Theoretical Computer Science 969 (2023) 114049

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Iterated uniform finite-state transducers on unary languages ☆,☆☆

Martin Kutrib [a], Andreas Malcher [a], Carlo Mereghetti [b,*], Beatrice Palano [b]

[a] Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
[b] Dipartimento di Informatica "Giovanni Degli Antoni", Università degli Studi di Milano, via Celoria 18, 20133 Milano, Italy

ARTICLE INFO

Article history:
Received 18 March 2022
Received in revised form 3 March 2023
Accepted 21 June 2023
Available online 26 June 2023
Communicated by E. Formenti

Keywords:
Iterated transducers
Descriptional complexity
Unary languages
Cellular automata

ABSTRACT

An *iterated uniform finite-state transducer* executes the same length-preserving transduction in iterative sweeps. The first sweep occurs on the input string, while any subsequent sweep works on the output of the previous one. All sweeps always start from the sole initial state. The device accepts upon halting in an accepting state at the end of a sweep.

We consider devices with *one-way* sweep motion and *two-way* sweep motion, i.e., sweeps are either from left to right only, or strictly alternate from left to right and from right to left. In addition, devices may work deterministically or nondeterministically.

We focus on iterated uniform finite-state transducers accepting *unary languages*, i.e., languages built over single-letter alphabets.

We show that any *unary regular language* can be accepted by a deterministic iterated uniform finite-state transducer with at most $\max\{2 \cdot \varrho, p\} + 1$ states, where ϱ and p are the greatest primes in the factorization of the, respectively, pre-periodic and periodic part of the language. Such a state cost cannot be improved by using two-way motion, and it turns out to greatly outperform in the worst case the state costs of equivalent classical models of finite-state automata.

Next, we give a characterization of classes of unary languages accepted by *non-constant* sweep-bounded iterated uniform finite-state transducers in terms of time-bounded one-way cellular automata. This characterization enables both to exhibit interesting families of unary nonregular languages accepted by iterated uniform finite-state transducers, and to prove the undecidability of several questions related to iterated uniform finite-state transducers accepting unary languages with an amount of sweeps that is at least logarithmic.

1. Introduction

The notion of an iterated uniform finite-state transducer (IUFST) has been introduced in [15]. Basically, it consists of a length-preserving finite-state transducer that works in iterative sweeps from left to right on its input tape. In the first

☆ This article belongs to Section C: Theory of natural computing, Edited by Lila Kari.

☆☆ A preliminary version of this work was presented at the *47th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, January 25–29, 2021, Bolzano-Bozen, Italy [17].

* Corresponding author.
E-mail addresses: kutrib@informatik.uni-giessen.de (M. Kutrib), andreas.malcher@informatik.uni-giessen.de (A. Malcher), carlo.mereghetti@unimi.it (C. Mereghetti), palano@di.unimi.it (B. Palano).

https://doi.org/10.1016/j.tcs.2023.114049

sweep the input string is processed, while any further sweep operates on the output of the previous sweep. The model is uniform in that every sweep always starts from the sole initial state on the leftmost tape symbol, and operates the same transduction rules in each sweep. An input string is accepted whenever the transducer halts in an accepting state at the end of a sweep.

A theoretical investigation of IUFSTs is motivated by the fact that iterated or cascade transductions show up in several fields of computer science. For example, in the context of natural language processing, cascades of finite-state transducers are used in [6] to extract information from natural language texts. In compiler design, the lexical analysis is often done by a finite-state transducer whose output is subsequently processed by a pushdown transducer implementing the syntactical analysis. Again, from a theoretical perspective, the Krohn-Rhodes decomposition theorem states that every regular language can be represented as a cascade of several finite-state transducers with a simple algebraic structure [7,9]. Finally, cascades of deterministic pushdown transducers as language accepting devices have been studied in [4]. Yet, in [1,21], iterated finite-state transducers as *language generating* devices have been proposed. These devices start with some symbol in the initial state of the transducer, iteratively apply in multiple sweeps the transducer to the output produced so far, and eventually halt in an accepting state of the transducer after a last sweep. These iterated finite-state transducers, which in general are not length-preserving, are quite powerful since their nondeterministic version can generate non-recursive languages with only three states. Additionally, in the deterministic case, one state suffices to generate the class of D0L Lindenmayer systems and two states are sufficient to generate languages which are neither context-free nor in 0L. It might be worth noticing that [1,21] are the only contributions that introduce a notion of "uniformity" on iterated transductions, in the sense that always the same transducer is iteratively applied.

Deterministic and nondeterministic IUFSTs (the nondeterministic model being abbreviated as NIUFST) have been deeply studied in [18,19]. In case of a constant number of sweeps, IUFSTs and NIUFSTs characterize the class of regular languages. Therefore, in both papers, the comparison with standard automata models as well as a detailed investigation of the descriptional power of constant sweep-bounded IUFSTs and NIUFSTs have been carried on. Moreover, the state cost of removing nondeterminism and sweeps and of some language operations, as well as the computational complexity of decidability questions have been studied. In case of a non-constant number of sweeps, nonregular languages can be accepted as soon as an at least logarithmic number of sweeps is provided, and infinite proper language hierarchies depending on the sweep complexity are shown. Finally, non-constant sweep-bounded NIUFSTs are proved to be strictly more powerful than their deterministic variant if an at least logarithmic amount of sweeps is allowed.

Recently, in [16], IUFSTs and NIUFSTs have been enhanced with the possibility of *two-way motion*, implying a strict sweep alternation from left to right and from right to left, always starting from the sole initial state. The resulting devices abbreviate as 2IUFST and 2NIUFST, respectively. When performing a constant amount of sweeps, 2IUFSTs and 2NIUFSTs still characterize regular languages. Thus, the descriptional power of constant sweep-bounded 2IUFSTs and 2NIUFSTs has been investigated and compared with that of classical models of finite-state automata. In addition, natural questions in descriptional complexity such as the cost of removing nondeterminism and two-way motion from constant sweep-bounded 2IUFSTs and 2NIUFSTs have been analyzed. On the other hand, it has been proved in [20] that 2IUFSTs and 2NIUFSTs are able to accept even nonregular languages whenever provided with an at least logarithmic amount of sweeps. Moreover, for the same and at least logarithmic sweep complexity, it has been shown that 2NIUFSTs and NIUFSTs share the same accepting capability, while 2IUFSTs are strictly more powerful than IUFSTs.

It is worth remarking that, in [16], an initial investigation of iterated transducers working on *unary languages*, i.e., languages built over single-letter alphabets, has been proposed. Unary acceptance is a classical framework, e.g., to test the descriptional power of models of computation (see [2,27,26]). It turns out that, quite often, meaningful differences with the general alphabet case show up when investigating the unary world. A *particular* family of unary languages has been defined in [16], whose acceptance by iterated transduction cannot benefit – from the point of view of the number of states and sweeps – from the usage of two-way motion. On the other hand, it has been shown that IUFSTs accepting such a unary language family greatly outperform equivalent classical models of finite-state automata from a state complexity point of view.

In this paper, we continue our investigation on iterated transduction on unary languages *in all generality*, not focusing on particular unary languages only.

In Section 3, we tackle the problem of constructing succinct iterated transducers for *general unary regular languages*. We provide our construction step by step, starting from finite unary languages. We show that *finite* unary languages consisting of words with length up to ℓ can be accepted by an IUFST with $2 \cdot \varrho$ states, ϱ being the greatest prime in the factorization of ℓ. Next, we switch to unary *periodic* (or cyclic) languages, and prove that any n-periodic unary language can be accepted by an IUFST featuring p states, p being the greatest prime in the factorization of n. By combining these two results, we show that *any* unary regular language can be accepted by an IUFST with at most $\max\{2 \cdot \varrho, p\} + 1$ states, where ϱ and p are the greatest primes in the factorization of the, respectively, pre-periodic and periodic part of the language. We then show that these state costs cannot be improved by using two-way motion, and that they turn out to be drastically lower in the worst case than the state costs of equivalent classical models of finite-state automata.

In Section 4, we provide a tight connection between the non-constant sweep complexity of IUFSTs and the time complexity of one-way cellular automata (OCAs), when both models work on unary inputs. Precisely, we show that a unary language is accepted by an $n + r(n)$ time-bounded OCA if and only if it is accepted by an $r(n) + 1$ sweep-bounded IUFST. This characterization directly brings some interesting consequences. First of all, it enables to exhibit a rich class of unary nonreg-

ular languages accepted by $\Omega(\ln n)$ sweep-bounded IUFSTs, together with some closure properties. Furthermore, it implies the undecidability of some classical problems – such as emptiness, finiteness, infiniteness, inclusion, and equivalence – for $\Omega(\ln n)$ sweep-bounded IUFSTs accepting unary languages.

2. Definitions and preliminaries

2.1. Numbers and languages

We denote the set $\{0, 1, 2, ...\}$ of positive integers and zero by $\mathbb{N}$. The Fundamental Theorem of Arithmetic establishes that any integer $n > 1$ can be uniquely expressed as a product $n = p_1^{\alpha_1} \cdot \cdots \cdot p_s^{\alpha_s}$, where $p_1 < \cdots < p_s$ are primes, and $\alpha_1, \alpha_2, \ldots, \alpha_s$ are positive integers. For any $n > 1$, we let $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ be the ring of integers modulo n. By $\ln n$ we denote the logarithm of n to base e, and by $\mathrm{ld}\, n$ we denote the logarithm of n to base 2. More generally, we let $\log_b n$ be the logarithm of n to any given base $b > 1$. 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 $|w|$, and its reversal by w^R.

2.2. Iterated transduction

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 preceded and followed by endmarkers 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 should be noted that: (i) all passes always start from the same sole initial state, (ii) the transducer may alter the two endmarkers as well. 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 [23], to be iterated.

Formally, we define a *nondeterministic iterated uniform finite-state transducer* (NIUFST) as a system $T = \langle Q, \Sigma, \Delta, q_0, \triangleright, \triangleleft, \delta, F \rangle$, where Q is the 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, $\triangleright \in \Delta \setminus \Sigma$ and $\triangleleft \in \Delta \setminus \Sigma$ are the *left and right endmarkers*, respectively, $F \subseteq Q$ is the set of *accepting states*, and $\delta\colon Q \times (\Sigma \cup \Delta) \to 2^{Q \times \Delta}$ is the partial *transition function*. Notice that since the transduction 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 symbols from $\Sigma \cup \Delta$. We let $T(w)$ be the set of possible outputs produced by T in a complete sweep on input $w \in (\Sigma \cup \Delta)^*$.

We distinguish between *one-way* and *two-way* computations. In a one-way computation all sweeps are from left to right, whereas in a two-way computation the sweeps strictly alternate from left to right and from right to left. So, during a computation on input $w \in \Sigma^*$, the NIUFST T produces a sequence of words $w_1, \ldots, w_i, w_{i+1}, \ldots \in (\Sigma \cup \Delta)^*$:

- if the computation is *one-way*, then $w_1 \in T(\triangleright w \triangleleft)$ and $w_{i+1} \in T(w_i)$ for $i \geq 1$,
- if the computation is *two-way*, then $w_1 \in T(\triangleright w \triangleleft)$ and $w_{i+1} \in T(w_i)$ for even $i \geq 2$, while $w_{i+1}^R \in T(w_i^R)$ for odd $i \geq 1$.

We denote an iterated uniform finite-state transducer operating in one-way (resp., two-way) mode by NIUFST (resp., 2NIUFST).

An iterated uniform finite-state transducer is said to be *deterministic* (IUFST, 2IUFST) 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 $\delta\colon Q \times (\Sigma \cup \Delta) \to Q \times \Delta$.

We emphasize that both in the one-way and in the two-way iterated transduction, all sweeps always start from the sole initial state.

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 [24] for separation of these modes with respect to space complexity).

The NIUFST T *halts* whenever the transition function is undefined or T enters an accepting state at the end of a sweep. 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. Otherwise it is *rejected*. Indeed, 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.

2.3. Unary languages

A language L is said to be unary whenever it is built over a single-letter alphabet, i.e., $L \subseteq \Sigma^*$ and $|\Sigma| = 1$. In this case, we usually let $L \subseteq 0^*$ or $L \subseteq a^*$.

A unary language $L \subseteq 0^*$ is n-periodic (or n-cyclic) whenever there exists a set $\mathcal{R} \subseteq \mathbb{Z}_n$ such that $L = \{ 0^{c \cdot n + R} \mid c \geq 0 \text{ and } R \in \mathcal{R} \}$. We will always be assuming that n is the *minimal* value defining L. This is usually referred to as L being *properly* n-periodic. To emphasize periodicity and the set $\mathcal{R}$ of remainders modulo n, we will express L in the form $L_{n,\mathcal{R}}$.

Concerning state requirements to accept periodic languages on classical models of finite-state automata, the following is known from the literature:

- By using standard pumping arguments, it can be shown that n states are necessary and sufficient for DFAs and NFAs to accept $L_{n,\mathcal{R}}$. In particular, the transition digraph of the minimal DFA for $L_{n,\mathcal{R}}$ consists of a single cordless cycle of n states, with an initial state and final states settled according to $\mathcal{R}$.
- For n factorizing as $n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \cdots \cdot p_s^{\alpha_s}$, we have that $\sum_{i=1}^{s} p_i^{\alpha_i}$ states are necessary and sufficient for any two-way DFA and NFA [26, Thm. 9], and any isolated cut-point probabilistic finite automaton (PFA) [25, Thm. 2.8] to accept $L_{n,\mathcal{R}}$.

It is well known that any (infinite) unary regular language can be seen as the disjoint union of a finite language and an ultimately periodic language (very roughly speaking, membership in an ultimately periodic language becomes periodic from a certain point on). More precisely, a unary regular language can be defined by three parameters ℓ, n, and $\mathcal{R} \subseteq \mathbb{Z}_n$ as

$$L_{\ell,n,\mathcal{R}} = L_\ell \cup \{ 0^{\ell + c \cdot n + R} \mid c \geq 0 \text{ and } R \in \mathcal{R} \}.$$

The language L_ℓ, called the *pre-periodic* part of $L_{\ell,n,\mathcal{R}}$, is a (possibly empty) finite unary language containing strings of length less than or equal to ℓ. Instead, the set $\{ 0^{\ell + c \cdot n + R} \mid c \geq 0 \text{ and } R \in \mathcal{R} \}$ is called the *periodic part* of $L_{\ell,n,\mathcal{R}}$. As usual, we assume the parameters ℓ and n are the smallest possible defining $L_{\ell,n,\mathcal{R}}$. Notice that the transition digraph of the minimal DFA for $L_{\ell,n,\mathcal{R}}$ consists of an initial path of ℓ states joined to a cordless cycle of n states.

Clearly, the above recalled state lower bounds for two-way DFAs and NFAs, or isolated cut-point PFAs for $L_{n,\mathcal{R}}$ carry over to $L_{\ell,n,\mathcal{R}}$ as well. By simulation results in [2,27], we have that if $L_{\ell,n,\mathcal{R}}$ is accepted by a b-state NFA or two-way NFA, then we can assume $\ell = O(b^2)$ and $n = \mathrm{e}^{\Theta(\sqrt{b \cdot \ln b})}$.

2.4. An example of iterated transduction on unary languages

In [16], we designed an n-state k-2IUFST for the unary language $L_{n,k} = \{ a^{c \cdot n^k} \mid c \geq 0 \}$. Here, for the sake of completeness and to clarify the notion of acceptance by iterated transduction, we provide a first improvement of the construction of the 2IUFST in [16] by removing the two-way feature. As a further improvement on the state number, we will show in Subsection 3.1 how to accept $L_{n,k}$ on a constant sweep-bounded IUFST with less than n states.

Example 1. For any $n, k > 0$, the unary language $L_{n,k} = \{ a^{c \cdot n^k} \mid c \geq 0 \}$ is accepted by the n-state k-IUFST $T = \langle Q, \Sigma, \Delta, q_0, \rhd_0, \lhd_0, \delta, F \rangle$, where

$$\begin{aligned} Q &= \{q_0, q_1, \ldots, q_{n-1}\}, \\ \Sigma &= \{a\}, \\ \Delta &= \{a, \sqcup, \#, \rhd_0, \rhd_1, \ldots, \rhd_{k-1}, \lhd_0, \lhd_1, \ldots, \lhd_{k-1}\}, \\ F &= \{q_0\}. \end{aligned}$$

To explain the definition of the transition function δ, we first implement the behavior of T on the endmarkers. In general, the sweep number is identified by the indexes of the endmarkers. Since q_0 is the accepting state, the last step of all but the last sweep sends T into state q_1 to avoid to accept accidentally:

$$\begin{aligned} &(1)\ \ \delta(q_0, \rhd_i) = (q_0, \rhd_{i+1}) \quad \text{for } 0 \leq i \leq k-2, \\ &(2)\ \ \delta(q_0, \lhd_i) = (q_1, \lhd_{i+1}) \quad \text{for } 0 \leq i \leq k-2. \end{aligned}$$

In the first sweep, T verifies that the length of the input is divisible by n and rewrites the input as a sequence of consecutive blocks of the form $\#\sqcup^{n-1}$:

$$\begin{aligned} &(3)\ \ \delta(q_0, a) = (q_1, \#), \\ &(4)\ \ \delta(q_i, a) = (q_{i+1}, \sqcup) \quad \text{for } 1 \leq i \leq n-2, \\ &(5)\ \ \delta(q_{n-1}, a) = (q_0, \sqcup). \end{aligned}$$

In the following $k-1$ sweeps, T verifies that the number of # symbols is divisible by n and rewrites the input such that, from each n symbols #, one remains and $n-1$ are replaced by $\sqcup$:

$$(6)\quad \delta(q_i, \sqcup) = (q_i, \sqcup) \qquad \text{for } 0 \le i \le n-1,$$
$$(7)\quad \delta(q_i, \#) = (q_{i+1}, \sqcup) \quad \text{for } 0 \le i \le n-2,$$
$$(8)\quad \delta(q_{n-1}, \#) = (q_0, \#).$$

Finally, in the last sweep the endmarkers do not have to be rewritten by new symbols. Moreover, if the divisibility check is positive, the last step sends T into state q_0, and thus T halts accepting:

$$(9)\quad \delta(q_0, \rhd_{k-1}) = (q_0, \rhd_{k-1}).$$
$$(10)\quad \delta(q_0, \lhd_{k-1}) = (q_0, \lhd_{k-1}).$$

By construction, T accepts if all divisibility checks are positive. Let m be the length of an input from language $L_{n,k}$. After the first sweep all blocks have length n, thus, there are m/n symbols # in the output. After the ith sweep there are m/n^i symbols # in the output. So, after the kth sweep it is verified the length of the input is a multiple of n^k. On the other hand, the sole accepting state q_0 is never entered at the end of the first $k-1$ sweeps. So, accepting is only possible after the last sweep. To see that no input of incorrect length is accepted it is sufficient to look at the states reached at the end of a sweep. If it turns out that the length of the input is not divisible by n (first sweep) or the number of # symbols is not divisible by n (remaining sweeps), the sweep ends in some state unequal to q_0 on the endmarker. However, the transition function is undefined for such situations and T halts in a non-accepting state. ■

3. Iterated transduction and unary regular languages

We already gave hints in [16], of the higher descriptional power of unary iterated transducers over classical models of finite-state automata by focusing on a *particular* family of unary periodic languages. Namely, $\Pi(p)$ being the product of all primes not exceeding a given prime p, we introduced the unary regular language $L_{\Pi(p)} = \{ 0^{c \cdot \Pi(p)} \mid c \ge 0 \}$. We designed an IUFST for $L_{\Pi(p)}$, featuring p states and $(p/\ln p)$ sweeps whereas any equivalent DFA or NFA needs at least $\Pi(p) \sim e^p$ states, and any 2DFA, 2NFA or isolated cut-point PFA needs at least $p^2/\ln p$ states.

In this paper, we are going to tackle in a more systematic way the general problem of constructing small-size iterated transducers for *any* given unary regular language. It should be mentioned that here we focus on the number of states as complexity measure disregarding, for example, the number of output symbols. To consider the descriptional complexity in its entirety, the sizes of all parameters in the definition of iterated transducers have to be combined.

For a better understanding, we will show our construction step by step, dealing one at a time with meaningful families of unary regular languages. We will start with unary *periodic* languages, pass through unary *finite* languages, and finally get to designing iterated transducers for *general* unary regular languages. Among others, the constructions we are going to present will lead to improving from a state viewpoint the device proposed in Example 1 for the unary language $L_{n,k}$.

3.1. Unary periodic languages

For reader's ease of mind, we start by considering a particular family of unary periodic languages, which is actually a generalization of $L_{\Pi(p)}$ languages above recalled. For any $n \ge 1$, we let

$$L_n = \{ 0^{c \cdot n} \mid c \ge 0 \}.$$

Theorem 2. *Let $n \ge 2$ factorize as $n = p_1^{\alpha_1} \cdot \cdots \cdot p_s^{\alpha_s}$, with $\alpha_i > 0$. The language L_n can be accepted by a p_s-state r-IUFST with $r = \sum_{i=1}^{s} \alpha_i$ sweeps.*

Proof. To accept L_n, we design the IUFST $T = \langle Q, \Sigma, \Delta, q_0, \rhd, \lhd, \delta, F \rangle$, where

$$\begin{aligned} Q &= \{q_0, q_1, \ldots, q_{p_s-1}\}, \\ \Sigma &= \{0\}, \\ \Delta &= \{\rhd, 1, 2, \ldots, s\} \cup \{ \lhd_j^i \mid 1 \le i \le s \text{ and } 1 \le j \le \alpha_i \} \\ &\qquad \cup \{ *_j^i, \sqcup_j^i \mid 1 \le i \le s \text{ and } 1 \le j \le \alpha_i - 1 \}, \\ F &= \{q_0\}. \end{aligned}$$

Informally, the computation of T runs through s consecutive **phases**. During the ith **phase**, for $1 \le i \le s$, the IUFST T checks whether or not the length of the input string is divisible by $p_i^{\alpha_i}$; assume for the moment $\alpha_i > 1$. Along this **phase**, α_i sweeps are performed. During the jth sweep, for $1 \le j \le \alpha_i$, the divisibility of the input length by ${p_i}^j$ is checked. The input string is accepted if and only if at the end of the s **phases**, all the $\sum_{i=1}^{s} \alpha_i$ many sweeps witness input length divisibility by prime powers, as explained.

Let us define the transition function δ by first modeling the behavior of T on the endmarkers. Since q_0 is the accepting state, the last step from q_0 of all but the last sweep of the last **phase** sends T into the non-accepting state q_1, to avoid premature incorrect acceptance. Whereas, if the divisibility check is positive at the end of the last sweep of the last **phase**, the last step from q_0 keeps T in q_0, and so T may halt and accept:

$$
\begin{aligned}
&(1) && \delta(q_0, \rhd) = (q_0, \rhd),\\
&(2) && \delta(q_0, \lhd) = (q_1, \lhd_1^1),\\
&(3) && \delta(q_0, \lhd_j^i) = (q_1, \lhd_{j+1}^i) && \text{for } 1 \le i \le s-1 \text{ and } 1 \le j \le \alpha_i - 1,\\
&(4) && \delta(q_0, \lhd_{\alpha_i}^i) = (q_1, \lhd_1^{i+1}) && \text{for } 1 \le i \le s-1,\\
&(5) && \delta(q_0, \lhd_j^s) = (q_1, \lhd_{j+1}^s) && \text{for } 1 \le j \le \alpha_s - 2,\\
&(6) && \delta(q_0, \lhd_{\alpha_s-1}^s) = (q_0, \lhd_{\alpha_s}^s).
\end{aligned}
$$

For $1 \le i \le s$, in the first sweep of the ith **phase**, during which phase the divisibility of the input length by $p_i^{\alpha_i}$ is to be checked, T verifies that the input length is divisible by p_i, while rewriting the input as a sequence of consecutive blocks of the form $(\sqcup_1^i)^{p_i-1} *_1^i$. This is achieved by (7)–(8) below. It is worth noticing that at the end of the $(i-1)$st **phase** $(2 \le i \le s)$ every input symbol has been replaced by the symbol '$i-1$'. The detailed transitions implementing this latter symbol replacement are defined in (12)–(14) below.

$$
\begin{aligned}
&(7) && \delta(q_k, i-1) = (q_{k+1}, \sqcup_1^i) && \text{for } 0 \le k \le p_i - 2,\\
&(8) && \delta(q_{p_i-1}, i-1) = (q_0, *_1^i).
\end{aligned}
$$

In each of the following $\alpha_i - 2$ sweeps of the ith **phase**, T verifies whether or not the number of '$*_j^i$' symbols, for $1 \le j \le \alpha_i - 2$, is divisible by p_i. While doing this, T rewrites the tape so that, for each group of p_i many symbols '$*_j^i$' encountered, the last is replaced by the symbol '$*_{j+1}^i$', while each of the previous $p_i - 1$ many symbols '$*_j^i$' is replaced by the symbol $\sqcup_{j+1}^i$. Moreover, all symbols $\sqcup_j^i$ are replaced by the symbol $\sqcup_{j+1}^i$. So, the last sweep of this ith **phase** is easily seen to globally check the divisibility of the length of the input string by $p_i^{\alpha_i}$, while replacing each tape symbol by the symbol 'i':

$$
\begin{aligned}
&(9) && \delta(q_k, \sqcup_j^i) = (q_k, \sqcup_{j+1}^i) && \text{for } 0 \le k \le p_i - 1,\\
&(10) && \delta(q_k, *_j^i) = (q_{k+1}, \sqcup_{j+1}^i) && \text{for } 0 \le k \le p_i - 2,\\
&(11) && \delta(q_{p_i-1}, *_j^i) = (q_0, *_{j+1}^i),\\
&(12) && \delta(q_k, \sqcup_{\alpha_i-1}^i) = (q_k, i) && \text{for } 0 \le k \le p_i - 1,\\
&(13) && \delta(q_k, *_{\alpha_i-1}^i) = (q_{k+1}, i) && \text{for } 0 \le k \le p_i - 2,\\
&(14) && \delta(q_{p_i-1}, *_{\alpha_i-1}^i) = (q_0, i).
\end{aligned}
$$

If $\alpha_i = 1$, i.e. the ith **phase** consists of a single sweep, the instructions for δ are those here provided for the last sweep of the ith **phase** for the case $\alpha_i > 1$, the only difference being the input symbols which are now '$i-1$'.

It is not hard to verify that the IUFST T accepts if and only if the length of the input string is divisible by every $p_i^{\alpha_i}$, i.e., if and only if the input string belongs to L_n. Indeed, T features p_s states and $\sum_{i=1}^{s} \alpha_i$ sweeps. □

It may be worth noticing that, applying the construction in Theorem 2 for the target language $L_{\Pi(p)}$ (a particular case of L_n by letting $n = \Pi(p)$) would return an IUFST T which is identical to that originally proposed in [16]. This is basically due to the fact that each **phase** in the resulting T collapses to a single sweep, since each exponent in the prime factorization of $\Pi(p)$ is clearly 1.

The minimality – in terms of number of states – of the IUFST for L_n designed in Theorem 2 is provided in the following theorem:

Theorem 3. *Let $n \ge 2$ factorize as $n = p_1^{\alpha_1} \cdot \cdots \cdot p_s^{\alpha_s}$ with $\alpha_i > 0$, and let $k \ge 1$ be an integer. Then any k-IUFST accepting L_n must use at least p_s states.*

Proof. In [15, Thm. 3], it is proved that, for any given prime p, the language $L_p = \{ 0^{c \cdot p} \mid c \ge 0 \}$ cannot be accepted by any k-IUFST with less than p states. The proof goes by contradiction, assuming the existence of a k-IUFST T accepting L_p with less than p states. A string $0^{c \cdot p} \in L_p$, for c large enough, is then taken. By a fooling argument, it is shown that T accepts $0^{c \cdot p}$ if and only if it accepts the string $0^{c \cdot p - \alpha}$ as well, α being a product of numbers all strictly less than p. This contradicts the fact that $0^{c \cdot p - \alpha}$ cannot belong to L_p, whence the result.

This approach here adapts to the language L_n as follows. We suppose, by contradiction, the existence of a k-IUFST T for L_n with less than p_s states. Now, we take the string $0^{c \cdot n} \in L_n$, for c large enough, as a fooling string. As above, it turns

out that T accepts $0^{c\cdot n}$ if and only if it accepts the string $0^{c\cdot n-\alpha}$, where α is a product of numbers all strictly less than p_s. Clearly, $0^{c\cdot n-\alpha}$ cannot belong to L_n, whence a contradiction and the claimed result. □

Let us now briefly account on comparing the size of iterated transducers, and that of classical automata on accepting the language L_n, with $n = p_1^{\alpha_1} \cdot \cdots \cdot p_s^{\alpha_s}$:

(i) As addressed in Section 2.3, n states are necessary and sufficient for DFAs and NFAs to accept L_n, while $\sum_{i=1}^{s} p_i^{\alpha_i}$ states are necessary and sufficient for 2DFAs, 2NFAs and isolated cut-point PFAs.
(ii) By Theorem 2 and Theorem 3, we have that p_s states are necessary and sufficient for constant sweep-bounded IUFSTs.

We quickly address the state economy of accepting the language L_n by two-way iterated transduction:

- By combining the ideas employed in the construction of the 2IUFST provided in [16] for $L_{n,k}$ and those in the construction proposed in Theorem 2 above for the language L_n, with a little formal work one may obtain a 2IUFST accepting L_n with p_s states and $\sum_{i=1}^{s} \alpha_i$ sweeps.
- Again, the pumping argument in Theorem 3 can be adapted to show that p_s is the minimum amount of states to accept L_n on constant sweep-bounded 2IUFSTs.

As a final remark, we emphasize a state improvement on accepting the unary regular language $L_{n,k} = \{ a^{c\cdot n^k} \mid c \geq 0 \}$, for which an n-state k-IUFST is designed in Example 1. By noticing that $L_{n,k} = L_{n^k}$, a direct application of Theorem 2 leads to an IUFST accepting $L_{n,k}$ with p_s states and $k \cdot \sum_{i=1}^{s} \alpha_i$ sweeps, p_s being the greatest prime in the factorization of n.

3.1.1. Sweep reduction

Concerning the number of sweeps to accept L_n on constant sweep-bounded IUFSTs with p_s states, i.e. the minimum possible amount of states by Theorem 3, our construction in Theorem 2 can actually be improved to return p_s-state IUFSTs which in some cases exhibit less than $\sum_{i=1}^{s} \alpha_i$ sweeps. Let us explain the key idea:

Let $n = p_1^{\alpha_1} \cdot \cdots \cdot p_s^{\alpha_s}$. Consider the **phase** in which the divisibility of the input length by $p_i^{\alpha_i}$ is checked by using α_i sweeps and involving p_i of the p_s total states. Let $\gamma_i = \max\{ h \in \mathbb{N} \mid p_i^h \leq p_s \} = \lfloor \log_{p_i} p_s \rfloor \geq 1$. Clearly, $\alpha_i = \gamma_i \cdot \lfloor \alpha_i/\gamma_i \rfloor + (\alpha_i \bmod \gamma_i)$. It is not hard to see that:

(i) By performing $\lfloor \alpha_i/\gamma_i \rfloor$ sweeps involving $p_i^{\gamma_i}$ of the p_s total states, we can check whether or not the length of the input string is a multiple of $p_i^{\gamma_i \cdot \lfloor \alpha_i/\gamma_i \rfloor}$.
(ii) Subsequently, by a single final sweep that involves $p_i^{(\alpha_i \bmod \gamma_i)}$ of p_s total states, we can check whether or not the length of the input string is a multiple of $p_i^{\gamma_i \cdot \lfloor \alpha_i/\gamma_i \rfloor} \cdot p_i^{(\alpha_i \bmod \gamma_i)} = p_i^{\alpha_i}$. This final sweep is avoided in case $\alpha_i \bmod \gamma_i = 0$.

By using this enhancement in our construction algorithm, we still obtain constant sweep-bounded IUFSTs with p_s states for the language L_n, but now the number of sweeps can be evaluated with more precision. To this aim, for $1 \leq i \leq s$, we let:

$$\phi_i = \left\lfloor \frac{\alpha_i}{\gamma_i} \right\rfloor + \sigma_i, \quad \text{where } \sigma_i = \begin{cases} 1 & \text{if } \alpha_i \bmod \gamma_i \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Then, it is easy to see that the number of sweeps allowed by the enhanced construction is actually $\sum_{i=1}^{s} \phi_i$. Clearly, we have $s \leq \sum_{i=1}^{s} \phi_i \leq \sum_{i=1}^{s} \alpha_i$.

As a matter of fact, different sweep reduction strategies can be designed and possibly combined. For instance, suppose a set of indexes $S \subseteq \{1, \ldots, s-1\}$ in the prime factorization of n can be found, satisfying $\prod_{j \in S} p_j^{\alpha_j} = P < p_s$. Then, a single sweep involving P of the p_s total states suffices to check the divisibility of the length of the input string by P. This enables to save $\sum_{j \in S} \alpha_j - 1$ sweeps.

Let us now move on to a slightly different version of L_n. Precisely, for any $n \geq 1$ and a fixed remainder $R \in \mathbb{Z}_n \setminus \{0\}$, we let

$$L_{n,R} = \{ 0^{c\cdot n+R} \mid c \geq 0 \}.$$

The next theorem shows that this modification of L_n does not increase the state and sweep complexity of acceptance on IUFSTs.

Theorem 4. *Let $n \geq 2$ factorize as $n = p_1^{\alpha_1} \cdot \cdots \cdot p_s^{\alpha_s}$ with $\alpha_i > 0$. The language $L_{n,R}$ can be accepted by a p_s-state r-IUFST with $r = \sum_{i=1}^{s} \alpha_i$ sweeps.*

Proof. To simplify our reasoning, we begin by considering the particular case where $n = p^\alpha$, for a prime p and a positive integer α. Thus, we consider the language $L_{p^\alpha,R} = \{ 0^{c\cdot p^\alpha + R} \mid c \geq 0 \}$, with $R \in \mathbb{Z}_{p^\alpha} \setminus \{0\}$.

In this case, we can construct a p-state α-IUFST T consisting of a single **phase** (recall from the proof of Theorem 2, a **phase** consists of a sequence of sweeps) where T checks whether the input length modulo p^α yields R. We make this single **phase** work as in Theorem 2, but now at every sweep T stores in its states a digit of the representation in base p of $R < p^\alpha$. More precisely: let $r_\alpha r_{\alpha-1} \cdots r_1$ be the representation of R in base p, with r_1 being the least significant digit. According to Theorem 2, the set of states of T is $\{q_0, q_1, \ldots, q_{p-1}\}$ and, for $1 \leq i \leq \alpha$, the digit r_i will be represented by the state q_{r_i}, which will be entered at the end of the ith sweep if and only if the input string belongs to $L_{p^\alpha,R}$.

To show this, consider the string $0^m \in L_{p^\alpha,R}$ and let q_{x_i} be the state reached by T on the ith sweep before reading the right endmarker. We are going to show that $q_{x_i} = q_{r_i}$ by induction on i. For $i = 1$, the property follows trivially. Otherwise, according to the construction in Theorem 2, it is not hard to see that, before reading the right endmarker on the ith sweep, T will represent in its states the number $m_i \mod p$ where, for a given k, we have

$$\begin{aligned} m_i &= \frac{m}{p^{i-1}} - \sum_{j=1}^{i-1} \frac{r_j}{p^{i-j}} = \frac{1}{p^{i-1}} \left(k \cdot p^i + \sum_{j=1}^{i} r_j \cdot p^{j-1} \right) - \sum_{j=1}^{i-1} \frac{r_j}{p^{i-j}} \\ &= k \cdot p + \sum_{j=1}^{i} \frac{r_j}{p^{i-j}} - \sum_{j=1}^{i-1} \frac{r_j}{p^{i-j}} = k \cdot p + r_i. \end{aligned}$$

So, $x_i = m_i \mod p = r_i$, whence the result follows. This property enables us to define the transition function δ of T on the right endmarkers only in those situations where the sequence $r_\alpha r_{\alpha-1} \cdots r_1$ of remainders correctly shows up along the α sweeps, eventually accepting the input string. In all the other situations, we let δ undefined, thus leading to rejection.

Let us now consider the general case, where $n = p_1^{\alpha_1} \cdot \cdots \cdot p_s^{\alpha_s}$. For every $1 \leq i \leq s$, we let $R_i = R \mod p_i^{\alpha_i}$. The Chinese Remainder Theorem states that R is the only non-negative integer less than n satisfying the system of modular equations $\{R_i = m \mod p_i^{\alpha_i}\}_{1 \leq i \leq s}$, with m being the unknown. All the other solutions are of the form $m = c \cdot n + R$, for any $c > 0$. So, to check whether an input string 0^m belongs to $L_{n,R}$, it suffices to check whether $m \mod p_i^{\alpha_i} = R_i$, for every $1 \leq i \leq s$. From a set theoretical point of view, this implies that the language $L_{n,R}$ can be expressed as

$$L_{n,R} = \bigcap_{i=1}^{s} L_{p_i^{\alpha_i},R_i}$$

where each of the languages involved in the intersection has already been dealt with in the first part of this proof. Therefore, we can construct an IUFST with p_s states and $\sum_{i=1}^{s} \alpha_i$ sweeps implementing the following recognition algorithm:

```
1: input(0^m);
2: for i = 1 to s do
3:     if 0^m ∉ L_{p_i^{α_i},R_i} then
4:         REJECT BY UNDEFINED δ;
5:     end if
6: end for
7: ACCEPT.
```

The test $0^m \notin L_{p_i^{\alpha_i},R_i}$ at line 3 is performed according to the single **phase** algorithm for the language $L_{p^\alpha,R}$ outlined at the beginning of this proof. However, for $i > 1$, the test actually takes as input the string $(i-1)^m$ which is the output of the previous **phase**, as described in the proof of Theorem 2. □

Finally, we come to tackle the acceptance of a general unary n-periodic language $L_{n,\mathcal{R}}$, for a fixed set $\mathcal{R} \subset \mathbb{Z}_n$:

$$L_{n,\mathcal{R}} = \{ 0^{c\cdot n + R} \mid c \geq 0 \text{ and } R \in \mathcal{R} \}.$$

Theorem 5. *Let $n \geq 2$ factorize as $n = p_1^{\alpha_1} \cdot \cdots \cdot p_s^{\alpha_s}$, with $\alpha_i > 0$. The language $L_{n,\mathcal{R}}$ can be accepted by a p_s-state r-IUFST with $r = \sum_{i=1}^{s} \alpha_i$ sweeps.*

Proof. Let the set of remainders be $\mathcal{R} = \{R^{(1)}, \ldots, R^{(h)}\}$. As in the proof of Theorem 4, for every $1 \leq t \leq h$, we let $R_i^{(t)} = R^{(t)} \mod p_i^{\alpha_i}$ for $1 \leq i \leq s$. Thus

$$L_{n,\mathcal{R}} = \bigcup_{t=1}^{h} \left(\bigcap_{i=1}^{s} L_{p_i^{\alpha_i},R_i^{(t)}} \right).$$

This formula suggests a direct construction of an IUFST for $L_{n,\mathcal{R}}$ with p_s states and $h \cdot \sum_{i=1}^{s} \alpha_i$ sweeps, which activates, one after the other, the h iterated transducers for the languages $\bigcap_{i=1}^{s} L_{p_i^{\alpha_i},R_i^{(t)}}$. Each one of these h iterated transducers is designed in the proof of Theorem 4. However, we are able to exhibit an alternative construction featuring only i **phases** (one per each $p_i^{\alpha_i}$), globally resulting in $\sum_{i=1}^{s} \alpha_i$ sweeps. The ith **phase** aims to reveal $x_i = m \mod p_i^{\alpha_i}$, where m is as usual the length of the input string.

We say that x_i is *compatible* with the remainder $R^{(t)} \in \mathcal{R}$ whenever $x_i = R_i^{(t)}$. Moreover, we let $X_i = \{ R^{(t)} \in \mathcal{R} \mid x_i \text{ is compatible with } R^{(t)} \}$. For the sake of simplicity, we start by considering the case where all α_i's are 1. The general case of α_i's being greater than 0 will be addressed at the end of the proof.

The key idea is to use right endmarker symbols to store the subsets of the set $\mathcal{R}$ of remainders which are compatible with x_i's singled out **phase** after **phase**. (Notice that, since we are assuming $\alpha_i = 1$, a **phase** here consists of a single sweep.) More precisely, the right endmarker symbols will be of the form $\lhd_i^C$, for $1 \le i \le s$ and a nonempty set $C \subseteq \mathcal{R}$ of compatible remainders. We set the initial right endmarker to $\lhd_0^{C_0}$, with $C_0 = \mathcal{R}$. Now, let $C_{i-1} \subseteq \mathcal{R}$ be the superscript of the right endmarker before the beginning of the ith **phase**. At the end of the ith **phase**, after singling out x_i, the superscript of the right endmarker will be updated as $C_i = C_{i-1} \cap X_i$. By iterating this processing, at the end of the sth **phase** we will be able to decide whether or not an input string 0^m belongs to $L_{n,\mathcal{R}}$ by focusing on the cardinality of C_s. In fact, if $C_s = \{R^{(t)}\}$ for a fixed $1 \le t \le h$, then $m \mod n = R^{(t)}$ and hence $0^m \in L_{n,\mathcal{R}}$. Otherwise, we have that $C_s = \emptyset$ and clearly $0^m \notin L_{n,\mathcal{R}}$.

Let us now consider the case where α_i's can be greater than 1. The form of right endmarker symbols modifies as $\lhd_{i,j}^C$ for $1 \le i \le s$ and $1 \le j \le \alpha_i$. By considering the construction in Theorem 4, it is not hard to understand the usage of the new subscript j for accounting the digits of the representation in base p_i of x_i. The updating of the superscript of the right endmarker, denoting the set of remainders compatible with the x_i's so far computed, cannot be done once at the end of every **phase** only. In fact, along the ith **phase**, the transducer cannot store every digit of the representation in base p_i of x_i. So, the updating of the superscript of the right endmarker must be done at each one of the α_i many sweeps the ith **phase** consists of. Without going into technical details, this can be suitably hardwired in the definition of the transitions function δ. □

We conclude by observing that, again from Theorem 3, one may obtain that any constant sweep-bounded IUFST accepting $L_{n,\mathcal{R}}$ must use at least p_s states. Moreover, reductions on the number of sweeps may possibly be obtained as previously explained in Subsection 3.1.1.

As recalled in Section 2.3, we remark that n states are necessary and sufficient for DFAs and NFAs to accept $L_{n,\mathcal{R}}$, while $\sum_{i=1}^{s} p_i^{\alpha_i}$ states are necessary and sufficient for two-way DFAs, NFAs, and isolated cut-point PFAs.[1]

3.2. Unary finite languages

For any positive integer ℓ, let L_ℓ be any unary language whose longest word has length ℓ. In what follows, we assume $\ell \ge 2$. The case $\ell = 1$ can be trivially managed by a 2-state DFA seen as a transducer.

Theorem 6. *Let $\ell \ge 2$ factorize as $\ell = \varrho_1^{\beta_1} \cdot \cdots \cdot \varrho_r^{\beta_r}$, with $\beta_i > 0$. (Notice that here ϱ_i stands for the ith prime in the factorization of ℓ). The language L_ℓ can be accepted by a $(2 \cdot \varrho_r)$-state t-IUFST with $t = \sum_{i=1}^{r} \beta_i$ sweeps.*

Proof. We let $L_\ell \subseteq a^*$. We define the set of states of an IUFST T for L_ℓ as $Q = \{ q_i \mid 0 \le i \le \varrho_r - 1 \} \cup \{ \bar{q}_i \mid 0 \le i \le \varrho_r - 1 \}$, barred states being accepting. Again, for the sake of simplicity, we consider the case where all β_i's equal 1, and hence the total number of sweeps will be r. On input a^m, the IUFST T acts as follows:

- The first sweep counts m modulo ϱ_1 by using states q_i's, while writing blocks of the form "$0\,1 \cdots \varrho_1 - 1$". Clearly, the last symbol written before the right endmarker is $m \mod \varrho_1$.
- The second sweep counts m modulo ϱ_2 by using states q_i's, but this time the output consists of two tracks: in the first track, the output of the previous sweep is copied, while in the second track blocks of the form "$0\,1 \cdots \varrho_2 - 1$" are written. Again, the last symbol of the second track written before the right endmarker is $m \mod \varrho_2$.
- This dynamic is repeated for $r - 1$ sweeps, the ith sweep adding a new track where the last symbol written before the right endmarker is $m \mod \varrho_i$.
- Along the rth sweep, we copy all the $r - 1$ tracks previously written and we add a last track where we count modulo ϱ_r while writing blocks of the form "$0\,1 \cdots \varrho_r - 1$". However, this time we use both states q_i's and $\bar{q}_i$'s, as we are now going to explain. Notice that, along this last sweep, the output symbol written after consuming an input prefix of length κ turns out to be $(\kappa \mod \varrho_1, \ldots, \kappa \mod \varrho_r)$ which, by the Chinese Remainder Theorem, uniquely identifies κ up to congruence modulo ℓ. So, if $a^\kappa \in L_\ell$, the transition function leads to $\bar{q}_{\kappa \mod \varrho_r}$, otherwise we reach $q_{\kappa \mod \varrho_r}$. This way of defining the transition function δ enables to reach the right endmarker on a barred state if a^m belongs to L_ℓ. So, to decide acceptance, it suffices that δ on the right endmarker let the transducer in the current state. Instead, to reject

[1] Recall that $L_{n,\mathcal{R}}$ is *properly* n-periodic, i.e., n is the minimal value defining $L_{n,\mathcal{R}}$.

any word of length exceeding ℓ, we leave δ undefined as soon as the written symbol is the sequence of remainders representing $\ell + 1$.

Let us now consider the case where β_i's can be greater than 1. Without going into technical details, we have that a **phase** of β_i sweeps (instead of a single sweep, as previously explained) is used for the prime factor $\varrho_i^{\beta_i}$. The remainder m mod $\varrho_i^{\beta_i}$ is now represented in base ϱ_i by β_i digits. This multiple-digit remainder encoding can be suitably managed by the transition function, so that the key idea used in the last sweep does not change. □

We conclude by observing that ℓ states are easily seen to be necessary on classical models of finite-state automata to accept L_ℓ. Again, a pumping argument as in Theorem 3 may be used to show that not less than ϱ_r states are needed on constant sweep-bounded IUFSTs and 2IUFSTs.

3.3. General unary regular languages

Finally, let us put things together. As addressed in Section 2, a general (infinite) unary regular language consists of the disjoint union of a (possibly empty) finite pre-periodic language and an ultimately periodic language. More precisely, it can be defined by three parameters ℓ, n, and $\mathcal{R} \subseteq \mathbb{Z}_n$ as

$$L_{\ell,n,\mathcal{R}} = L_\ell \cup \{\, 0^{\ell+c\cdot n+R} \mid c \geq 0 \text{ and } R \in \mathcal{R} \,\},$$

where, with a slight abuse of notation with respect to the definition of L_ℓ stipulated in Section 3.2, we now intend L_ℓ as a finite unary language that is accepted by an ℓ-state DFA. So, differently from Section 3.2, here and from now on L_ℓ does not necessarily contain the unary word of length ℓ. In the following theorem, we consider $\ell, n \geq 2$. Otherwise, we have languages that can be trivially dealt with by our constructions in this paper.

Theorem 7. *Let $\ell, n \geq 2$ factorize, respectively, as $n = p_1^{\alpha_1} \cdot \cdots \cdot p_s^{\alpha_s}$ with $\alpha_i > 0$, and $\ell = \varrho_1^{\beta_1} \cdot \cdots \cdot \varrho_r^{\beta_r}$ with $\beta_i > 0$. The unary language $L_{\ell,n,\mathcal{R}}$ can be accepted by an x-state t-IUFST, where $x = \max\{2 \cdot \varrho_r, p_s\} + \xi$ with $\xi = 0$ if $\varrho_r < p_s$, 1 otherwise, and $t = \sum_{i=1}^{s} \alpha_i + \sum_{i=1}^{r} \beta_i$.*

Proof. We use in cascade the transducer, say T', provided in Theorem 6 and the transducer, say T'', designed in Theorem 5. However, T' is slightly modified to take into account that T'' must be activated only for inputs of length exceeding ℓ. To this aim, T':

(i) on input words of length less than or equal to ℓ not in L_ℓ, it now halts on the right endmarker by undefined δ,
(ii) it accepts words in L_ℓ by the original construction,
(iii) for words of length exceeding ℓ, it enters a "new" non-accepting state sweeping the rest of the input; such a new state can be q_{ϱ_r} whenever $\varrho_r < p_s$.

A further modification of T' is the fact that on the last sweep any input symbol is rewritten by 0, so that T'' can start working on a string of 0's, as assumed in Theorem 5. Even T'' needs a slight tuning. In fact, the remainders in $\mathcal{R}$ must be updated to consider the initial segment of length ℓ. Precisely, any $R^{(t)} \in \mathcal{R}$ will be replaced by $(R^{(t)} + \ell)$ mod n. The reader may easily obtain the claimed number of states and sweeps for the transducer described so far. □

Again, the pumping argument in Theorem 3 can be adapted to show that not less than $\max\{\varrho_r, p_s\}$ states can be used to accept $L_{\ell,n,\mathcal{R}}$ on constant sweep-bounded IUFSTs and 2IUFSTs. On the other hand, as recalled in Section 2.3, we have that $\ell + n$ states are necessary and sufficient for a DFA to accept $L_{\ell,n,\mathcal{R}}$, whereas not less than $\sum_{i=1}^{s} p_i^{\alpha_i}$ states on two-way DFAs and NFAs, and on isolated cut-point PFAs are needed. Yet, some sweeps may be saved in the IUFST proposed in Theorem 7 for $L_{\ell,n,\mathcal{R}}$, by using the techniques outlined in Subsection 3.1.1.

3.4. Some concluding remarks on unary iterated transduction

There are many different measures that have been considered in descriptional complexity from the very beginning. Some of them are related to the length of the description, that is roughly the length of some encoding of a system. Others are not, for instance, the well-studied number of nonterminals in context-free grammars. In [10], measures that are recursively related to length are called s-measures. So, it is distinguished between measuring sizes of systems and measuring *resources* of systems. Here, for our IUFSTs accepting unary regular languages, we focused on measuring the computational resources "number of states and sweeps", in order to get comparisons with classical finite-state automata.

Nevertheless, it would be an interesting research to study the size of IUFSTs with respect to an s-measure. From this point of view, in addition to the amount of states and sweeps, it is well worth pointing out the *cardinality of the output alphabet* Δ of our IUFST T for a general unary regular language $L_{\ell,n,\mathcal{R}}$, provided in Theorem 7. By adopting notations there stated, and following the sequence of constructions culminating in T, we get

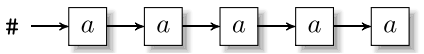

Fig. 1. Initial configuration of a unary one-way cellular automaton with input a^5.

$$|\Delta| \leq \left(2 + 2^{|\mathcal{R}|}\right) \cdot \sum_{i=1}^{s} \alpha_i + 4 - s + \prod_{i=1}^{r} Q_i.$$

We again emphasize that only the number of states and sweeps were at stake here, so reducing $|\Delta|$ is well worth pursuing, e.g., by combining the modules T consists of in a smarter way. More generally, it would be interesting to deepen tradeoffs among the amounts of states, sweeps, and output symbols in iterated transduction.

4. Beyond constant sweep complexity

In this section, we will discuss the case of IUFSTs working on a unary input and having a non-constant sweep complexity. Since for sweep bounds of order $o(\ln n)$ deterministic and nondeterministic IUFSTs in the one-way case as well as in the two-way case accept regular languages only (see [15,8,29]), it remains to consider sweep bounds of order $\Omega(\ln n)$.

4.1. Characterizing sweep complexity by one-way cellular automata time complexity

The purpose of this subsection is to establish a meaningful relation between the sweep complexity of a unary IUFST and the time complexity that is needed by a one-way cellular automaton beyond real time. In this way, the sweep complexity can be characterized by the time complexity of a massively parallel device.

In detail, a one-way cellular automaton (OCA) is a linear array of identical deterministic finite automata, called *cells*, where each cell except the leftmost one is connected to its left neighbor. The transition of a cell depends on its current state and the current states of its neighbor, where the leftmost cell receives information associated with a boundary symbol on its free input line. The cells work synchronously at discrete time steps. The input mode for OCAs is called parallel. One can suppose that all cells fetch their input symbol during a pre-initial step.

More formally, an OCA is a system $M = \langle S, \#, A, \delta, F\rangle$, where $S \neq \emptyset$ is the finite set of cell states, $\# \notin S$ is the permanent boundary symbol, $A \subseteq S$ is the input alphabet, $F \subseteq S$ is the set of accepting cell states and $\delta : (S \cup \{\#\}) \times S \to S$ is the local transition function. A *configuration* of an OCA at some time step $t \geq 0$ is a mapping $c_t : \{1, 2, \ldots, n\} \to S$, for $n \geq 1$, which maps the single cells to their current states. The computation starts at time 0 in a so-called *initial configuration*, which is defined by the given input $w = x_1 x_2 \cdots x_n \in A^+$ as $c_0(i) = x_i$, for $1 \leq i \leq n$. As an example, Fig. 1 depicts an OCA when starting its computation on the unary string a^5.

Let c_t, with $t \geq 0$, be a configuration. Its successor configuration c_{t+1} is defined as $c_{t+1}(i) = \delta(c_t(i-1), c_t(i))$ for $i \in \{2, 3, \ldots, n\}$ and $c_{t+1}(1) = \delta(\#, c_t(1))$. An input w is *accepted* by an OCA M if at some time step during the course of its computation the rightmost cell enters an accepting state.

Let $t : \mathbb{N} \to \mathbb{N}$ be a mapping. If all $w \in L(M)$ are accepted within at most $t(|w|)$ time steps, then M is said to be of time complexity t. Since in general OCAs do not halt, this implies also that all $w \notin L(M)$ are not accepted, that is, rejected at time $t(|w|)$. Clearly, the identity function n is the least time complexity for non-trivial computations. So, if $t(n) = n$ acceptance is said to be in *real time* (see, for example, [13] for further information about cellular automata as language acceptors).

The next lemma shows that a non-constant number of sweeps significantly increase the computational power of IUFSTs even in the unary case. In particular, they increase the computational power in the same way as adding the same amount of time to a real-time OCA.

Lemma 8. *Let $r : \mathbb{N} \to \mathbb{N}$ be a mapping and M be a unary* OCA *obeying the time complexity $n + r(n)$. Then an equivalent $(r(n) + 1)$-*IUFST *can effectively be constructed.*

Proof. Let $M = \langle S, \#, \{a\}, \delta_M, F_M\rangle$ be an OCA. By a straightforward modification we can always achieve that M does not accept before time step n and that the input state a is never entered again. Fig. 2 displays the time-space diagram of a computation of M.

One may easily see that all cells that did not receive any information from the left border are in the same state in any configuration. Moreover, these states can be computed as $a_1 = \delta_M(a, a)$, $a_2 = \delta_M(a_1, a_1)$, and so on. Now the basic idea of the construction of an equivalent IUFST $T = \langle Q, \{a\}, \Delta, q_0, \rhd, \lhd, \delta_T, F_T\rangle$ is as follows. In a first sweep, T reads the a's and emits the first diagonal of the space-time diagram that consists of states on which the left border may had an effect (the green diagonal in Fig. 2). To this end, T enters in every step a state that is actually a pair of states of M, that is, the state on the diagonal and the state to the right of the diagonal. In subsequent sweeps, T reads the states of a diagonal and emits the states of the next diagonal. To achieve this, it enters the states on the diagonal.

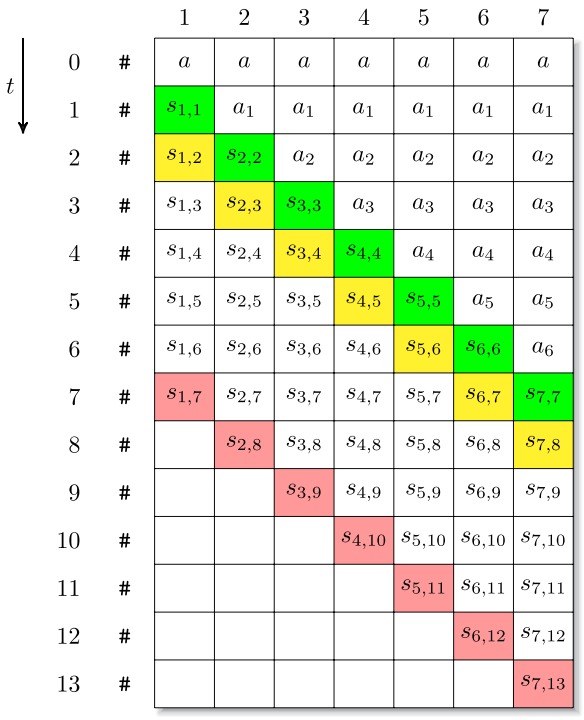

Fig. 2. Space-time diagram of a schematic computation of an OCA on input a^n with $n = 7$, that accepts at time $n + r(n) = 7 + 6$.

More formally, we set $Q = \{q_0, q_+\} \cup S \cup S^2$, $\Delta = \{\rhd, \lhd\} \cup S$, $F_T = \{q_+\}$, and specify δ_T as follows:

$$
\begin{aligned}
&(1) && \delta_T(q_0, \rhd) = (q_0, \rhd),\\
&(2) && \delta_T(q_0, a) = \Big(\big(\delta_M(\#, a), \delta_M(a, a)\big), \delta_M(\#, a)\Big),\\
&(3) && \delta_T((s, a'), a) = \Big(\big(\delta_M(s, a'), \delta_M(a', a')\big), \delta_M(s, a')\Big),\\
&(4) && \delta_T((s, a'), \lhd) = (q_0, \lhd) \quad \text{if } s \notin F_M,\\
&(5) && \delta_T((s, a'), \lhd) = (q_+, \lhd) \quad \text{if } s \in F_M,
\end{aligned}
$$

$$
\begin{aligned}
&(6) && \delta_T(q_0, s) = \big(\delta_M(\#, s), \delta_M(\#, s)\big),\\
&(7) && \delta_T(s', s) = \big(\delta_M(s', s), \delta_M(s', s)\big),\\
&(8) && \delta_T(s', \lhd) = (q_0, \lhd) \quad \text{if } s' \notin F_M,\\
&(9) && \delta_T(s', \lhd) = (q_+, \lhd) \quad \text{if } s' \in F_M.
\end{aligned}
$$

Finally, T accepts if it simulates an accepting state of M when reading the right endmarker. In this case, the rightmost cell of M has entered an accepting state and so M has accepted. If M never accepts then T will never accept. Thus, both devices accept the same language. Moreover, the state of the rightmost cell of M at time $n + r(n)$ is simulated at the end of the $(r(n) + 1)$st sweep of T. □

Now we turn to the converse simulation of Lemma 8.

Lemma 9. *Let T be an $s(n)$-IUFST with unary input alphabet. Then an equivalent OCA obeying the time complexity $n + s(n) - 1$ can effectively be constructed.*

Proof. Let $T = \langle Q, \{a\}, \Delta, q_0, \rhd, \lhd, \delta_T, F_T\rangle$ be an IUFST. By a straightforward modification we can always achieve that T rewrites the left endmarker only by itself which means that the left endmarker is left unchanged. So, we consider a fixed state $s_0 = \delta_T(q_0, \rhd)$ in which T starts to scan the tape symbol next to the left endmarker. Moreover, we may safely assume that T will never halt rejecting. This behavior can be obtained by adding a non-accepting dummy state in which the sweep continues. Then the right endmarker is labeled appropriately, which prevents accepting in subsequent sweeps. The underlying idea of the construction of an equivalent OCA $M = \langle S, \#, \{a\}, \delta_M, F_M\rangle$ is similar to that of the proof of Lemma 8, where the IUFST simulates the states on diagonals of the space-time diagram by sweeps. Now the OCA simulates the sweeps

on diagonals. More precisely, assume that T enters state p on emitting symbol x at tape position i in its kth sweep. Then the ith cell of the OCA to be constructed will enter state (p,x) in the configuration at time $i+k-1$. This behavior can be implemented since the next step of T is the application of $\delta_T(p,y)$ on tape position $i+1$ carrying symbol y. Then the $(i+1)$st cell of the OCA is in state (q,y) in the configuration at time $i+k-1$, for some $q \in Q$. So, a simulation of $\delta_T(p,y)$ by cell $i+1$ of the OCA is possible and this cell can enter the state $\delta_T(p,y)$ in the configuration at time $i+k$.

There is still a technical problem to be settled, caused by the fact that T may rewrite its right endmarker. Therefore, the symbol at the position of the right endmarker has to be simulated by the rightmost cell of the OCA. However, the cells of the OCA cannot know whether or not they are at the right border. So, all cells have to simulate a possible endmarker to their right, but only the simulation by the true rightmost cell plays a role for acceptance. More formally, let $E_R \subseteq \Delta$ be the set of symbols that may appear at the position of the right endmarker. We set $S = \{a\} \cup (Q \times \Delta \times E_R)$, $F_M = \{\, (p,x,e) \mid \delta_T(p,e) \in F_T \,\}$, and specify δ_M as follows:

$$
\begin{aligned}
&(1) && \delta_M(a,a) = a, \\
&(2) && \delta_M(\#,a) = (p',a',\lhd) && \text{where } (p',a') = \delta_T(s_0,a), \\
&(3) && \delta_M((p,x,e),a) = (p',a',\lhd) && \text{where } (p',a') = \delta_T(p,a), \\
&(4) && \delta_M(\#,(q,y,f)) = (p',a',e') && \text{where } (p',a') = \delta_T(s_0,y) \text{ and} \\
& && && (p'',e') = \delta_T(q,f) \text{ for some } p'' \in Q, \\
&(5) && \delta_M((p,x,e),(q,y,f)) = (p',a',e') && \text{where } (p',a') = \delta_T(p,y) \text{ and} \\
& && && (p'',e') = \delta_T(q,f) \text{ for some } p'' \in Q.
\end{aligned}
$$

Now, M accepts if and only if its rightmost cell n enters a state (p,x,e) such that $\delta_T(p,e) \in F_T$. By the construction idea this means that T enters state p on emitting symbol x at tape position n where e is the symbol on position $n+1$ of the endmarker. Since the next symbol read by T is e, we conclude that M accepts if and only if T accepts. Finally, if the nth cell of M accepts at time $t = n+k-1$ then T enters state p at tape position n where e is the symbol on position $n+1$ in sweep k. □

The previous lemmas prove the following characterization:

Theorem 10. *Let $r : \mathbb{N} \to \mathbb{N}$ be a mapping. A unary language is accepted by an* OCA *obeying the time complexity $n + r(n)$ if and only if it is accepted by an* IUFST *with sweep complexity $r(n)+1$.*

This characterization opens the door to a rich family of unary languages that can thus be accepted by IUFSTS. It is known that the family of languages accepted by OCAS obeying time complexity $(1+\varepsilon)n$, where ε is an arbitrarily small positive number, coincides with the reversals of languages accepted by two-way cellular automata in real time (see, for example, [13]). Since the reversal of a unary language is the language itself, all unary languages accepted by two-way cellular automata in real time are accepted by $(\varepsilon n+1)$-IUFSTS. This language family is very rich (see, e.g., [5,22,30,31]). Examples are:

- $\{\, a^{n^k} \mid n \geq 1 \,\}$, for all $k \geq 1$,
- $\{\, a^{k^n} \mid n \geq 1 \,\}$, for all $k \geq 1$,
- $\{\, a^{n!} \mid n \geq 1 \,\}$,
- $\{\, a^p \mid p \text{ is prime} \,\}$,
- $\{\, a^n \mid n \text{ is a Fibonacci number} \,\}$.

Moreover, the class of these languages is closed under several language operations applied to the functions that give the word lengths [5,12,22].

4.2. Reducing sweeps on unary languages

In this subsection, we show a general result on sweep reduction for unary languages accepted by $s(n)$-IUFSTS with $s(n) \in O(n)$. In detail, it is shown that the sweep complexity can be exponentially reduced at the price of accepting a modified language. However, the modification does not change the size, i.e., the number of words in the language. This feature will be essential for the undecidability results given in the next subsection. Our main result in this subsection is the following translational lemma:

Lemma 11. *Let a unary language L be accepted by an $s(n)$-*IUFST *with sweep complexity $s(n) \in O(n)$. Then, the language $L' = \{\, a^{2^m} \mid a^m \in L \,\}$ is accepted by an $s'(n)$-*IUFST *with $s'(n) \in O(\ln n)$.*

	0	1	2	3	4
0	$\triangleright$	a	a	a	$\triangleleft$
1	$\triangleright_1$	t_1	t_2	t_3	$\triangleleft_1$
2	$\triangleright_2$	t_4	t_5	t_6	$\triangleleft_2$
3	$\triangleright_3$	t_7	t_8	t_9	$\triangleleft_3$
4	$\triangleright_4$	t_{10}	t_{11}	t_{12}	$\triangleleft_4$
5	$\triangleright_5$	t_{13}	t_{14}	t_{15}	$\triangleleft_5$
6	$\triangleright_6$	t_{16}	t_{17}	t_{18}	$\triangleleft_6$

Fig. 3. Output protocol of an IUFST accepting *aaa* after six sweeps.

Proof. Let L be accepted by some $s(n)$-IUFST T. We assume that $s(n) \leq c \cdot n$ for some integer constant $c \geq 1$. By introducing additional states we may also assume that T halts only at the end of a sweep on the endmarker. The basic idea is to check whether the input length is 2^m for some $m \geq 1$, in parallel with checking whether T accepts on input a^m. To construct an $s'(n)$-IUFST T' accepting L', we divide each output tape cell into $c+1$ components, called tracks. Hence, we can speak of track 1, track 2, ..., track $c+1$. Additionally, each of the tracks track 2, track 3, ..., track $c+1$ is divided into two subtracks.

Now, track 1 will be used to check that the input length is a power of two by implementing the idea of iteratively dividing by two. To this end, in every sweep every odd unmarked a is suitably marked. The input length is 2^m if and only if in sweep $m+1$ the last a of the input is the only unmarked a. At this moment, it can be decided whether the input length is a power of two. Whenever T' finds that the input length is not a power of two, it halts non-accepting.

In addition to the above marking procedure, we can use another marking to tag in every sweep the first untagged a. Hence, after $m+1$ sweeps the first $m+1$ positions of the input are tagged and indicate together with the left endmarker exactly the positions in which T should be simulated on $\triangleright a^m \triangleleft$. For this simulation we have to take into account the fact that T' is uniform and starts every sweep in its initial state. Hence, we do not know that the exact position of the right endmarker is the position $m+1$. Thus, we use the first subtrack of track 2, track 3, ..., track $c+1$ to simulate the computation of T under the assumption that the initial input symbol was an a, whereas the second subtrack is used to simulate the computation of T under the assumption that the initial input symbol is the endmarker $\triangleleft$. Additionally, we enter some permanent state f_+ and emit it whenever T would enter an accepting state on the endmarker $\triangleleft$. Similarly, some permanent state f_- is entered and emitted whenever T would halt non-accepting on the endmarker $\triangleleft$.

Now, the dynamics of T' can be sketched as follows. In its first sweep, T' starts the computation in track 1 and performs the first division by two.

In the next c sweeps of T', c sweeps of T are consecutively simulated in tracks $2, 3, \ldots, c+1$. In the next sweep, the next division by two is performed in track 1 and in the next c sweeps another c sweeps of T are consecutively simulated in tracks $2, 3, \ldots, c+1$. This behavior is iterated until either there is some sweep working on track 1 that halts non-accepting or it is detected in sweep $(1+c)m+1$ while reading the right endmarker that the input length is 2^m. In the latter case, T' enters an accepting state if the simulation of T ended up in the state f_+ in some second subtrack at position $m+1$. Since in sweep $(1+c)m+1$ exactly the position $m+1$ is tagged, the states of T' can carry the information whether or not the state f_+ has been entered in some second subtrack at position $m+1$ to the right endmarker. Otherwise, T' halts in a non-accepting state.

By construction, clearly T' accepts an input a^{2^m} if and only if T halts accepting on $\triangleright a^m \triangleleft$. Moreover, we have that T' performs $(1+c)m+1$ sweeps on input length $n = 2^m$. Hence, $s'(n) = (1+c)(\mathrm{ld}\, n)+1$ belongs to $O(\ln n)$. □

To illustrate the construction in the proof of Lemma 11, we provide the following example.

Example 12. We consider the computation of an $s(n)$-IUFST T with $s(n) = 2n$ on input *aaa* whose output protocol is depicted in Fig. 3. Let us assume that the input is accepted after $s(3) = 6$ sweeps.

In Fig. 4 we depict the output protocol of an $s'(n)$-IUFST accepting a^8 after $s'(n) = (1+c)(\mathrm{ld}\, n)+1$, here, $s'(8) = 10$ sweeps according to the construction in the proof of Lemma 11. The number of the sweep performed by the $s'(n)$-IUFST is indicated in the first column. To avoid overloading the picture, for every sweep only the track in which changes are made is shown. The shown track is indicated in the second column. A marked input symbol a is denoted by a' and a tagged a' is denoted by $\underline{a'}$. The tracks 2 and 3 are divided into two subtracks where the upper component denotes the first subtrack and the lower component denotes the second subtrack. Finally, an output symbol given by the transition function of T but not relevant for the current example is denoted by '·'. ■

		0	1	2	3	4	5	6	7	8	9
0		$\rhd$	a	a	a	a	a	a	a	a	$\lhd$
1	1	$\rhd$	$\underline{a}'$	a	a'	a	a'	a	a'	a	$\lhd$
2	2	$\rhd_1$	t_1	t_2	t_3	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\lhd$
			$\cdot$	$\cdot$	$\cdot$	$\lhd_1$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	
3	3	$\rhd_2$	t_4	t_5	t_6	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\lhd$
			$\cdot$	$\cdot$	$\cdot$	$\lhd_2$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	
4	1	$\rhd$	$\underline{a}'$	$\underline{a}'$	a'	a	a'	a'	a'	a	$\lhd$
5	2	$\rhd_3$	t_7	t_8	t_9	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\lhd$
			$\cdot$	$\cdot$	$\cdot$	$\lhd_3$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	
6	3	$\rhd_4$	t_{10}	t_{11}	t_{12}	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\lhd$
			$\cdot$	$\cdot$	$\cdot$	$\lhd_4$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	
7	1	$\rhd$	$\underline{a}'$	$\underline{a}'$	$\underline{a}'$	a'	a'	a'	a'	a	$\lhd$
8	2	$\rhd_5$	t_{13}	t_{14}	t_{15}	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\lhd$
			$\cdot$	$\cdot$	$\cdot$	$\lhd_5$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	
9	3	$\rhd_6$	t_{16}	t_{17}	t_{18}	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\lhd$
			$\cdot$	$\cdot$	$\cdot$	f_+	$\cdot$	$\cdot$	$\cdot$	$\cdot$	
10	1	$\rhd$	$\underline{a}'$	$\underline{a}'$	$\underline{a}'$	$\underline{a}'$	a'	a'	a'	a'	$\lhd$

Fig. 4. Output protocol of an IUFST accepting a^8 after ten sweeps. Note that for every sweep only the track in which changes are made is depicted. The number of the track is indicated in the second column.

4.3. Undecidability results

Concerning decidability questions it is shown in [19] that all commonly studied decidability questions such as emptiness, finiteness, infiniteness, inclusion, and equivalence are undecidable for $s(n)$-IUFSTs with $s(n) \in \Omega(\ln n)$. Obviously, these undecidability results carry over to the stronger models with nondeterministic moves and/or two-way motion. Taking a look on the proof of these results it is clear that the underlying languages used in the proof are not unary. Thus, the question arises whether the above-mentioned decidability questions become decidable in case of IUFSTs working on unary inputs. For example, it is known for one-way multi-head finite automata (see, e.g., [11]) that all above-mentioned decidability questions are undecidable in general, but become decidable in case of unary languages. However, for IUFSTs we can show in what follows that the decidability questions for IUFSTs remain undecidable even if only unary inputs are considered.

The way to obtain these results is to first show that the questions of emptiness, finiteness, infiniteness, inclusion, and equivalence are undecidable for OCAs working in linear time and accepting a unary input. Using the results of Subsection 4.1 we can construct an equivalent $s(n)$-IUFST with $s(n) \in O(n)$. Using then the construction from Subsection 4.2 we can construct an $s'(n)$-IUFST with $s'(n) \in O(\ln n)$ and translate the undecidability results for unary linear-time OCAs to undecidability results for IUFSTs having a sweep complexity in $\Omega(\ln n)$.

Theorem 13. *For linear-time* OCA*s accepting unary languages, the problems emptiness, finiteness, infiniteness, inclusion, and equivalence are undecidable.*

Proof. For the undecidability results we will use the well-known fact that it is undecidable whether or not a deterministic counter machine having two counters that are initially zero and starting with empty input will eventually halt [28]. Thus, the basic idea is to construct a linear-time OCA that simulates a two-counter machine C on empty input and accepts some input if and only if C halts on empty input. The simulation is realized by using the states of the cells to encode the current values of both counters.

We define the language $L_C = \{\, a^n \mid C \text{ halts on empty input after } n \text{ steps} \,\}$ and first show that L_C is accepted by a real-time CA which is defined similarly as an OCA with the difference that every cell is connected with its left and right neighbor. It is known (see, e.g., [13]) that for any unary language accepted by a real-time CA an equivalent linear-time OCA can be constructed.

A real-time CA A accepting L_C simulates in two tracks two binary counters which are realized by storing the binary encoding of the current value of each counter in their cells. Basically, in each track each cell stores one bit, where the rightmost cell stores the least significant bit. Due to the finite neighborhood it is impossible to obtain configurations that are binary representations literally. Instead, carry-overs are sent from cell to cell until they can be processed. On the other hand, the test for zero requires to mark the cell carrying the most significant bit, and this mark may move. Hence, carry-overs are transported by sending information from right to left, and the cell carrying the most significant bit can be marked by sending information from left to right. In this way, the number of cells needed to store in real time the current values of the binary counters can suitably be increased and decreased if necessary. (See, e.g., [14] for a similar construction.) In the rightmost cell of A the simulation of C is started with both counters being zero. The simulation of both counters is suitably

updated and the current state of C is carried in the rightmost cell. In addition, in the first time step the leftmost cell starts a signal with maximum speed to the right. When this signal reaches the rightmost cell at the moment when a halting state of C is entered then A accepts; A rejects in all other cases. It is clear that A accepts L_C in real time and, moreover, $L(A)$ is not empty if and only if C halts.

Similarly, a real-time CA A' can be constructed that accepts

$$L'_C = \{ a^m \mid m \geq n \text{ and } C \text{ halts on empty input after } n \text{ steps} \}.$$

The basic modification is that the rightmost cell enters an accepting state if and only if a halting state of C is entered before or at the arrival of the right-moving signal. Then, we have that $L(A')$ is infinite if and only if C halts.

Since any real-time CA accepting a unary language can be converted to an equivalent linear-time OCA, we can construct linear-time OCAS T and T' accepting L_C and L'_C. Hence, $L(T)$ is not empty and $L(T')$ is infinite if and only if C halts. Since the halting problem on empty input is undecidable for counter machines, we therefore obtain that emptiness, finiteness, and infiniteness are undecidable for unary linear-time OCAS. The fact that the empty set is accepted by a linear-time OCA and the result that emptiness is undecidable for unary linear-time OCAS immediately imply the undecidability of inclusion and equivalence for unary linear-time OCAS. □

Using the results of Subsections 4.1 and 4.2 we are able to prove undecidability results for $s(n)$-IUFSTs accepting unary languages with $s(n) \in \Omega(\ln n)$.

Theorem 14. *Let $s(n) \in \Omega(\ln n)$. Then for $s(n)$-IUFSTs accepting unary languages, the problems emptiness, finiteness, infiniteness, inclusion, and equivalence are undecidable.*

Proof. First, we note that owing to a speed-up result for linear-time OCAS (see, e.g., [13]) we may always assume that linear-time OCAS work in time $t(n) = 2n$. Then, due to Lemma 8 we know that for every unary linear-time OCA accepting a language L with time complexity $2n$ an equivalent $(n+1)$-IUFST can effectively be constructed. By applying Lemma 11 we can construct an $s'(n)$-IUFST T with $s'(n) \in O(\ln n)$ that accepts L'.

Let us now assume that emptiness is decidable for an $s(n)$-IUFST with $s(n) \in \Omega(\ln n)$. Then, we can decide the emptiness of T accepting L'. Hence, we can decide the emptiness of L, since L' is empty if and only if L is empty. This implies the decidability of emptiness for linear-time OCAS which is a contradiction to Theorem 13.

Similarly, we can show that both finiteness and infiniteness are undecidable, since L' is finite if and only if L is finite. Finally, the fact that the empty set is accepted by an $s(n)$-IUFST with $s(n) \in \Omega(\ln n)$ and the result that emptiness is undecidable imply that the questions of inclusion and equivalence are undecidable as well. □

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgement

The authors gratefully acknowledge comments and suggestions by the anonymous referees which significantly contributed to improving the paper.

References

[1] H. Bordihn, H. Fernau, M. Holzer, V. Manca, C. Martín-Vide, Iterated sequential transducers as language generating devices, Theor. Comput. Sci. 369 (2006) 67–81.
[2] M. Chrobak, Finite automata and unary languages, Theor. Comput. Sci. 47 (1986) 149–158, Errata: [3].
[3] M. Chrobak, Errata to "Finite automata and unary languages", Theor. Comput. Sci. 302 (2003) 497–498.
[4] C. Citrini, S. Crespi-Reghizzi, D. Mandrioli, On deterministic multi-pass analysis, SIAM J. Comput. 15 (1986) 668–693.
[5] P.C. Fischer, Generation of primes by a one-dimensional real-time iterative array, J. ACM 12 (1965) 388–394.
[6] N. Friburger, D. Maurel, Finite-state transducer cascades to extract named entities in texts, Theor. Comput. Sci. 313 (2004) 93–104.
[7] A. Ginzburg, Algebraic Theory of Automata, Academic Press, 1968.
[8] J. Hartmanis, Computational complexity of one-tape Turing machine computations, J. ACM 15 (1968) 325–339.
[9] J. Hartmanis, R.E. Stearns, Algebraic Structure Theory of Sequential Machines, Prentice-Hall, 1966.
[10] M. Holzer, M. Kutrib, Descriptional complexity—an introductory survey, in: C. Martín-Vide (Ed.), Scientific Applications of Language Methods, Imperial College Press, 2010, pp. 1–58.
[11] M. Holzer, M. Kutrib, A. Malcher, Complexity of multi-head finite automata: origins and directions, Theor. Comput. Sci. 412 (2011) 83–96.

[12] M. Kutrib, Cellular automata - A computational point of view, in: G.B. Enguix, M.D. Jiménez-López, C. Martín-Vide (Eds.), New Developments in Formal Languages and Applications, vol. 113, Springer, 2008, pp. 183–227.
[13] M. Kutrib, Cellular automata and language theory, in: R. Meyers (Ed.), Encyclopedia of Complexity and System Science, Springer, 2009, pp. 800–823.
[14] M. Kutrib, A. Malcher, Cellular automata with limited inter-cell bandwidth, Theor. Comput. Sci. 412 (2011) 3917–3931.
[15] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, Descriptional complexity of iterated uniform finite-state transducers, in: M. Hospodár, G. Jirásková, S. Konstantinidis (Eds.), Descriptional Complexity of Formal Systems (DCFS 2019), in: LNCS, vol. 11612, Springer, 2019, pp. 223–234.
[16] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, Iterated uniform finite-state transducers: descriptional complexity of nondeterminism and two-way motion, in: G. Jirásková, G. Pighizzini (Eds.), Descriptional Complexity of Formal Systems (DCFS 2020), in: LNCS, vol. 12442, Springer, 2020, pp. 117–129.
[17] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, Iterated uniform finite-state transducers on unary languages, in: T. Bures, R. Dondi, J. Gamper, G. Guerrini, T. Jurdzinski, C. Pahl, F. Sikora, P.W.H. Wong (Eds.), Theory and Practice of Computer Science (SOFSEM 2021), in: LNCS, vol. 12607, Springer, 2021, pp. 218–232.
[18] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, Computational and descriptional power of nondeterministic iterated uniform finite-state transducers, Fundam. Inform. 185 (2022) 337–356, https://doi.org/10.3233/FI-222113.
[19] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, Descriptional complexity of iterated uniform finite-state transducers, Inf. Comput. 284 (2022) 104691, https://doi.org/10.1016/j.ic.2021.104691.
[20] M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, Iterated uniform finite-state transducers: descriptional complexity of nondeterminism and two-way motion, J. Autom. Lang. Comb. (2023), in press.
[21] V. Manca, On the generative power of iterated transductions, in: M. Ito, G. Păun, S. Yu (Eds.), Words, Semigroups, and Transductions - Festschrift in Honor of Gabriel Thierrin, World Scientific, 2001, pp. 315–327.
[22] J. Mazoyer, V. Terrier, Signals in one-dimensional cellular automata, Theor. Comput. Sci. 217 (1999) 53–80.
[23] G.H. Mealy, A method for synthesizing sequential circuits, Bell Syst. Tech. J. 34 (1955) 1045–1079.
[24] C. Mereghetti, Testing the descriptional power of small Turing machines on nonregular language acceptance, Int. J. Found. Comput. Sci. 19 (2008) 827–843.
[25] C. Mereghetti, B. Palano, G. Pighizzini, Note on the succinctness of deterministic, nondeterministic, probabilistic and quantum finite automata, RAIRO Inform. Théor. 35 (2001) 477–490.
[26] C. Mereghetti, G. Pighizzini, Two-way automata simulations and unary languages, J. Autom. Lang. Comb. 5 (2000) 287–300.
[27] C. Mereghetti, G. Pighizzini, Optimal simulations between unary automata, SIAM J. Comput. 30 (2001) 1976–1992.
[28] M.L. Minsky, Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines, Ann. Math. 74 (1961) 437–455.
[29] G. Pighizzini, Nondeterministic one-tape off-line Turing machines and their time complexity, J. Autom. Lang. Comb. 14 (2009) 107–124.
[30] H. Umeo, N. Kamikawa, A design of real-time non-regular sequence generation algorithms and their implementations on cellular automata with 1-bit inter-cell communications, Fundam. Inform. 52 (2002) 257–275.
[31] H. Umeo, N. Kamikawa, Real-time generation of primes by a 1-bit-communication cellular automaton, Fundam. Inform. 58 (2003) 421–435.