number of transducers involved.

In contrast to all these examples and other works in the literature (see, e.g., [8, 9, 10]), where the subsequently applied transducers are in principle different and not necessarily length-preserving, the model of IUFSTs introduced in [2] requires that the same transducer is applied in every sweep and that the transduction is deterministic and length-preserving. More precisely, an IUFST works in several sweeps on a tape which initially contains the input string concatenated with a right endmarker. In every sweep, the finite state transducer starts in its initial state at the first tape cell, is applied to the tape, and prints its output on the tape. The input is accepted, if the transducer halts in an accepting state at the end of a sweep. In [2], IUFSTs both with a constant number and a non-constant number of sweeps are investigated. In the former case, it is possible to characterize exactly the set of regular languages and upper and lower state bounds for converting IUFSTs into deterministic finite automata (DFAs) and vice versa are established. Furthermore, as always done for several models (see, e.g., [11, 12, 13, 14, 15, 16]), the state complexity of language operations, that is, the costs in terms of the number of states needed for union, intersection, complementation, and reversal, is investigated in depth. Finally, the usually studied decidability questions such as emptiness, finiteness, equivalence, and inclusion are proved to be NL-complete, showing that these questions have the same computational complexity as for DFAs. For the case of a non-constant number of sweeps, the situation is quite different. It is shown that a logarithmic number of sweeps is sufficient to accept unary non-semilinear languages, while with a sublogarithmic number of sweeps only regular languages can be accepted. Moreover, the existence of a finite hierarchy with respect to the number of sweeps is obtained. Finally, all usually studied decidability questions are shown to be undecidable and not even semidecidable for IUFSTs performing at least a logarithmic number of sweeps.

In this paper, we enhance the model of IUFSTs by *nondeterminism*, thus obtaining their *nondeterministic* version (NIUFSTs). As in [2], we are interested in NIUFSTs exhibiting both a constant and non-constant number of sweeps.

Constant sweep bounded NIUFSTs are proved to accept exactly regular languages. So, their ability of representing regular languages in a very succinct way turns out to be worth investigating, as well as comparing such an ability with that of other more traditional models of finite-state automata. This type of investigation, whose importance is witnessed by a well consolidated trend in the literature,

focuses on the number of states for representing languages and belongs to the area of descriptional complexity. Being able to have "small" devices representing/accepting certain languages, leads to relevant consequences either from a practical point of view (less hardware needed to construct such devices, less energy absorption, less cooling problems, *etc.*), and from a theoretical point of view (higher manageability of proofs and representations for languages, reductions of difficult problems on general computing devices to the same problems on simpler machines, *etc.*). The reader is referred to, e.g., [17], for a thoughtful survey on descriptional complexity and its consequences.

Non-constant sweep bounded NIUFSTs are then studied for their computational power, i.e., the ability of accepting language families. In particular, such an ability is related to the number of sweeps as a function of the input length.

After defining NIUFSTs in Section 2, we discuss in detail an example that demonstrates the state number advantages of NIUFSTs with a constant number of sweeps in comparison with its deterministic variant and the classical models of deterministic and nondeterministic finite automata (NFAs). Precisely, we exhibit a language accepted by an NIUFST such that any equivalent IUFST requires exponentially more states and sweeps, while any equivalent NFA (resp., DFA) requires exponentially (resp., double-exponentially) more states.

In Section 3, we consider NIUFSTs with a constant number of sweeps in more generality. By evaluating the state cost of sweep removal, we show that any NIUFST featuring n states and k sweeps can be simulated by a $2n^k$-state NFA, and hence by a 2^{2n^k}-state DFA as well. Next, we exhibit a unary (resp., binary) language to establish lower bounds for the obtained size blow-up for turning a constant sweep NIUFST into an equivalent NFA (resp., DFA). Moreover, we study the computational complexity of several decidability questions for NIUFST with k sweeps and obtain NL-completeness results for the questions of emptiness, finiteness, and infiniteness, whereas the questions of universality, inclusion, and equivalence are shown to be PSPACE-complete.

In the last two sections, we consider NIUFSTs with a non-constant number of sweeps. First, we establish in Section 4 an infinite proper hierarchy with respect to the number of sweeps. Interestingly, this result also extends the known finite hierarchy in the deterministic case to an infinite hierarchy. Then we show that NIUFSTs can simulate linear bounded automata, though NIUFSTs are "one-way" devices where the information flow is from left to right only. So, NIUFSTs whose sweep complexity is not bounded a priori characterize the class of context-sensitive languages, that is, they capture the complexity class DSpace(lin).

Finally, we study in Section 5 the question of whether the nondeterministic model is more powerful than the deterministic model. We get that the question can be answered in the affirmative if at least a logarithmic number of sweeps is provided. Moreover, we show that nondeterminism cannot be matched in power by the deterministic paradigm even if a sublinear number of sweeps is given.

2. Definitions and preliminaries

We denote the set of positive integers and zero by $\mathbb{N}$. Set inclusion is denoted by $\subseteq$ and strict set inclusion by $\subset$. Given a set S, we write 2^S for its power set and $|S|$ for its cardinality. Let Σ^* denote the set of all words over the finite alphabet Σ. The *empty word* is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The length of a word w is denoted by $|w|$. By $\operatorname{ld} n$ we denote the logarithm of n to base 2.

Roughly speaking, an iterated uniform finite-state transducer is a finite-state transducer which processes the input in multiple passes (also sweeps). In the first pass it reads the input word followed by an endmarker and emits an output word. In the following passes it reads the output word of the previous pass and emits a new output word. It can be seen as a restricted variant of a one-tape Turing machine. The number of passes taken, the *sweep complexity*, is given as a function of the length of the input. Here, we are interested in weak processing devices: we will consider length-preserving finite-state transducers, also known as Mealy machines [18], to be iterated.

Formally, we define a *nondeterministic iterated uniform finite-state transducer* (NIUFST) as a system $T = \langle Q, \Sigma, \Delta, q_0, \lhd, \delta, F \rangle$, where:

- Q is the finite set of *internal states*,

- Σ is the set of *input symbols*,

- Δ is the set of *output symbols*,

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

- $\lhd \in \Delta \setminus \Sigma$ is the *endmarker*,

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

- $\delta \colon Q \times (\Sigma \cup \Delta) \to 2^{Q \times \Delta}$ is the partial *transition function*.

The NIUFST T *halts* whenever the transition function is undefined or whenever it enters an accepting state at the end of a sweep. Since the transducer is applied in multiple passes, that is, in any but the initial pass it operates on an output of the previous pass, the transition function depends on input symbols from $\Sigma \cup \Delta$. We denote by $T(w)$ the set of possible outputs produced by T in a complete sweep on input $w \in (\Sigma \cup \Delta)^*$. During a computation on input $w \in \Sigma^*$, the NIUFST T produces a sequence of words $w_1, \ldots, w_i, w_{i+1}, \ldots \in (\Sigma \cup \Delta)^*$, where $w_1 \in T(w \lhd)$ and $w_{i+1} \in T(w_i)$ for $i \geq 1$.

An NIUFST is said to be *deterministic* (IUFST) if and only if $|\delta(p, x)| \leq 1$, for all $p \in Q$ and $x \in \Sigma \cup \Delta$. In this case, we simply write $\delta(p, x) = (q, y)$ instead of $\delta(p, x) = \{(q, y)\}$ assuming that the transition function is a mapping $\delta \colon Q \times (\Sigma \cup \Delta) \to Q \times \Delta$.

Now we turn to language acceptance. With respect to nondeterministic computations and some complexity bound, in the literature several acceptance modes are considered. For example, a machine accepts a language in the *weak mode*, if for any input $w \in L$ there is an accepting computation that obeys the complexity bound. Language L is accepted in the *strong mode*, if the machine obeys the complexity bound for all computations (accepting or not) on all inputs. Here we deal with the number of sweeps as (computational) complexity measure. The weak mode seems too optimistic for this measure, while the strong mode seems too restrictive. Therefore, here we consider an intermediate mode, the so-called accept mode. A language is accepted in the *accept mode* if all accepting computations obey the complexity bound (see [19] for separation of these modes with respect to space complexity).

A computation is halting if there exists an $r \geq 1$ such that T halts on w_r, thus performing r sweeps. The input word $w \in \Sigma^*$ is *accepted* by T if at least one computation on w halts at the end of a sweep in an accepting state. That is, the initial input is a word over the input alphabet Σ

followed by the endmarker, and there is an output computed after $r - 1$ sweeps that drives T in a complete final sweep where it halts in an accepting state. Otherwise, it is *rejected*. Note that the output of the last sweep is not used. The language accepted by T is the set $L(T) \subseteq \Sigma^*$ defined as $L(T) = \{ w \in \Sigma^* \mid w \text{ is accepted by } T \}$.

Given a function $s \colon \mathbb{N} \to \mathbb{N}$, an iterated uniform finite-state transducer T is said to be of *sweep complexity* $s(n)$ if for all $w \in L(T)$ all accepting computations on w halt after at most $s(|w|)$ sweeps. In this case, we add the prefix $s(n)$- to the notation of the device. It is easy to see that 1-IUFSTs (resp., 1-NIUFSTs) are essentially deterministic (resp., nondeterministic) finite-state automata (DFAs and NFAs, respectively).

Throughout the paper, two accepting devices are said to be *equivalent* if and only if they accept the same language.

We chose to denote our transducers as "uniform" since they perform the same transduction at each sweep: they always start from the same initial state on the leftmost tape symbol, operating the same transduction rules at every computation step. Yet, we quickly observe that an NIUFST is clearly a restricted version of a linear bounded automaton (see, e.g., [20]). So, any language accepted by an NIUFST is context sensitive and it will turn out that the converse is also true.

Accepting languages by iterated transductions: an example

In order to clarify the notion of acceptance by iterated transduction, for any integer $k \geq 2$, we consider the block language

$$B_k = \{ u_1 \# u_2 \# \cdots \# u_m \mid u_i \in \{0, 1\}^k,\ m > 1,\ \exists i < m \colon u_i = u_m \}.$$

By counting arguments, it is not hard to see that at least 2^{2^k+1} states are necessary to accept B_k by a DFA. On the other hand, an exponentially smaller NFA A may accept B_k as follows.

1. In a first phase, on each block in the input string, A stores the block in its finite control and then nondeterministically decides whether to keep the block or to ignore it. Along this phase, the correct block structure of the input scanned so far is checked by A as well. This phase takes 2^{k+1} states.

2. Once A decides to keep a block, say u, in its finite control, a second phase starts in which A scans the rest of the input checking the correct block structure and guessing another block w to be matched symbol-by-symbol against u. If the matching is successful and w is the last block, then A accepts. This phase takes at most $2^{k+1} \cdot (k+1)$ states.

Altogether, the NFA A features $2^{k+1} + 2^{k+1} \cdot (k+1) = 2^{k+1} \cdot (k+2)$ states.

Indeed, A can also be seen as a $2^{k+1} \cdot (k+2)$-state 1-NIUFST which outputs the scanned symbol at each step. However, paying by the number of sweeps (see, e.g., [21]), we can build a k-NIUFST T for B_k with only $O(k)$ states. Informally:

1. In a first sweep, T checks the correct block structure of the input string, nondeterministically chooses two blocks to be matched symbol-by-symbol, and compares the first symbol of the two blocks by storing the first symbol of the first block in its finite control and replacing these two symbols with a blank symbol.

2. At the ith sweep, T checks the ith symbol of the two blocks chosen in the first sweep by storing and blank-replacing symbols as explained at the previous point. To distinguish the first sweep (where both nondeterministic block choices and symbol comparisons take place) from the others (where only symbol comparisons take place), a special symbol can replace the first input symbol at the beginning of the first sweep.

It is not hard to see that $O(k)$ states are needed to check the input formatting along the first sweep, and that a constant number of states suffices to blank-replacing and comparing input symbols. Indeed, after k sweeps all nondeterministically chosen block symbols are compared so that T may correctly accept or reject. This gives the claimed state and sweep bounds for T.

We remark that: (i) a $2^k(k+4)$-state 2^k-IUFST is designed in [3] for B_k, (ii) 2^{2^k+1} states are necessary to accept B_k by a DFA, and that (iii) $2n^k$ states are sufficient for a DFA to simulate an n-state k-IUFST [3]. These facts, together with the above designed $O(k)$-states k-NIUFST, show that NIUFSTs can be exponentially more succinct than IUFSTs either in the number of states, or in the number of sweeps, or possibly both. Indeed, we also have that NIUFSTs can be exponentially more succinct than NFAs and double-exponentially more succinct than DFAs.

In the next section, we approach more generally the analysis of the descriptional power of NIUFSTs with respect to their deterministic counterparts and classical finite-state models.

3. NIUFSTs with a constant number of sweeps

3.1. Reducing sweeps and removing nondeterminism

Let us begin by showing how to reduce sweeps from NIUFSTs and evaluate the state cost of reduction. We will then use this construction to reduce constant sweep bounded NIUFSTs to one sweep NIUFSTs, thus obtaining equivalent NFAs whose number of states will be suitably bounded.

Theorem 3.1. Let $n, k, i > 0$ be integers and $i \leq k$. Every n-state k-NIUFST (resp., k-IUFST) can be converted to an equivalent $2n^i$-state $\lceil \frac{k}{i} \rceil$-NIUFST (resp., $\lceil \frac{k}{i} \rceil$-IUFST).

Proof:
The principal idea of the construction of an equivalent $\lceil \frac{k}{i} \rceil$-NIUFST T' is that the state set of T' is used to simulate i sweeps of the given k-NIUFST T in parallel, step by step. There may occur the problem that T gets stuck in some sweep $j \leq k$, but an earlier sweep $1 \leq \ell < j$ ends accepting. To cope with this problem, we enforce T' on the one hand to continue the simulation of all sweeps $1 \leq \ell < j$ and, on the other hand, to remember that the results of all sweeps after j are no longer relevant for the simulation. Hence, these sweeps are simulated by entering some dummy state d. In addition, a symbol d is printed on the tape that ensures that all later sweeps in T' are eventually performed in the dummy state d as well. The formal construction is as follows.

Let $T = \langle Q, \Sigma, \Delta, q_0, \triangleleft, \delta, F \rangle$ be a k-NIUFST with $|Q| = n$. The state set Q' of an equivalent $\lceil \frac{k}{i} \rceil$-NIUFST T' is defined as $Q' = \bigcup_{t=0}^{i} Q^t \times \{d\}^{i-t}$, where d is a new state not belonging to Q. The initial state q'_0 of T' is $(q_0, q_0, \ldots, q_0)$. The output alphabet is $\Delta' = \Delta \cup \{d\}$, where d is a new output

symbol not belonging to Δ. The transition function $\delta' \colon Q' \times (\Sigma \cup \Delta') \to 2^{Q' \times \Delta'}$ of T' is defined by a procedure that determines the successor states and the output symbols. For $x \in \Sigma \cup \Delta'$ we obtain

$$((r_1, r_2, \ldots, r_i), y_i) \in \delta'((s_1, s_2, \ldots, s_i), x)$$

as follows.

```
 1:  y₀ := x;     dummynow := false;
 2:  for t = 1 to i do
 3:      if dummynow or sₜ = d or yₜ₋₁ = d then
 4:          (rₜ, yₜ) := (d, d);
 5:      else
 6:          S := δ(sₜ, yₜ₋₁);
 7:          if S ≠ ∅ then
 8:              guess (rₜ, yₜ) ∈ S;
 9:          else
10:              (rₜ, yₜ) := (d, d);
11:              dummynow := true;
12:          end if
13:      end if
14:  end for
```

If T accepts (for the first time) at the end of a sweep j, the NIUFST T' can simulate at least these j sweeps of T successfully. So, any state $(r_1, r_2, \ldots, r_i) \in Q'$ with an $r_t \in F$ is accepting for T'. This works well if i divides k, since in this case any of the i sweeps simulated in T' corresponds to an original sweep in T. If i does not divide k, then in the last, the $\lceil \frac{k}{i} \rceil$th sweep of T' only the first $k - \lfloor \frac{k}{i} \rfloor \cdot i$ simulated sweeps of T have to be considered, since the remaining sweeps simulated do not exist in T. However, since all words accepted by T are accepted after at most k sweeps, these additional sweeps can never lead to an erroneous acceptance. For the number of states in T' we have, for $n \geq 2$, $|Q'| = \sum_{t=0}^{i} n^t = \frac{n^{i+1}-1}{n-1} \leq \frac{n^{i+1}}{n/2} = 2n^i$.

If T is deterministic, the set S in line 6 is either empty or a singleton. So, the guess in the former case boils down to select the sole element deterministically. That is, the construction preserves determinism. □

The sweep reduction presented in Theorem 3.1 can directly be used to transform constant sweep bounded NIUFSTs into equivalent NFAs.

Theorem 3.2. Let $n, k > 0$ be integers. Every n-state k-NIUFST can be converted to an equivalent NFA with at most $2n^k$ states.

Proof:
Given an n-state k-NIUFST T over the input alphabet Σ, by Theorem 3.1 we can obtain an equivalent $2n^k$-state 1-NIUFST T'. The difference between a 1-NIUFST and an NFA is that the former reads the endmarker, whereas the latter does not read an endmarker. Hence, T' can be converted to an equivalent

NFA T'' as follows. On input Σ, the transition functions of T' and T'' are defined identically. The accepting states of T'' are defined to be those states $(s_1, s_2, \ldots, s_k)$ such that there is a component in the k-tuple $\delta'((s_1, s_2, \ldots, s_k), \lhd)$ that is an accepting state in T. $\qquad\square$

We obtain a lower bound for the state blow-up in Theorem 3.2 by establishing a lower bound for the state cost of sweep reduction proved in Theorem 3.1. To this aim, for $n, k > 0$, let $L_{n,k}$ be the unary language

$$L_{n,k} = \{\, a^{c \cdot n^k} \mid c \geq 0 \,\}.$$

In [3], an n-state k-IUFST for $L_{n,k}$ is provided, whereas any equivalent DFA or NFA needs at least n^k states. By using $L_{n,k}$ as a witness language, we can show the following theorem.

Theorem 3.3. Let $n, k, i > 0$ be integers such that i divides k. There exists an n-state k-NIUFST such that any equivalent $\frac{k}{i}$-NIUFST cannot have less than $2^{\frac{-i}{k}} n^i$ states.

Proof:
Let T be an n-state k-NIUFST. Suppose by way of contradiction that we could always design an equivalent NIUFST T' with $\frac{k}{i}$ sweeps and $s < 2^{\frac{-i}{k}} n^i$ states. By using the result of Theorem 3.2, we can obtain from T' an equivalent NFA with $2s^{\frac{k}{i}} < 2 \cdot \left(2^{\frac{-i}{k}} n^i \right)^{\frac{k}{i}} = n^k$ states. By applying this approach in particular to the n-state k-IUFST recalled above for the language $L_{n,k}$, we could obtain an equivalent NFA having less than n^k states which is a contradiction. $\qquad\square$

The lower bound provided by Theorem 3.3 is general in the sense that the number of k sweeps can be reduced to any $\frac{k}{i}$ as long as i divides k. However, for the special case $i = k$, that is, reducing the sweeps to one only, we have a better lower bound as stated in the following corollary.

Corollary 3.4. For any integers $n, k > 0$, there exists an n-state k-NIUFST which cannot be converted to an equivalent NFA with less than n^k states.

Proof:
The language $L_{n,k}$ is accepted by some n-state k-IUFST but any NFA for $L_{n,k}$ has at least n^k states.
$\qquad\square$

We conclude this section by discussing the state blow-up of turning constant sweep bounded NIUFSTs into DFAs, i.e., the cost of removing both nondeterminism and sweeps at once.

Theorem 3.5. Let $n, k > 0$ be integers. Every n-state k-NIUFST can be converted to an equivalent DFA with at most 2^{2n^k} states.

Proof:
The result follows by first converting, according to Theorem 3.2, the n-state k-NIUFST into an equivalent $2n^k$-state NFA which, in turn, is converted to an equivalent 2^{2n^k}-state DFA by the usual powerset construction (see, e.g., [20]). $\qquad\square$

A lower bound for the state blow-up in Theorem 3.5 can be proved by considering the following language for any $n, k > 1$:

$$E_{n,k} = \{\, ubv \mid u, v \in \{a, b\}^*, |v| = c \cdot n^k - 1 \text{ for } c > 0 \,\}.$$

Theorem 3.6. For any integers $n > 1$ and $k > 0$, there is an $(n+1)$-state k-NIUFST which cannot be converted to an equivalent DFA with less than 2^{n^k} states.

Proof:
First we construct an $(n+1)$-state k-NIUFST $T = \langle Q, \{a, b\}, \Delta, q_0, \triangleleft_0, \delta, F \rangle$ that accepts the language $E_{n,k}$. We set $Q = \{q_0, q_1, q_2, \ldots, q_n\}$, $\Delta = \{a, b, \sqcup, 1, 2, \ldots, n, \bar{n}, \triangleleft_0, \triangleleft_1, \ldots, \triangleleft_{k-1}\}$, and $F = \{q_n\}$.

Basically, T processes an input string as follows. In a first sweep, T reads and blanks the input in its initial state q_0, where it guesses on every input symbol b whether it separates the prefix u from the suffix v. If it does not find a b or never guesses in the affirmative, T reaches the right endmarker in state q_0 and halts rejecting. Otherwise, beginning with the b transducer T starts to check whether the remaining input length is a multiple of n by rewriting it as a sequence of consecutive blocks of the form $12 \cdots n$, followed by $\triangleleft_1$. If and only if T reaches the endmarker in state q_1 the test was positive and the computation continues.

(1) $\delta(q_0, x) = \{(q_0, \sqcup)\}$ if $x \in \{a, \sqcup\}$
(2) $\delta(q_0, b) = \{(q_0, \sqcup), (q_2, 1)\}$
(3) $\delta(q_i, x) = \{(q_{i+1}, i)\}$ if $x \in \{a, b, n\}$, for $1 \le i \le n-1$
(4) $\delta(q_n, x) = \{(q_1, n)\}$ if $x \in \{a, b, n\}$
(5) $\delta(q_1, \triangleleft_i) = \{(q_0, \triangleleft_{i+1}\}$, for $0 \le i \le k-2$

In the second sweep, after reaching the first 1 in state q_0, transducer T checks whether the length of the remaining input string is a multiple of n^2 by rewriting n consecutive blocks $12 \cdots n$ with the block $1^n \, 2^n \, \cdots \, (n-1)^n \, \bar{n}^{n-1}n$, followed by $\triangleleft_2$. If and only if T reaches the endmarker in state q_1 the test was positive and the computation continues.

(6) $\delta(q_0, 1) = \{(q_1, 1)\}$
(7) $\delta(q_i, x) = \{(q_i, i)\}$ if $x \in \{1, 2, \ldots, n-1, \bar{n}\}$, for $1 \le i \le n-1$
(8) $\delta(q_n, x) = \{(q_n, \bar{n})\}$ if $x \in \{1, 2, \ldots, n-1, \bar{n}\}$

In the third sweep, after reaching the first 1 in state q_0, transducer T checks whether the length of the remaining input string is a multiple of n^3 by rewriting n consecutive blocks $1^n \, 2^n \, \cdots \, (n-1)^n \, \bar{n}^{n-1}n$ with the block $1^{n^2} \, 2^{n^2} \, \cdots \, (n-1)^{n^2} \, \bar{n}^{n^2-1}n$, followed by $\triangleleft_3$. If and only if T reaches the endmarker in state q_1 the test was positive and the computation continues.

This behavior is iterated and, according to Transition (9), the input string is easily seen to be accepted after the kth sweep only, upon reading $\triangleleft_{k-1}$.

(9) $\delta(q_1, \triangleleft_{k-1}) = \{(q_n, \triangleleft_{k-1})\}$

It remains to be shown that language $E_{n,k}$ cannot be accepted by a DFA with less than 2^{n^k} states. Suppose by contradiction that there is a DFA A which accepts $E_{n,k}$ with less than 2^{n^k} states. Since there are 2^{n^k} different words of length n^k over alphabet $\{a, b\}$ and the DFA A has less than 2^{n^k} states, at least two of these words drive A into the same state. Say the words are $w_1 = s_1 s_2 \cdots s_{n^k}$ and $w_2 = t_1 t_2 \cdots t_{n^k}$. Since $w_1 \neq w_2$, there is some $1 \leq i \leq n^k$ such that $s_i \neq t_i$. Without loss of generality, we may assume $s_i = b$ and $t_i = a$.

Since A is in the same state after processing w_1 and w_2, it is in the same state after processing $w_1 a^{i-1}$ and $w_2 a^{i-1}$. We have $n^k \leq |w_1 a^{i-1}| = |w_2 a^{i-1}| = n^k + (i - 1) \leq 2n^k - 1$. More-over, $w_1 a^{i-1} = s_1 s_2 \cdots s_{i-1} b s_{i+1} \cdots s_{n^k} a^{i-1}$ and, thus, $w_1 a^{i-1}$ belongs to $E_{n,k}$. On the other hand, $w_2 a^{i-1} = t_1 t_2 \cdots t_{i-1} a t_{i+1} \cdots t_{n^k} a^{i-1}$ and, thus, $w_w a^{i-1}$ does not belong to $E_{n,k}$. This is a contra-diction to the assumption that A has less than 2^{n^k} states. □

3.2. Decidability questions

Since every k-NIUFST can be converted into an equivalent DFA by Theorem 3.5 it is clear that all decidable questions for DFAs are also decidable for k-NIUFSTs. It has been shown in [3] that for k-IUFSTs the questions of testing emptiness, universality, finiteness, infiniteness, inclusion, or equiv-alence are NL-complete. Here, we will investigate the computational complexity of these questions for k-NIUFSTs and will obtain that the questions of emptiness, finiteness, and infiniteness are NL-complete, whereas the questions of universality, inclusion, and equivalence turn out to be PSPACE-complete. So, for k-NIUFSTs the questions of testing emptiness, universality, finiteness, infiniteness, inclusion, or equivalence have the same computational complexity as for NFAs (see, e.g., [22, 23] and [24] for a survey).

Theorem 3.7. Let $k \geq 1$ be an integer. Then for k-NIUFSTs the problems of testing emptiness, finiteness, and infiniteness are *NL*-complete.

Proof:
First, we show that the problem of non-emptiness belongs to NL. Since NL is closed under com-plementation, emptiness belongs to NL as well. We describe a two-way nondeterministic Turing machine M which receives an encoding $\mathrm{cod}(A)$ of some k-NIUFST A on its read-only input tape and accepts the input if and only if A accepts a non-empty language while the space used on its working tape is bounded by $O(\mathrm{ld} \, |\mathrm{cod}(A)|)$. Then, the work space is bounded by $O(\mathrm{ld} \, n)$, with n being the maximum among the number of states in A, the size of the input alphabet of A, and the size of the output alphabet of A, since all parameters are part of the encoding of A on the input tape of M.

It is shown in Theorem 3.2 which is based on Theorem 3.1 that A can be converted into an equiv-alent NFA with at most $2n^k$ states. This construction basically simulates k sweeps in one sweep and keeps in parallel track of the k states occuring in each sweep.

The basic idea for the Turing machine M is to guess a word and to check "on the fly" whether it is accepted by A. To simulate the k sweeps of A on the guessed input we apply the construction described above. Thus, we have to keep track of the k current states which are obtained by basically applying the nondeterministic procedure described in the proof of Theorem 3.1. A close look on that procedure shows that it can be implemented using no more than $O(\mathrm{ld} \, n)$ tape cells. Moreover, it is

clear that each state along a sweep can be represented by $O(\operatorname{ld} n)$ tape cells. Hence, the current states of the k sweeps of A are altogether represented by $O(\operatorname{ld} n)$ tape cells.

Now, the Turing machine M guesses one input symbol a and updates all stored states of A. This behavior is iterated until either A halts or A halts after having guessed the endmarker. Then, M halts accepting if A accepts and halts rejecting in any other case. Thus, M decides the non-emptiness of A using an amount of tape cells which is at most logarithmic in the length of the input.

To show that the problem of testing infiniteness belongs to NL we basically use the construction for non-emptiness. Additionally, we count the length of the guessed word and accept only if the underlying k-NIUFST A would have accepted an input of length at least $2n^k$. If $L(A)$ is infinite, then A clearly accepts an input of length at least $2n^k$. On the other hand, if A accepts an input of length at least $2n^k$, then the equivalent NFA with at most $2n^k$ states according to Theorem 3.2 accepts an input of length at least $2n^k$ as well. But this means that there is an accepting computation in the NFA in which at least one state is entered twice which implies that an infinite language is accepted. Thus, it remains to be argued that the counting up to $2n^k$ can be realized in logarithmic space. However, we can implement on M's working tape a binary counter that counts up to $2n^k$. With the usual construction this needs at most $O(\operatorname{ld} 2n^k) = O(\operatorname{ld} n)$ tape cells. Altogether, we obtain that the problem of testing infiniteness belongs to NL. Since NL is closed under complementation the problem of testing finiteness belongs to NL as well.

The hardness results follow directly from the hardness results for NFAs, which are basically shown in [22], considering the fact that every NFA N with n states can be converted into an equivalent k-NIUFST N' simulating N and having the same n states plus an additional accepting state which is entered exactly when N' reads the endmarker at the end of the first sweep and the previous simulation of N ended up in an accepting state of N. Obviously, the latter construction can be realized in deterministic logarithmic space. □

Theorem 3.8. Let $k \geq 1$ be an integer. Then for k-NIUFSTs the problems of testing universality, inclusion, and equivalence are *PSPACE*-complete.

Proof:
To show that the problems are in PSPACE it suffices to show that the conversion of a k-NIUFST into an equivalent NFA is possible in $\mathsf{P} \subseteq \mathsf{PSPACE}$, since the corresponding problems for NFAs are known to be in PSPACE. We consider a deterministic Turing machine which receives an encoding $\operatorname{cod}(A)$ of a k-NIUFSTs A and denote with n the maximum among the number of states in A, the size of the input alphabet of A, and the size of the output alphabet of A. Hence, we have to show that the time is bounded by some polynomial in n.

To estimate the time complexity of the conversion procedure described in Theorem 3.1 and Theorem 3.2 we note that the most costly action is that a nondeterministic procedure which determines the successor states and the output symbol is performed for each combination from $Q' \times (\Sigma \cup \Delta')$. Since each run of the nondeterministic procedure can be realized with an amount of space that is logarithmic in n, each run can be realized by a deterministic procedure with an amount of time that is polynomial in n. Since $|Q' \times (\Sigma \cup \Delta')| \in O(n^k)$, we obtain that the time complexity of the conversion is in P.

The hardness results follow again directly from the hardness results for NFAs and the fact that every NFA N with n states can be converted in deterministic polynomial time into an equivalent k-NIUFST simulating N and having $n + 1$ states. □

4. An infinite sweep hierarchy

We now consider $s(n)$-NIUFSTs where $s(n)$ is a non-constant function. In [3] it is proved that $o(\operatorname{ld} n)$ sweep bounded IUFSTs accept regular languages only, and that such a logarithmic sweep lower bound is tight for nonregular acceptance. Then, a three-level proper language hierarchy is established, where $O(n)$ sweeps are better than $O(\sqrt{n})$ sweeps which, in turn, are better than $O(\operatorname{ld} n)$ sweeps for IUFST. Here, we extend the hierarchy to infinitely many levels for both IUFSTs and NIUFSTs.

Let $f : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function. Its inverse is defined as the function $f^{-1}(n) = \min\{\, m \in \mathbb{N} \mid f(m) \geq n \,\}$. To show an infinite hierarchy dependent on some resources, where the limits of the resources are given by some functions in the length of the input, it is often necessary to control the lengths of the input so that they depend on the limiting functions. Usually, this is done by requiring that the functions are *constructible* in a desired sense. The following notion of constructibility expresses the idea that the length of a word relative to the length of a prefix is determined by a function.

Definition 4.1. A non-decreasing computable function $f \colon \mathbb{N} \to \mathbb{N}$ with $f(n) \geq n$ is said to be *constructible* if there exists an $s(n)$-IUFST T with $s(n) \in O(f^{-1}(n))$ and an input alphabet $\Sigma \uplus \{a\}$, such that

$$L(T) \subseteq \{\, a^m v \mid m \geq 1, v \in \Sigma^*, |v| = f(m) \,\}$$

and such that, for all $m \geq 1$, there exists a word of the form $a^m v$ in $L(T)$. The $s(n)$-IUFST T is said to be a *constructor* for f.

In order to show that the class of functions that are constructible in this sense is sufficiently rich to witness an infinite dense hierarchy, we next show that it is closed under addition and multiplication.

Proposition 4.2. Let $f \colon \mathbb{N} \to \mathbb{N}$ and $g \colon \mathbb{N} \to \mathbb{N}$ be two constructible functions. Then the function $f + g$ is constructible as well.

Proof:
Let T_f be a constructor for f and T_g be a constructor for g. We may safely assume that the input alphabet of T_f is $\Sigma_f \uplus \{a\}$ and the input alphabet of T_g is $\Sigma_g \uplus \{a\}$, where Σ_f and Σ_g are disjoint. Now, the idea for designing a constructor T for $f + g$ is to have

$$L(T) \subseteq \{\, a^m v_f v_g \mid m \geq 1, \, v_f \in \Sigma_f^*, \, |v_f| = f(m), \, v_g \in \Sigma_g^*, \, |v_g| = g(m) \,\}.$$

The constructor T splits its input into two tracks. In any sweep, on the first track the constructor T_f is simulated, whereby the symbols from Σ_g, that is, the factor v_g is ignored. Similarly, on the second track the constructor T_g is simulated. If one of the simulations halts, only the other simulation is continued. Now, T accepts if and only if both simulations end accepting.

Clearly, the language $L(T)$ is a subset as desired. In addition, for any $m \geq 1$, there exists a word of the form $a^m v_f v_g$ in $L(T)$. On such a word, the constructor T has a sweep complexity which is in $O(\max\{f^{-1}(m + f(m)), g^{-1}(m + g(m))\})$. Since $f(n) \geq n$ and $g(n) \geq n$ we conclude that the sweep complexity is in $O(m)$. So, the sweep complexity $s(n)$ of T is $s(n) \in O((f+g)^{-1}(n))$, which proves the proposition. $\qquad\square$

Proposition 4.3. Let $f \colon \mathbb{N} \to \mathbb{N}$ and $g \colon \mathbb{N} \to \mathbb{N}$ be two constructible functions. Then the function $f \cdot g$ is constructible as well.

Proof:
Let T_f be a constructor for f and T_g be a constructor for g. As in the proof of Proposition 4.2, we may safely assume that the input alphabet of T_f is $\Sigma_f \cup \{a\}$ and the input alphabet of T_g is $\Sigma_g \cup \{a\}$, where Σ_f and Σ_g are disjoint. Now, the idea for designing a constructor T for $f \cdot g$ is to have

$$L(T) = \{ a^m x_1 v_1 x_2 v_2 \cdots x_{g(m)} v_{g(m)} \mid m \geq 1,\, a^m v_i \in L(T_f),\, a^m x_1 x_2 \cdots x_{g(m)} \in L(T_g) \}.$$

To this end, T splits its input into two tracks. In any sweep, on the first track the constructor T_f is simulated, whereby the symbols from Σ_g are ignored. Moreover, the computation of T_f is independently simulated on any maximal factor v_i of symbols from Σ_f. These simulations are all started in the state in which T_f enters the first symbol after the prefix a^m. On the second track the constructor T_g is simulated on symbols of Σ_g, whereby the symbols from Σ_f are ignored. If a simulation halts, only the other simulations are continued. Now, T accepts if and only if all simulations end accepting.

Since T can verify the correct form of the input in an initial sweep, the language $L(T)$ is a subset as desired. In addition, for any $m \geq 1$, there exists a word $a^m x_1 v_1 x_2 v_2 \cdots x_{g(m)} v_{g(m)}$ in $L(T)$. As before, on such a word, T has sweep complexity $O(\max\{f^{-1}(m + f(m)), g^{-1}(m + g(m))\})$. Since $f(n) \geq n$ and $g(n) \geq n$ we conclude that the sweep complexity is in $O(m)$. So, the sweep complexity $s(n)$ of T is $s(n) \in O((f \cdot g)^{-1}(n))$. $\qquad\square$

In [3] it is shown that the unary language $L_{\text{uexpo}} = \{ a^{2^k} \mid k \geq 0 \}$ is accepted by some $s(n)$-IUFST with $s(n) \in O(\operatorname{ld} n)$. The construction can straightforwardly be extended to show that the function $f(n) = 2^n$ is constructible.

Moreover, again from [3], we know that $L_{\text{eq}} = \{ u\$v \mid u \in \Sigma_1^*, v \in \Sigma_2^*, \text{ and } |u| = |v| \}$ is a language accepted by some $s(n)$-IUFST with $s(n) \in O(n)$, where Σ_1 is an alphabet not containing the symbol \$ and Σ_2 is an arbitrary alphabet. Even in this case, only a tiny modification shows that the identity function is constructible. These facts together with Proposition 4.3 yield, in particular, that the function $f(n) = n^x$ is constructible for all positive integers x.

In what follows, we will use the fact, proved in [3], that the copy language with center marker $\{ u\$u \mid u \in \{a, b\}^* \}$ is accepted by some $s(n)$-IUFST satisfying $s(n) \in O(n)$. The next theorem provides some language that separates the levels of the hierarchy.

Theorem 4.4. Let $f \colon \mathbb{N} \to \mathbb{N}$ be a constructible function, T_f be a constructor for f with input alphabet $\Sigma \cup \{a\}$, and b be a new symbol not belonging to $\Sigma \cup \{a\}$. Then language

$$L_f = \{ u\$uv \mid u \in \{a, b\}^*,\, v \in \Sigma^*,\, a^{2|u|+1}v \in L(T_f) \}$$

is accepted by some $s(n)$-IUFST with $s(n) \in O(f^{-1}(n))$.

Proof:

Since the suffix v of a word $w \in L_f$ must be the suffix of some word $a^{2|u|+1}v$ in $L(T_f)$, we have that $|v| = f(2|u|+1)$ and $|w| = 2|u| + 1 + f(2|u|+1)$. Since $s(n)$ is claimed to be of order $O(f^{-1}(n))$, an $s(n)$-IUFST accepting L_f may perform at least $O(2|u|+1)$ many sweeps.

An $s(n)$-IUFST T accepting L_f essentially combines in parallel the acceptors for the copy language with center marker and the language $L(T_f)$. To this end, T establishes two tracks in its output. On the first track T simulates an acceptor for the copy language $\{ u\$u \mid u \in \{a, b\}^* \}$, where the first symbol of Σ (i.e., the first symbol of v) acts as endmarker. In this way, the prefix $u\$u$ is verified. The result of the computation is written to the output track. This task takes $O(2|u|+1)$ sweeps. On the second track T simulates the constructor T_f, where all symbols up to the first symbol of Σ (i.e., all symbols of the prefix $u\$u$) are treated as input symbols a. In this way, T verifies that $|v| = f(2|u|+1)$. The result of the computation is written to the output track. This task takes $O(2|u|+1)$ sweeps.

Finally, T rejects whenever one of the above simulations ends rejecting. Instead, T accepts if it detects positive simulation results of the two tasks on the tracks.   □

To show that the witness language L_f of Theorem 4.4 is not accepted by any $s(n)$-NIUFST with $s(n) \in o(f^{-1}(n))$, we use Kolmogorov complexity and incompressibility arguments. General information on this technique can be found, for example, in the textbook [25, Ch. 7]. Let $w \in \{a, b\}^*$ be an arbitrary binary string. Its Kolmogorov complexity $C(w)$ is defined to be the minimal size of a binary program (Turing machine) describing w. The following key fact for using the incompressibility method is well known: there exist binary strings w of *any* length such that $|w| \leq C(w)$.

Theorem 4.5. Let $f\colon \mathbb{N} \to \mathbb{N}$ be a constructible function, T_f be a constructor for f with input alphabet $\Sigma \cup \{a\}$, and b be a new symbol not belonging to $\Sigma \cup \{a\}$. Then language

$$L_f = \{ u\$uv \mid u \in \{a, b\}^*, \, v \in \Sigma^*, \, a^{2|u|+1}v \in L(T_f) \}$$

cannot be accepted by any $s(n)$-NIUFST with $s(n) \in o(f^{-1}(n))$.

Proof:

Contrarily, let us assume that some $s(n)$-NIUFST $T = \langle Q, \Sigma \cup \{a, b\}, \Delta, q_0, \triangleleft, \delta, F \rangle$ with sweep complexity $s(n) \in o(f^{-1}(n))$ accepts L_f.

We choose a word $u \in \{a, b\}^*$ long enough such that $C(u) \geq |u|$. Then, we consider an accepting computation of T on $u\$uv$, and derive a contradiction by showing that u can be compressed via T. To this end, we describe a program P which reconstructs u from a description of T, the length $|u|$, and the sequence of the $o(f^{-1}(n))$ many states $q_1, q_2, \ldots, q_r$ entered along the accepting computation at that moments when T reads the first symbol after the $\$$ along its $o(f^{-1}(n))$ sweeps, n being the total length of the input.

Basically, the program P takes the length $|u|$ and enumerates the finitely many words $u'v'$ with $u' \in \{a, b\}^{|u|}$, $v' \in \Sigma^*$, and $|v'| = f(2|u|+1)$. Then, for each word in the list, it simulates by dovetailing all possible computations of T on $u'v'$ where, in particular, it simulates the $o(f^{-1}(n))$ successive partial sweeps of T on $u'v'$, where the ith sweep is started in state q_i for $1 \leq i \leq r$. If the simulation ends accepting, we know that $u\$u'v'$ belongs to L_f and, thus, $u' = u$.

Let us consider the Kolmogorov complexity of u. Let $|T|$ denote the constant size of the description of T, and $|P|$ denote the constant size of the program P itself. The binary description of the length $|u|$ takes $O(\mathrm{ld}(|u|))$ bits. Each state of T can be encoded by $O(\mathrm{ld}(|Q|))$ bits. So, we have

$$C(u) \in |P| + |T| + O(\mathrm{ld}(|u|) + o(f^{-1}(n)) \cdot O(\mathrm{ld}(|Q|)) = O(\mathrm{ld}(|u|)) + o(f^{-1}(n)).$$

Since $n = 2|u| + 1 + f(2|u| + 1)$ and $f(n) \geq n$, for all $n \geq 1$, we have $n \in \Theta(f(2|u| + 1))$. So, we can conclude that $C(u) \in O(\mathrm{ld}(|u|)) + o(|u|) = o(|u|)$. This contradicts our initial assumption $C(u) \geq |u|$, for u long enough. Therefore, T cannot accept L_f with sweep complexity $o(f^{-1}(n))$. $\quad\square$

We would like to remark that, due to our observation that all functions $f(n) = n^x$ are constructible for $x \geq 1$, it is an easy application of the above theorems to obtain the following infinite hierarchies with regard to the number of sweeps both in the deterministic and the nondeterministic case. Namely: *For every $x \geq 1$ we have that the set of all languages that are accepted by $s(n)$-IUFSTs ($s(n)$-NIUFSTs) with $s(n) \in O(n^{1/(x+1)})$ is properly included in the set of all languages that are accepted by $s(n)$-IUFSTs ($s(n)$-NIUFSTs) with $s(n) \in O(n^{1/x})$.*

Finally, we turn to an upper bound of the computational capacity of NIUFSTs, where we do not limit the number of sweeps at all. Yet, we quickly observe that an NIUFST is clearly a restricted version of a linear bounded automaton (see, e.g., [20]). So, any language accepted by an NIUFST is context-sensitive. However, we can also show the converse though the transducers are one-way devices only that, at a glance, cannot transmit information from right to left. The proof uses the simulation of linear bounded automata (LBA).

For a given LBA M, we denote its state set by Q where q_0 is the initial state, by T its tape alphabet containing the endmarkers $\triangleright$ and $\triangleleft$, and by $\Sigma \subset T \setminus \{\triangleright, \triangleleft\}$ its input alphabet. The set of accepting states is $F \subseteq Q$ and the transition function δ maps from $Q \times T$ to the subsets of $Q \times (T \cup \{-1, +1\})$. So, the LBA M either can rewrite the current tape cell or move its head to the left (-1) or to the right ($+1$). We may safely assume that an LBA never moves its head beyond the endmarkers, starts with its head on the left endmarker, always halts, and accepts only on the right endmarker by halting in an accepting state.

Theorem 4.6. Let L be a context-sensitive language. Then L is accepted by an NIUFST.

Proof:
Let L be accepted by some nondeterministic LBA $M = \langle Q, \Sigma, T, q_0, \triangleright, \triangleleft, \delta, F \rangle$. Since M is assumed to be halting, its computations are finite sequences of configurations passed through. We will construct an NIUFST $T = \langle Q', \Sigma, \Delta, p_0, \triangleleft, \delta', F' \rangle$ that simulates M such that these configurations are successively emitted one in each sweep. So, the number of sweeps taken by T is one more than the number of steps performed by M.

We set $Q' = Q \cup \hat{Q} \cup \{p_0, p_s, p_1, p_+\}$, where $\hat{Q} = \{\hat{q} \mid q \in Q\}$ is a disjoint copy of Q, and $F = \{p_+\}$. The basic idea is to represent a configuration of M by two tracks. The current tape content of M is written on the second track, while the only nonblank cell of the first track is the cell currently scanned by the input head and its content is the current state of M. However, since M has a left endmarker that can be visited an arbitrary number of times, but T does not have a left endmarker, the first symbol on T's tracks represents the left endmarker in addition.

So, we set $\Delta = ((Q \cup \{\sqcup\}) \times \{\triangleright\}) \times ((Q \cup \{\sqcup\}) \times T) \cup ((Q \cup \{\sqcup\}) \times T) \cup \Sigma \cup \{\triangleleft\}$.

The initial input of T is of the form $\Sigma^* \triangleleft$. Here we assume that the input is non-empty. The construction can straightforwardly be extended to handle an empty input word as well. During its first sweep, T splits this input into two tracks. To this end, for $x \in \Sigma \cup \{\triangleleft\}$ we define:

(1) $\delta'(p_0, x) \quad = \quad \{(p_s, ((q_0, \triangleright), (\sqcup, x)))\}$

(2) $\delta'(p_s, x) \quad = \quad \{(p_s, (\sqcup, x))\}$

During successor sweeps, T uses its initial state p_0 to get close to the position of the cell with non-empty first track. In each step in state p_0 the NIUFST T guesses whether it reads the symbol to the left of the non-empty first track and whether M performs a left move (the states from $\hat{Q}$ are used for this purpose). Should T read the non-empty first track in state p_0 it has guessed that M does not perform a left move and simulates the stationary or right move with the help of the states from Q. Subsequently, T enters state p_1 to reach the right endmarker for a new sweep. So, for $x \in T \setminus \{\triangleright\}$ we define:

(3) $\delta'(p_0, ((\sqcup, \triangleright), (\sqcup, x))) \quad = \quad \{(p_0, ((\sqcup, \triangleright), (\sqcup, x)))\} \cup \{(\hat{q}, ((\sqcup, \triangleright), (q, x))) \mid q \in Q\}$

(4) $\qquad\qquad \delta'(p_0, (\sqcup, x)) \quad = \quad \{(p_0, (\sqcup, x))\} \cup \{(\hat{q}, (q, x)) \mid q \in Q\}$

Transitions (3) and (4) implement the guessing of a left move of M on the position coming next, where M enters state q. Once in some state $\hat{q}$ the guess is verified by the following Transition (5). If the guess was wrong, the transition is undefined and the computation halts rejecting.

(5) $\delta'(\hat{q}, (r, x)) \quad = \quad \{(p_1, (\sqcup, x))\}$ if $(q, -1) \in \delta(r, x)$

The next transitions implement the simulation of stationary and right moves of M when T reaches the non-empty first track in state p_0. Let $u \in T \setminus \{\triangleleft\}$.

(6) $\delta'(p_0, (q, u)) \quad = \quad \{(p_1, (r, z)) \mid (r, z) \in \delta(q, u)\} \cup \{(r, (\sqcup, u)) \mid (r, +1) \in \delta(q, u)\}$

(7) $\delta'(p, (\sqcup, x)) \quad = \quad \{(p_1, (p, x))\}$

Next, T uses state p_1 to proceed to the end of the track.

(8) $\delta'(p_1, (\sqcup, x)) \quad = \quad \{(p_1, (\sqcup, x))\}$

The next transition deals with steps that M may perform on its right endmarker in state p_0. It complements Transition (6).

(9) $\delta'(p_0, (q, \triangleleft)) \quad = \quad \{(p_1, (r, \triangleleft)) \mid (r, \triangleleft) \in \delta(q, \triangleleft)\} \cup \{(p_+, (\sqcup, \triangleleft)) \mid q \in F\}$

So, if and only if M halts accepting (which appears only on the right endmarker) T enters its sole accepting state q_+ at the end of a sweep. It remains to extend the definition of δ' for some cases concerning the first "double" symbol untreated so far.

(10) $\delta'(p_0, ((q, \triangleright), (\sqcup, x))) \quad = \quad \{(p_1, ((r, \triangleright), (\sqcup, x))) \mid (r, \triangleright) \in \delta(q, \triangleright)\}$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad \cup \{(p_1, ((\sqcup, \triangleright), (r, x))) \mid (r, +1) \in \delta(q, x)\}$

(11) $\delta'(p_0, ((\sqcup, \triangleright), (q, x))) \quad = \quad \{(p_1, ((\sqcup, \triangleright), (r, y))) \mid (r, y) \in \delta(q, x)\}$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad \cup \{(p_1, ((r, \triangleright), (\sqcup, x))) \mid (r, -1) \in \delta(q, x)\}$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad \cup \{(r, ((\sqcup, \triangleright), (\sqcup, x))) \mid (r, +1) \in \delta(q, x)\}$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \square$

5. Nondeterminism beats determinism on all levels

We now turn to compare the computational power of IUFSTs and NIUFSTs. Since for sweep bounds of order $o(\operatorname{ld} n)$ both variants accept regular languages only (see [3, 26]), it remains to consider sweep bounds beyond $o(\operatorname{ld} n)$. Here, we will show that there exist witness languages that are accepted by some nondeterministic $s(n)$-NIUFST with $s(n) \in O(\operatorname{ld} n)$, but cannot be accepted by any deterministic $s(n)$-IUFST with $s(n) \in o(n)$, thus separating determinism from nondeterminism for almost all levels of the sweep hierarchy.

For any integer $k \geq 1$, let $\operatorname{bin}_k \colon \{0, 1, 2, \ldots, 2^k - 1\} \to \{0, 1\}^k$ map any integer in the range from 0 to $2^k - 1$ to its binary representation of length k, starting from the left with the least significant digit and possibly completed with zeroes to the right. E.g., $\operatorname{bin}_4(5) = 1010$ and $\operatorname{bin}_4(12) = 0011$. We consider the language

$$D = \{\, a^k b^{2^k} \operatorname{bin}_k(0) u_0 \operatorname{bin}_k(1) u_1 \cdots \operatorname{bin}_k(2^k - 1) u_{2^k - 1} \operatorname{bin}_k(i) u_i \mid$$
$$k \geq 2, \ 1 \leq i \leq 2^k - 1, \ u_j \in \{a, b\}^k \text{ for all } 1 \leq j \leq 2^k - 1 \,\}.$$

Theorem 5.1. The language D can be accepted by an $s(n)$-NIUFST satisfying $s(n) \in O(\operatorname{ld} n)$.

Proof:
We sketch the construction of an $s(n)$-NIUFST T that accepts D with $s(n) \in O(\operatorname{ld} n)$. The basic idea of the construction is to use two output tracks. So, during its first sweep, T splits the input into two tracks, each one getting the original input. In addition, T verifies if the structure of the input is correct, that is, if the input is of the form $a^+ b^+ 0^+ \{a, b\}^+ (\{0, 1\}^+ \{a, b\}^+)^+ 1^+ \{a, b\}^+$ with at least two leading a's. If the form is incorrect, T rejects.

In subsequent sweeps, T behaves as follows. The original input on the first track is kept but the symbols can be marked, while on the second track the input is successively shifted to the right. More precisely, in any sweep the first unmarked symbol a in the leading a-block is marked. In the following b-block, every second unmarked symbol b is marked. In the further course of the sweep, the leftmost unmarked symbol in any $\{0, 1\}$-block as well as in any $\{a, b\}$-block is marked. On the second track, the input is shifted to the right by one symbol, whereby the last symbol is deleted and some blank symbol is added at the left.

Let $k \geq 2$ be the length of the leading a-block. When the last of its symbols is marked, T checks in the further course of the sweep whether in the following b-block exactly one symbol remains unmarked, and whether in all remaining blocks the last symbol is being marked. Only in this case the computation continues. In all other cases T halts rejecting.

From the construction so far, we derive that if the computation continues then all but the second block have the same length, namely, length k. Moreover, since in the second block every second unmarked symbol has been marked during a sweep and one symbol is left, the length of the block is 2^k.

Next, T continues to shift the content of the second track to the right until the $\{0, 1\}$-blocks are aligned with their neighboring $\{0, 1\}$-blocks (except for the last one). This takes another k sweeps. In the next sweep, T checks if the $\{0, 1\}$-block on the second track represents an integer that is one less

than the integer represented by the aligned block on the first track. This can be done by adding one on the fly and comparing the result with the content on the first track. Only if the check is successful, T continues. Otherwise, it halts rejecting. In the former case, we get that the sequence of $\{0, 1\}$-blocks represents the numbers from 0 to $2^k - 1$ in ascending order.

In the next sweep, T guesses the $\{0, 1\}$-block that has to match the rightmost $\{0, 1\}$-block and marks it appropriately. Finally, this block together with its following $\{a, b\}$-block is symbolwise compared with the last $\{0, 1\}$-block together with its following $\{a, b\}$-block in another $2k$ sweeps. To this end, we note that T can detect that the last block follows when it scans a $\{0, 1\}$-block consisting of 1's only. For the comparison, the symbols can further be marked appropriately.

Now, T accepts only if the guessed $\{0, 1\}$-block together with its following $\{a, b\}$-block match the last $\{0, 1\}$- and $\{a, b\}$-block. Otherwise T rejects. The construction shows that for any word from D there is one accepting computation and that only words from D are accepted. So, T accepts D.

Altogether, T performs at most $1 + k + k + 1 + 2k \in O(k)$ sweeps. The length of the input is $k + 2^k + (2^k + 1) \cdot 2k = O(k2^k)$. Since $\mathrm{ld}(O(k2^k)) \in O(k)$, the NIUFST T obeys the sweep bound $s(n) \in O(\mathrm{ld}\, n)$.   □

To show that the witness language D is not accepted by any $s(n)$-IUFST with $s(n) \in o(n)$, we use again Kolmogorov complexity and incompressibility arguments.

Theorem 5.2. The language D cannot be accepted by any $s(n)$-IUFST satisfying $s(n) \in o(n)$.

Proof:
In contrast to the assertion, we assume that some $s(n)$-IUFST $T = \langle Q, \{a, b, 0, 1\}, \Delta, q_0, \lhd, \delta, F \rangle$ with $s(n) \in o(n)$ accepts D. We choose an integer $k \geq 2$ and a word $u \in \{a, b\}^*$ of length $k2^k$ satisfying $C(u) \geq |u|$. Now, u is split into 2^k factors $u_0, u_1, \ldots, u_{2^k-1}$ of length k. Then, we choose an arbitrary factor u_i and consider the accepting computation of T on

$$a^k b^{2^k} \mathrm{bin}_k(0) u_0 \, \mathrm{bin}_k(1) u_1 \cdots \mathrm{bin}_k(2^k - 1) u_{2^k-1} \, \mathrm{bin}_k(i) u_i.$$

We are going to show that u can be compressed via T.

A program P reconstructs u from a description of T, the number k, and the sequence of the $o(n)$ many states $q_1, q_2, \ldots, q_r$ in which T reads the first symbol of the suffix $\mathrm{bin}_k(i) u_i$ as follows. Since T is deterministic, this sequence of states is the same for any suffix $\mathrm{bin}_k(i) u_i$, $0 \leq i \leq 2^k - 1$. The program P takes the length k and performs, for all $\mathrm{bin}_k(i)$, $0 \leq i \leq 2^k - 1$, the following. It enumerates the words $v \in \{a, b\}^+$ of length k. For each v, it simulates $o(n)$ successive partial sweeps of T on $\mathrm{bin}_k(i) v$, where the jth sweep is started in state q_j, for $1 \leq j \leq r$. If the simulation ends accepting, we know that v is the ith factor u_i of u. In this way, all factors of u and, thus, u are determined.

Let us consider the Kolmogorov complexity of u. Let $|T|$ denote the constant size of the description of T, and $|P|$ denote the constant size of the program P itself. The binary description of k takes $O(\mathrm{ld}(|k|))$ bits. Each state of T can be encoded by $O(\mathrm{ld}(|Q|))$ bits. So, we have

$$C(u) \in |P| + |T| + O(\mathrm{ld}(k)) + o(n) \cdot O(\mathrm{ld}(|Q|)) = O(\mathrm{ld}(k)) + o(n).$$

The length n of the input is $k + 2^k + 2|u| + 2k$. Since $|u| = k2^k$, we have $n \in \Theta(k2^k)$. So, we can conclude that $C(u) \in O(\mathrm{ld}(k)) + o(|u|) = o(|u|)$. This contradicts our initial assumption $C(u) \geq |u|$. So, T cannot accept D with sweep complexity $o(n)$. $\qquad\square$

References

[1] Kutrib M, Malcher A, Mereghetti C, Palano B. Deterministic and Nondeterministic Iterated Uniform Finite-State Transducers: Computational and Descriptional Power. In: Computability in Europe (CiE 2020), volume 12098 of *LNCS*. Springer, 2020 pp. 87–99. doi.:10.1007/978-3-030-51466-2_8.

[2] Kutrib M, Malcher A, Mereghetti C, Palano B. Descriptional Complexity of Iterated Uniform Finite-State Transducers. In: Hospodár M, Jirásková G, Konstantinidis S (eds.), Descriptional Complexity of Formal Systems (DCFS 2019), volume 11612 of *LNCS*. Springer, 2019 pp. 223–234. doi:10.1007/978-3-030-23247-4_17.

[3] Kutrib M, Malcher A, Mereghetti C, Palano B. Descriptional Complexity of Iterated Uniform Finite-State Transducers. *Inf. Comput.*, 2022. **284**:104691. doi:10.1016/j.ic.2021.104691.

[4] Friburger N, Maurel D. Finite-State Transducer Cascades to Extract Named Entities in Texts. *Theor. Comput. Sci.*, 2004. **313**(1):93–104. doi:10.1016/j.tcs.2003.10.007.

[5] Ginzburg A. Algebraic Theory of Automata. Academic Press, 1968.

[6] Hartmanis J, Stearns RE. Algebraic Structure Theory of Sequential Machines. Prentice-Hall, 1966.

[7] Citrini C, Crespi-Reghizzi S, Mandrioli D. On Deterministic Multi-Pass Analysis. *SIAM J. Comput.*, 1986. **15**(3):668–693. doi:10.1137/0215049.

[8] Bordihn H, Fernau H, Holzer M, Manca V, Martín-Vide C. Iterated Sequential Transducers as Language Generating Devices. *Theor. Comput. Sci.*, 2006. **369**(1-3):67–81. doi:10.1016/j.tcs.2006.07.059.

[9] Manca V. On the Generative Power of Iterated Transductions. In: Words, Semigroups, and Transductions – Festschrift in Honor of Gabriel Thierrin. World Scientific, 2001 pp. 315–327. doi:10.1142/9789812810908_0024.

[10] Pierce A. Decision Problems on Iterated Length-Preserving Transducers. Bachelor's thesis, SCS Carnegie Mellon University, Pittsburgh, 2011.

[11] Bednárová Z, Geffert V, Mereghetti C, Palano B. The Size-Cost of Boolean Operations on Constant Height Deterministic Pushdown Automata. *Theoret. Comput. Sci.*, 2012. **449**:23–36. doi:10.1016/j.tcs.2012.05.009.

[12] Bertoni A, Mereghetti C, Palano B. Trace Monoids with Idempotent Generators and Measure-Only Quantum Automata. *Natural Computing*, 2010. **9**(2):383–395. doi:10.1007/s11047-009-9154-8.

[13] Bianchi MP, Mereghetti C, Palano B. Quantum Finite Automata: Advances on Bertoni's Ideas. *Theoret. Comput. Sci.*, 2017. **664**:39–53. doi:10.1016/j.tcs.2016.01.045.

[14] Bednárová Z, Geffert V, Mereghetti C, Palano B. Boolean Language Operations on Nondeterministic Automata with a Pushdown of Constant Height. In: Bulatov A, Shur A (eds.), Computer Science in Russia (CSR 2013), volume 7913 of *LNCS*. Springer, 2013 pp. 100–111. doi:10.1007/978-3-642-38536-0_9.

[15] Bednárová Z, Geffert V, Mereghetti C, Palano B. Boolean Language Operations on Nondeterministic Automata with a Pushdown of Constant Height. *J. Comput. Syst. Sci.*, 2017. **90**:99–114. doi:10.1016/j.jcss.2017.06.007.

[16] Jakobi S, Meckel K, Mereghetti C, Palano B. Queue Automata of Constant Length. In: Jürgensen H, Reis R (eds.), Descriptional Complexity of Formal Systems (DCFS 2013), volume 8031 of *LNCS*. Springer, 2013 pp. 124–135. doi:10.1007/978-3-642-39310-5_13.

[17] Holzer M, Kutrib M. Descriptional Complexity – An Introductory Survey. In: Martín-Vide C (ed.), Scientific Applications of Language Methods, pp. 1–58. Imperial College Press, 2010. doi:10.1142/9781848165458_0001.

[18] Mealy GH. A Method for Synthesizing Sequential Circuits. *Bell System Tech. J.*, 1955. **34**:1045–1079. doi:10.1002/j.1538-7305.1955.tb03788.x.

[19] Mereghetti C. Testing the Descriptional Power of Small Turing Machines on Nonregular Language Acceptance. *Int. J. Found. Comput. Sci.*, 2008. **19**:827–843. doi:10.1142/S012905410800598X.

[20] Hopcroft JE, Ullman JD. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.

[21] Malcher A, Mereghetti C, Palano B. Descriptional Complexity of Two-Way Pushdown Automata with Restricted Head Reversals. *Theoret. Comput. Sci.*, 2012. **449**:119–133. doi:10.1016/j.tcs.2012.04.007.

[22] Jones ND. Space-Bounded Reducibility among Combinatorial Problems. *J. Comput. System Sci.*, 1975. **11**:68–85. doi:10.1016/S0022-0000(75)80050-X.

[23] Garey MR, Johnson DS. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979. ISBN:0-7167-1044-7.

[24] Holzer M, Kutrib M. Descriptional and Computational Complexity of Finite Automata - A Survey. *Inf. Comput.*, 2011. **209**(3):456–470. doi:10.1016/j.ic.2010.11.013.

[25] Li M, Vitányi PMB. An Introduction to Kolmogorov Complexity and Its Applications. Springer, 3rd edition, 2008.

[26] Hartmanis J. Computational Complexity of One-Tape Turing Machine Computations. *J. ACM*, 1968. **15**(2):325–339. doi:10.1145/321450.321464.