Polynomial Complementation of Nondeterministic 2-Way Finite Automata by 1-Limited Automata

Bruno Guillon
Université Clermont Auvergne, Clermont Auvergne INP, LIMOS, CNRS, F-63000 Clermont-Ferrand, France

Luca Prigioniero
Department of Computer Science, Loughborough University, Epinal Way, Loughborough LE11 3TU, UK

Javad Taheri
Université Clermont Auvergne, Clermont Auvergne INP, LIMOS, CNRS, F-63000 Clermont-Ferrand, France

Abstract

We prove that, paying a polynomial increase in size only, every unrestricted two-way nondeterministic finite automaton (2NFA) can be complemented by a 1-limited automaton (1-LA), a nondeterministic extension of 2NFAs still characterizing regular languages. The resulting machine is actually a restricted form of 1-LAs — known as 2NFAs with common guess — and is self-verifying. A corollary of our construction is that a single exponential is necessary and sufficient for complementing 1-LAs.

1 Introduction

The study of the resources used by computational models is a central topic in automata theory. One classical problem in this area is to determine the cost of applying operations between languages (*e.g.*, union, intersection, concatenation, Kleene star, *etc.*). Here, the cost is defined as the increase in size of the resulting (or *target*) devices after applying the operation to the languages recognized by the original (or *source*) machines.

In this paper, we focus on the cost of the complementation of regular languages. This operation is usually cheap (*i.e.*, costs at most polynomial) when dealing with deterministic devices, while it is often expensive (*i.e.*, at least exponential) for nondeterministic devices; see Table 1. The reason for this separation has been understood for a long time and originates from the nature of nondeterminism, as illustrated by the case of classical (*one-way*) nondeterministic finite automata (1NFAs). Indeed, the semantics of such a device is that a word is accepted as long as *there exists* a computational path leading to an accepting state. Therefore, in order to acknowledge that a word does not belong to the recognized language, one should somehow check that *every computational path* leads to a non-accepting state,[1] a semantic which is hardly captured by nondeterminism. This issue does not exist for one-way deterministic finite automata (1DFAs), because they admit a unique computational path on each input, and thus existential and universal quantification on computational paths collapse. Indeed, it is folklore that exchanging accepting and non-accepting states of a 1DFA, while keeping the rest of the structure (initial state and transitions) unchanged, yields a 1DFA recognizing the complement of the language.[1] On the other hand, it is well known that the transformation of a 1NFA into another one recognizing the complement of the language may cost as much as determinizing it and then complementing it (as explained above) in the worst case [14, 1, 7].

[1] For ease of discussion, we admit here that the automata are complete, that is, no computational path gets stuck in the middle of the input.

model	cost
1DFA	trivial
2DFA	linear [3]
D1-LA	poly [4]

model	cost
1NFA	exp [14]
2NFA	??? (related to SS78, *via* [3])
1-LA	exp (lower bound in [12, 5], and upper bound in Corollary 3.3)

Table 1 Tight cost orders for the complementation on different models of finite automata. Here, the target device is the same as the source device, and it is indicated in the first column. "SS78" stands for "the Sakoda and Sipser conjecture" [14]. It can be observed that the transformation is cheap (*i.e.*, at most polynomial) for deterministic devices, and expensive (*i.e.*, exponential) or unknown for nondeterministic ones.

Yet, the corresponding question remains unsolved for other regular language recognizers, and in particular when dealing with *two-way finite automata*, an extension of finite automata allowing the machine to move its head both back and forth, and which still characterizes regular languages. Indeed, although complementing two-way deterministic automata (2DFAs) has been non-trivially shown to cost linear only [3], the cost for complementing their nondeterministic counterparts (2NFAs) is still unknown in the general case.Worse still, the best-known upper bound is exponential and is obtained by transforming the source 2NFA into an equivalent 1DFA. That is, neither two-wayness nor nondeterminism are exploited for complementing arbitrary 2NFAs. Indeed, also the cost for determinizing 2NFAs is a longstanding open question known as *"the Sakoda and Sipser problem"* [14] (see *e.g.* [10] for a survey). As shown in [3], the two problems are related *via* the linear-cost complementation of 2DFAs. On the one hand, finding an exponential (or super-polynomial) lower bound for complementing 2NFAs would imply a similar lower bound for determinizing them. On the other hand, finding a polynomial (or sub-exponential) upper bound for determinizing 2NFAs would imply a similar upper bound for complementing them. It is worth noting that a polynomial-cost complementation of 2NFAs has been obtained in some particular cases, *e.g.*, in the *unary* setting [3], or when the 2NFA makes a restricted use of nondeterminism, known as *outer-nondeterminism* [2].

In this paper, we study the cost of complementing two-way finite automata following a different approach: Instead of using the same model as source and target devices, we relax the target machines to have some rewriting capabilities. In particular, we use as a target device a machine called *1-limited automaton* (1-LA), which is an extension of 2NFAs that can rewrite the contents of the tape *only* during the first visit to each cell. This model is not more powerful than 2NFAs [16], *i.e.*, it recognizes regular languages only. However, there are cases where it can represent languages more succinctly than 2NFAs (for a recent survey on this model, see [11]). This approach has already been used to provide succinct representations of operations that have exponential cost when both source and target machines are 1DFAs (Kleene star, reversal and concatenation), but can be done at a polynomial cost when the target machine is a *deterministic 1-LA* (D1-LA) [13].

In the survey [11], the author identifies some problems regarding the descriptional complexity of 1-LAs. In particular, the question of the cost of the conversion of 2NFAs into equivalent D1-LAs (Problem 4), as well as those of the cost of determinizing 1-LAs (Problem 2) are raised. Just as complementing 2NFAs relates to the Sakoda and Sipser problem, complementing 2NFAs with 1-LAs relates to these problems; see Figure 1.

Our results. We show polynomial simulations of 1NFAs and 2NFAs by *self-verifying* 1-LAs (Theorems 3.2 and 4.13), the property of being self-verifying meaning that the devices are

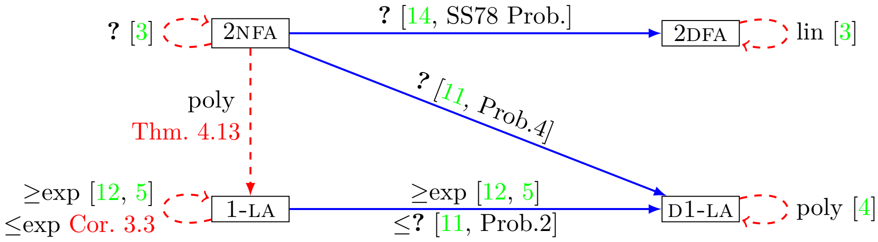

Figure 1 Size cost orders for various transformations discussed in introduction. Plain blue arrows mean conversions of sources into equivalent targets, while dashed red arrows mean complementations of sources with targets. Question marks indicates the open problems, including "the Sakoda and Sipser conjecture" denoted as "SS78" [14].

able to recognize both the language and its complement (the formal definition is given in Section 2). In both constructions, the resulting devices are 1-LAs of particular form, known as 2NFAs *with common guess* (2NFA+cgs), in which the rewriting of the tape is made during an initial nondeterministic memoryless traversal of the input – see [4, 5] for details and results on this model. Although a polynomial simulation of 1NFAs by self-verifying 1-LAs is implied by Theorem 4.13, this particular case is treated in Theorem 3.2, which presents a specific construction for this case that is simpler, cheaper, and serves as a preparatory step for the more technical second construction. Also, because every 1-LA can be converted into a 1NFA paying a single exponential [12], a consequence of Theorem 3.2 is a single exponential upper bound for the cost of the complementation of 1-LAs (Corollary 3.3). This improves the best-known upper bound for the transformation, which used the simulation of 1-LAs by 1DFAs at a doubly-exponential cost. Since an exponential lower bound is known for this transformation [5], the cost order is tight. Figure 1 summarizes our results and their connection to open questions and some related results.

Outline. The paper is organized as follows. Section 2 gathers the definitions and notations used throughout the subsequent sections. In Section 3 we present the conversion of 1NFAs into self-verifying 2NFA+cgs, and its consequence on the cost of the complementation of 1-LAs. In Section 4 we develop the conversion of 2NFAs into self-verifying 2NFA+cgs. A brief conclusion is given in Section 5.

2 Preliminaries

In this section, we recall some fundamental definitions and notations used throughout the paper. We assume that the reader is familiar with basic concepts from formal languages and automata theory (see, *e.g.*, [6]).

For a set S, $\#S$ denotes its cardinality and 2^S denotes its powerset. For $n \in \mathbb{N}$, $[n]$ denotes the set consisting of the first n natural numbers (including 0), namely $[n] = \{0, \ldots, n-1\}$; in particular $[0] = \emptyset$ and $[1] = \{0\}$. Given an alphabet Σ, the set of strings over Σ is denoted by Σ^*. It includes the empty string denoted by ε. The length of a word $w \in \Sigma^*$ is denoted by $|w|$, and the set of strings over Σ of length i is denoted by Σ^i. The number of occurrences of a symbol $\sigma \in \Sigma$ in w is denoted by $|w|_\sigma$. The positions (or indices) of symbols within a string w are indexed from 0 to $|w| - 1$. We use the notation $w[i]$ to indicate the symbol at position $i \in [|w|]$ of w (*i.e.*, the $(i+1)$-th symbol of w, *e.g.*, $w[0]$ is the first symbol of w) and $w[i, j]$ for the factor of w from index i to index j included, $0 \le i, j < |w|$. If $i > j$, we set $w[i,j] = \varepsilon$ by convention. Hence, the length of $w[i,j]$ is 0 if $i > j$ and $j - i + 1$ otherwise.

▶ **Definition 2.1.** *A two-way nondeterministic finite automaton (2NFA) A is a tuple $\langle Q, \Sigma, \delta, q_{\mathsf{start}}, q_{\mathsf{f}} \rangle$, where Q is the finite set of states, Σ is the input alphabet, $q_{\mathsf{start}} \in Q$ is the initial state, q_{f} is the final state,*[2] *and $\delta : Q \times \Sigma_{\triangleright,\triangleleft} \to 2^{Q \times \{-1,+1\}}$ is a nondeterministic transition function with $\Sigma_{\triangleright,\triangleleft} = \Sigma \cup \{\triangleright, \triangleleft\}$, where $\triangleright, \triangleleft \notin \Sigma$ are two special symbols called the left and the right endmarker, respectively.*

We shall often assume $Q = [n]$ for $n = \#Q$, so that Q is ordered, with $-1 \notin Q$ as a minimum.

In 2NFAs, the input is written on the tape surrounded by the two endmarkers, the left endmarker being at tape position zero. Hence, on input w, the right endmarker is at position $|w| + 1$. In one move, A reads an input symbol, changes its state, and moves the head one position backward or forward depending on whether δ returns –1 (a *left move*) or +1 (a *right move*), respectively. Furthermore, the head cannot pass the endmarkers. The machine accepts the input if there exists a *computational path* which starts from the initial state q_{start} with the head on the cell at position 1 (*i.e.*, scanning the first letter of w if $w \neq \varepsilon$ and scanning $\triangleleft$ otherwise) and eventually halts in the final state q_{f} with the head scanning the right endmarker. The language accepted by A is denoted by $\mathcal{L}(\mathsf{A})$.

A 2NFA is *one-way* (1NFA) if its head can never move left, *i.e.*, if no transition returns –1. The transition function of a 1NFA is seen as a function $\delta : Q \times \Sigma \to 2^Q$ (that is, the endmarkers and the instruction for head direction are irrelevant).

The above models are all read-only machines. We now extend 2NFAs with a limited write ability. A *1-limited automaton* (1-LA, for short) A is a 2NFAs over Σ that, on every step on symbol σ, rewrites it with some symbol $\tau \notin \Sigma$ from a *work alphabet* $\Delta \supset \Sigma$ if the contents of the cell has not been already rewritten, namely if $\sigma \in \Sigma$. In other words, A can modify the contents of a cell only when it visits the cell for the first time and the cell does not contain an endmarker. Acceptance for 1-LAs is defined exactly as for 2NFAs, and the language accepted by a given 1-LA A is denoted by $\mathcal{L}(\mathsf{A})$.

Common guess. In this paper, we use as target devices particular cases of 1-LAs, whose computations are somehow split into two phases. In the first phase, using one state only, these machines make a single one-way pass over the input during which they nondeterministically rewrite (or *annotate*) every tape cell symbol, and then move the head back to the left endmarker. In the second phase, they perform read-only two-way computations on the annotated word. In other words, this model can be seen as an extension of 2NFAs with the ability (called *common guess*) to initially annotate the input word $w \in \Sigma^*$ using some annotation symbols from a fixed alphabet Γ, to which we refer as the *annotation alphabet*. The annotated word resulting from this initial phase is a word over the product alphabet $\Sigma \times \Gamma$. It is nondeterministically chosen among all the words $v \in (\Sigma \times \Gamma)^*$ such that $\pi_1(v) = w$ (implying $|v| = |w|$), where π_1 is the natural projection of $(\Sigma \times \Gamma)^*$ onto Σ^*. We shall also use the notation π_2 for the projection of $(\Sigma \times \Gamma)^*$ onto Γ^*, and the convention $\pi_1(\triangleright) = \pi_2(\triangleright) = \triangleright$ and $\pi_1(\triangleleft) = \pi_2(\triangleleft) = \triangleleft$.

▶ **Definition 2.2.** *A 2NFA with common guess (2NFA+cg) is a triplet $\mathsf{M} = \langle \mathsf{A}, \Sigma, \Gamma \rangle$ where Σ is the input alphabet, Γ is the annotation alphabet, and A is a 2NFA over the product alphabet $\Sigma \times \Gamma$.*

---

[2] Notice that, for convenience, we shall assume that 2NFAs have a unique final state. This is not a limitation, since each n-state 2NFA with many final states can easily be simulated by a $(n + 1)$-state 2NFA with, as in our definition, a single final state. (We do not consider one-way deterministic finite automata, for which this restriction makes a difference.) The same comment applies on initial states.

The language recognized by M is:

$$\mathcal{L}(\mathsf{M}) = \{w \in \Sigma^* \mid \exists v \in (\Sigma \times \Gamma)^*,\ v \in \mathcal{L}(\mathsf{A}) \text{ and } \pi_1(v) = w\} = \pi_1(\mathcal{L}(\mathsf{A})).$$

In [4], the authors showed that a polynomial increase in size is always sufficient for the conversion of 1-LA into 2NFA+cg. Conversely, every n-state 2NFA+cg can be converted into an $(n+1)$-state 1-LA. For a recent discussion on this model, we refer the reader to [5].

Configurations and computations. Given one of the devices M under consideration, a *configuration* is represented as a string $\triangleright x \cdot p \cdot y \triangleleft$, meaning that p is the current state, $xy \in \triangleright\Psi^*\triangleleft$ is the contents of the tape (here Ψ denotes the *tape alphabet*: Σ, Δ, or $\Sigma \times \Gamma$, depending on the model under consideration) and the head is scanning the first symbol of $y\triangleleft$. Hence, the *initial configuration* over input w is $\triangleright \cdot q_{\mathsf{start}} \cdot w \triangleleft$, and an *accepting configuration* is a configuration of the form $\triangleright x \cdot q_{\mathsf{f}} \cdot \triangleleft$. The transition relation between configurations of M is denoted by $\overset{\mathsf{M}}{\vdash}$, and its reflexive-transitive closure by $\overset{\mathsf{M}*}{\vdash}$. A *computational path* of M is a sequence of configurations $c_0, \ldots, c_r$ such that $c_i \overset{\mathsf{M}}{\vdash} c_{i+1}$ for each $i < r$. It is *initial* (resp. *accepting*) if c_0 is initial (resp. c_r is accepting). In order to emphasize the locality of some computational paths, we also represent *partial configurations* as $u \cdot p \cdot v$, where p is the current state and $uv \in \{\varepsilon, \triangleright\}\Delta^*\{\varepsilon, \triangleleft\}$ is a factor of the tape content. The relations $\overset{\mathsf{M}}{\vdash}$ and $\overset{\mathsf{M}*}{\vdash}$, whence the notion of computational path, naturally extend onto partial configurations.

Self-verifying. A nondeterministic state machine M is said to be *self-verifying* if it has two different final states, one *accepting* and one *rejecting*, and the set of words recognized by M is partitioned into those admitting an *accepting computational path* (*i.e.*, ending in the accepting state), and those admitting a *rejecting computational path* (*i.e.*, ending in the rejecting state). In particular, no word admits both an accepting and a rejecting computational path. Computational paths that are neither accepting nor rejecting are called *aborted* (*i.e.*, ending in a non-final state).

The set of words admitting an accepting computational path is denoted as $\mathcal{L}(\mathsf{M})$, and the set of words admitting a rejecting computational path is its complement.

Size of models. For each model under consideration, we evaluate its size as the total number of symbols used to describe it. Hence, under standard representation and denoting by Σ the input alphabet, the *size* of an n-state 2NFA is $\mathcal{O}(n^2\,\#\Sigma)$, that of an n-state 1-LA with work alphabet $\Delta \supset \Sigma$ is $\mathcal{O}(n^2\,\#\Delta^2)$, and that of an n-state 2NFA+cg with annotation alphabet Γ is $\mathcal{O}(n^2\,\#\Sigma \cdot \#\Gamma)$. In our work, we generally consider $\#\Sigma$ as a constant.

3 Complementing 1nfas with 2nfa+cgs

In this section, we present a construction that transforms an arbitrary n-state 1NFA into an equivalent self-verifying 2NFA+cg with polynomially many states and 2 annotation symbols. As in [3, 2], our simulation is based on the *inductive counting* technique.

Let $\mathsf{A} = \langle Q, \Sigma, \delta, q_{\mathsf{start}}, q_{\mathsf{f}} \rangle$ be an n-state 1NFA. We assume $Q = [n]$. For $w \in \Sigma^*$, we define X_w^{A} to be the set of states the automaton A reaches after reading w, *i.e.*, $X_w^{\mathsf{A}} = \delta(q_{\mathsf{start}}, w)$. Working on w, a 2NFA can simulate A several times, bringing the head back to the left endmarker before each simulation. It may thus find several states belonging to X_w^{A}. Suppose that the number m of states reached after reading w is given, *i.e.*, $m = \#X_w^{\mathsf{A}}$. Then, by simulating m times A on w, and controlling at each iteration but the first that the state reached at the end of the simulation is larger than the preceding one, a 2NFA can enumerate

Procedure 1 `enum_`$X(m)$

1 $\mathtt{q}_{\mathtt{prev}} \leftarrow -1$
2 **for** $i \leftarrow 1$ **to** m **do**
3     $\mathtt{q}_{\mathtt{next}} \leftarrow$ `nsimul_`A`()`
4   **if** $\mathtt{q}_{\mathtt{next}} \leq \mathtt{q}_{\mathtt{prev}}$ **then abort**
5     $\mathtt{q}_{\mathtt{prev}} \leftarrow \mathtt{q}_{\mathtt{next}}$
6   **output** $\mathtt{q}_{\mathtt{next}}$

Procedure 2 `check_annot(m)`

// starting from last position of x_i
7 $\mathtt{m} \leftarrow \mathtt{m} - |\pi_2(x_i)|_1$ // move $n-1$ times leftward
8 **if** $\mathtt{m} \neq 0$ **then abort**
9 $\mathtt{m} \leftarrow \mathtt{m} + |\pi_2(x_i)|_1$ // move $n-1$ times rightward
10 **foreach** $p \in$ `enum_`$X(m)$ **do**
11   move $n-p-1$ times leftward
12   **if** $\pi_2($`read()`$) \neq 1$ **then abort**
13   move $n-p-1$ times rightward

Procedure 3 `member_`$X(q_t,\ m)$

14 **foreach** $\mathtt{q} \in$ `enum_`$X(m)$ **do**
15   **if** $\mathtt{q} = q_t$ **then return** true
16 **return** false

Procedure 4 `count_next_`$X(m)$

17 $\mathtt{m}_{\mathtt{next}} \leftarrow 0$
18 **foreach** $\mathtt{p} \in Q$ **do**
19   **foreach** $\mathtt{r} \in$ `enum_`$X(m)$ **do**
20   **if** $\mathtt{p} \in \delta(\mathtt{r},$ `read()`$)$ **then**
21             $\mathtt{m}_{\mathtt{next}} \leftarrow \mathtt{m}_{\mathtt{next}} + 1$
22   **break**
23 **return** $\mathtt{m}_{\mathtt{next}}$

all the m states of X_w^{A}. This process is described in Procedure 1, where `nsimul_`A stands for the simulation of A from the initial configuration.[3] Hence, according to whether one or none of them is the accepting state, the 2NFA recognizes whether w belongs to $\mathcal{L}(\mathsf{A})$ or to its complement; see Procedure 3 in which the accepting state is passed as a parameter q_t. Remarkably, a 2NFA B_m with $\mathcal{O}(n^2 m) \subseteq \mathcal{O}(n^3)$ states can implement this process. Indeed, such a 2NFA only has to simultaneously store in its finite control: the index $i \leq m$ of the iteration, the previously found state ($\mathtt{q}_{\mathtt{prev}}$), and the state in the current simulation ($\mathtt{q}_{\mathtt{next}}$). One strategy to detect whether an input w belongs to the complement of $\mathcal{L}(\mathsf{A})$ would therefore be to first compute $m = \#X_w^{\mathsf{A}}$, and then run the 2NFA B_m. Yet, computing $\#X_w^{\mathsf{A}}$ is not an easy task.

Suppose now that $\#X_w^{\mathsf{A}}$ is unknown, but that there exists a nondeterministic procedure `nsimul_`A, which, starting from some position $|u|$, where u is a prefix of the input w, eventually halts at the same position returning a state q if and only if $q \in X_u^{\mathsf{A}}$. Then a 2NFA equipped with `nsimul_`A may inductively compute $\#X_u^{\mathsf{A}}$ for each successive prefix u of w as follows. Initially, $\#X_\varepsilon^{\mathsf{A}} = 1$ since $X_\varepsilon^{\mathsf{A}} = \{q_{\mathsf{start}}\}$ by definition. Let $u\sigma$ be a prefix of w. In order to compute $\#X_{u\sigma}^{\mathsf{A}}$ from $\#X_u^{\mathsf{A}}$, the automaton tests for each state p whether it belongs to $X_{u\sigma}^{\mathsf{A}}$, and counts those for which the answer is positive. This process is described in Procedure 4, in which σ is read from the tape (indicated as `read()`). Testing whether a given state p belongs to $X_{u\sigma}^{\mathsf{A}}$ is based on the following basic observation:

$$p \in X_{u\sigma}^{\mathsf{A}} \qquad \iff \qquad \exists r \in X_u^{\mathsf{A}} \text{ such that } p \in \delta(r, \sigma). \tag{1}$$

Using the knowledge of $m = \#X_u^{\mathsf{A}}$ and `nsimul_`A, the automaton enumerates the m distinct states belonging to X_u^{A} in ascending order as explained before (*i.e.*, using Procedure 1). As soon as it finds one from which a transition on σ allows to enter p, it breaks the loop as $p \in X_{u\sigma}^{\mathsf{A}}$ has been witnessed. If otherwise the m states are correctly found but none of them has an outgoing transition to p on σ, a witness of $p \notin X_{u\sigma}^{\mathsf{A}}$ has been obtained. Once the number of states in $X_{u\sigma}^{\mathsf{A}}$ has been computed, the automaton forgets those of X_u^{A}, moves its head one cell to the right, and proceeds with the next iteration of the induction.

However, implementing `nsimul_`A with a 2NFA is challenging. Indeed, since there is an unbounded number of prefixes of inputs, it is not possible to bring the head back to its

[3] In enumeration algorithms, the outputs are not returned at the end of the execution, but yielded as soon as they are found. This allows an outer calling procedure to treat them one-by-one (see, *e.g.*, Procedures 3 and 4). In Procedure 1, this yielding is indicated with the **output** keyword on Line 6.

initial position (in order to restart a computation of A), and then recover the position $|u|$. To overcome this issue, we use annotations, so that it is possible to simulate computations of A without moving the head more than $2n$ cells to the left of position $|u|$. In this way, the automaton can recover the position $|u|$ by maintaining, in its finite control, the distance of the head from that position.

Let $w \in \Sigma^*$. The idea is to consider an annotated word $x \in (\Sigma \times \{0,1\})^*$ such that $\pi_1(x) = w$ and the following property holds. Logically dividing x into factors of length n (or possibly less for the last one), the i-th factor encodes $X^{\mathsf{A}}_{w[0,i\cdot n-1]}$. Formally, decomposing x as $x = x_1 \cdots x_{k+1}$ with $|x_{k+1}| \leq n$ and $|x_i| = n$ for each $i \leq k$, every $\pi_2(x_i)$ (except, possibly $\pi_2(x_{k+1})$ if it has length less than n) encodes the set $X^{\mathsf{A}}_{\pi_1(x_1\cdots x_i)} = X^{\mathsf{A}}_{w[0,i\cdot n-1]}$. In this way, our nondeterministic procedure `nsimul_`A starting from some position $i\cdot n+j$, $0 \leq i \leq k$ and $0 \leq j < n$, operates in two phases. First, it scans $\pi_2(x_i)$, from which it extracts some nondeterministically-chosen state $p \in X^{\mathsf{A}}_{w[0,i\cdot n-1]}$ — for $i = 0$, we assume $x_i = \triangleright$ from which the machine extracts $p = q_{\mathsf{start}}$. Second, it performs a direct simulation of A on $\pi_1(x_{i+1}[0,j])$, starting from the selected state p and halting as soon as the cell at position $i\cdot n+j$ is entered. At that time, the reached state in the simulated path, belongs to $\delta(p, w[i\cdot n, i\cdot n+j-1])$, and thus to $X^{\mathsf{A}}_{w[0,i\cdot n+j-1]}$ since $p \in X^{\mathsf{A}}_{w[0,i\cdot n-1]} = \delta(q_{\mathsf{start}}, w[0, i\cdot n-1])$.

A subset S of $Q = [n]$ is naturally encoded as a length-n word $\mathsf{enc}(S) \in \{0,1\}^*$ as follows:

$$\mathsf{enc}(S)[p] = 1 \qquad \Longleftrightarrow \qquad p \in S. \tag{2}$$

Hence, provided $\pi_2(x_i) = \mathsf{enc}(X^{\mathsf{A}}_{w[0,i\cdot n]})$, in order for `nsimul_`$A$ to select $p \in X^{\mathsf{A}}_{w[0,i\cdot n]}$ during its first phase, it is sufficient to choose a position p of x_i such that $\pi_2(x_i[p]) = 1$. The annotation of a word $w \in \Sigma^*$ is defined now.

▶ **Definition 3.1.** *For each $w \in \Sigma^*$, we let $\mathbf{a}_w$ be the $|w|$-length word over $\{0,1\}$ defined as follows. Let k and r be such that $|w| = kn + r$ with $0 \leq r < n$:*

- *if $k = r = 0$ (i.e., $w = \varepsilon$) then $\mathbf{a}_w = \varepsilon$, otherwise,*
- *if $r = 0$ then $\mathbf{a}_w = zy$, where $z = \mathbf{a}_{w[0,(k-1)n]}$, and $y = \mathsf{enc}(X^{\mathsf{A}}_w)$,*
- *if $r > 0$ then $\mathbf{a}_w = zy$, where $z = \mathbf{a}_{w[0,kn]}$, and $y = 0^r$.*

The word $\mathsf{annot}(w)$ is the word x over $\Psi = \Sigma \times \{0,1\}$ such that $\pi_1(x) = w$ and $\pi_2(x) = \mathbf{a}_w$.

Our goal is to design a 2NFA B over $\Psi = \Sigma \times \{0,1\}$ of polynomial size in n, with two distinguished states q_{acc} (for acceptance) and q_{rej} (for rejection) that satisfies:

- B accepts x if and only if $\pi_1(x) \in \mathcal{L}(\mathsf{A})$ and $x = \mathsf{annot}(\pi_1(x))$;
- B rejects x if and only if $\pi_1(x) \notin \mathcal{L}(\mathsf{A})$ and $x = \mathsf{annot}(\pi_1(x))$.

$$\begin{aligned} &\mathsf{B} \text{ accepts } x \text{ if and only if } \pi_1(x) \in \mathcal{L}(\mathsf{A}) \text{ and } x = \mathsf{annot}(\pi_1(x)); \\ &\mathsf{B} \text{ rejects } x \text{ if and only if } \pi_1(x) \notin \mathcal{L}(\mathsf{A}) \text{ and } x = \mathsf{annot}(\pi_1(x)). \end{aligned} \tag{D}$$

Notice that B itself is not self-verifying, as it can neither accept nor reject an input x such that $x \neq \mathsf{annot}(\pi_1(x))$. However, the 2NFA+cg $\langle \mathsf{B}, \Sigma, \{0,1\}\rangle$ is self-verifying since every input $w \in \Sigma^*$ admits an annotated variant $\mathsf{annot}(w)$ that should be either accepted or rejected by B according to whether w belongs to $\mathcal{L}(\mathsf{A})$ or not.

The design of B follows the above-explained inductive counting strategy. In particular, it uses the already-presented procedures `enum_`X, `member_`X, and `count_next_`X (see Procedures 1, 3, and 4), as well as `nsimul_`A. To implement the latter, B maintains two variables in its finite control: the value of its head position modulo n (so it knows where each factor x_i begins, and to which state p an annotation symbol 1 correspond), and the distance of its current head position to the position $|z|$, where z is the input prefix under consideration. By the locality of $\mathtt{nsimul\_}A$, the latter information is an integer less than $2n$.

Not only B has to behave correctly on inputs $\mathsf{annot}(w)$: it also has to detect ill-formed inputs, namely words $x \in \Psi^*$ for which $x \neq \mathsf{annot}(\pi_1(x))$. To this end, each time the length of the prefix z under consideration is a positive multiple of n, before considering the next prefix, B checks the annotation of the length-n suffix x_i $(i = |z|/n)$ of z. This can easily be done, since at that time $m = \#X^{\mathsf{A}}_{\pi_1(z)}$ has been computed. Hence, the automaton can check that $\pi_2(x_i)$ has m occurrences of 1, and, using $\mathtt{enum\_}X$, that $\pi_2(x_i[p]) = 1$ for each $p \in X^{\mathsf{A}}_{\pi_1(z)}$.

Evaluating the size of B, we get that $\mathcal{O}(n^3)$ states are sufficient for implementing $\mathtt{nsimul\_}A$, including the two above-mentioned state components relative to the head position. Next, B further uses a $\mathcal{O}(n^4)$-state component for storing $\mathtt{m} = \#X^{\mathsf{A}}_u$, $\mathtt{m_{next}} \leq \#X^{\mathsf{A}}_{u\tau}$, and two intermediate states $\mathtt{q}$ and $\mathtt{q_{prev}}$. Hence, the total number of states belongs to $\mathcal{O}(n^7)$.

▶ **Theorem 3.2.** *Every n-state 1NFA has an equivalent self-verifying 2NFA+cg with $\mathcal{O}(n^7)$ many states and 2 annotation symbols.*

A direct consequence is that complementing 1-LAs costs at most a single exponential.

▶ **Corollary 3.3.** *For each n-state 1-LA recognizing some language $L \subseteq \Sigma^*$, there exists a 1-LA with a single exponential number of states in n and $3\,\#\Sigma$ work symbols which recognizes the complement of L.*

Proof. From [12, Theorem 2], each n-state 1-LA over Σ can be simulated by a 1NFA with at most $n2^{n^2}$ states. Also by Theorem 3.2, each m-state 1NFA can be converted into an equivalent self-verifying 2NFA+cg (which is a particular case of 1-LA) with $\mathcal{O}(m^7)$ states and 2 annotation symbols. Combining these two results, one can simulate each n-state 1-LA, by a self-verifying 1-LA with work alphabet $\Delta = \Sigma \cup (\Sigma \times \{0,1\})$ and a number of states in $\mathcal{O}(2^{7(n^2+\log n)}) \subset 2^{\mathcal{O}(n^2)}$. ◀

As demonstrated in [5], the exponential bound in the above corollary cannot be avoided in the worst case, even when the source device is a *unary* 2DFA+cg, namely a 2NFA+cg over a singleton input alphabet whose underlying 2NFA (working on nonunary annotated words) is *deterministic*.[4]

4 Complementing 2nfas with 2nfa+cgs

In this section, we show how to simulate an arbitrary n-state 2NFA A over Σ with a self-verifying 2NFA+cg M that has polynomially many states in n and annotation alphabet $\{0,1\}$. In order for $\mathsf{M} = \langle \mathsf{B}, \Sigma, \{0,1\} \rangle$ to be self-verifying, we shall ensure that its underlying 2NFA B over $\Psi = \Sigma \times \{0,1\}$ satisfies the following sufficient property (see Lemma 4.12). For every word $w \in \Sigma^*$, there exists a word $\mathsf{annot}(w) \in \Psi^*$ such that $\pi_1(\mathsf{annot}(w)) = w$, and on input $x \in \Psi^*$:

- B *accepts* x if and only if $\pi_1(x) \in \mathcal{L}(\mathsf{A})$ and $x = \mathsf{annot}(\pi_1(x))$;
- B *rejects* x if and only if $\pi_1(x) \notin \mathcal{L}(\mathsf{A})$ and $x = \mathsf{annot}(\pi_1(x))$.

[4] Remark that 2DFA+cgs are nondeterministic devices, since the annotation phase is nondeterministic.

Notice that B itself is not a self-verifying 2NFA. Indeed, on inputs not belonging to $\mathsf{annot}(\Sigma^*)$, B can neither accept nor reject.[5] Yet, as the goal is to check whether a word $w \in \Sigma^*$ belongs to $\mathcal{L}(\mathsf{A})$, and since each such word has an annotated version $\mathsf{annot}(w)$ that is either accepted or rejected by B, we obtain self-verifyingness for the ~~1-LA~~ 2NFA+cg M.

The section is structured as follows. In Section 4.1, we recall the principle of Shepherdson's construction [15], introducing the basic concepts and properties on which such a simulation as much as ours rely. The definition of $\mathsf{annot}(w)$ is given at the end of the section. The automaton B is then designed in Section 4.2 and the main result is stated.

4.1 L-tables

In [15], Shepherdson proposed a construction to simulate 2DFAs by 1DFAs, which has then been generalized to the simulation of 2NFAs and even 1-LAs by 1DFAs, see, *e.g.*, [8, 12]. The main ingredient of this construction is to store in each state of the finite control of the simulating 1DFA, a table, that we call L-*table*,[6] describing the finitely-many possible behaviors of the simulated two-way machine that may occur on the portion of the tape to the left of the current head position. This is formalized in the following.

For $u \in \Sigma^*$, an L-*segment over u with respect to* A is a computational path of A over $\rhd uv \lhd$ for some $v \in \Sigma^*$ that starts from tape position $|u|$ (hence reading the last symbol of $\rhd u$) and ends in position $|u| + 1$ (hence, reading the first symbol of $v\lhd$) visiting only positions $j \leq |u|$ in the meantime (hence, independent from v). The L-*table of u with respect to* A, denoted t_u^{A}, is the set of pairs (p, q) such that there exists an L-segment over u starting in state p and ending in state q. Formally:

$$t_u^{\mathsf{A}} = \{(p, q) \in Q^2 \mid x \cdot p \cdot \sigma \vdash_{\mathsf{A}}^{*} \rhd u \cdot q\}, \qquad \text{where } x\sigma = \rhd u \text{ with } |\sigma| = 1.$$

Observe that there are finitely many such tables. Also, for every $u \in \Sigma^*$, we denote by X_u^{A} the set of states that A may enter when it visits the cell containing the right endmarker for the first time during a computation over input u. Namely,

$$X_u^{\mathsf{A}} = \{q \in Q \mid \rhd \cdot q_{\mathsf{start}} \cdot u \vdash_{\mathsf{A}}^{*} \rhd u \cdot q\}.$$

Again, remark that there are finitely many such sets. Shepherdson observed that knowing X_u^{A} and t_u^{A} (but not u) is sufficient for deciding whether $u \in \mathcal{L}(\mathsf{A})$. In order to simplify, we slightly modify the simulated 2NFA, so that knowing t_u^{A} will be sufficient to recover X_u^{A} and thus the acceptance of u.

▶ **Lemma 4.1.** *Each n-state 2NFA A over Σ admits an equivalent $(n+1)$-state 2NFA A′ with an inaccessible distinguished state q_{restart} such that, for each $u \in \Sigma^*$ and each state p of A′, on the one hand $(q_{\mathsf{restart}}, p) \in t_u^{\mathsf{A}'}$ if and only if $p \in X_u^{\mathsf{A}'}$, and on the other hand $(p, q_{\mathsf{restart}}) \notin t_u^{\mathsf{A}'}$. Furthermore, $\delta'(q_{\mathsf{restart}}, \lhd) = \{(q_{\mathsf{restart}}, -1)\}$ where δ' is the transition function of A′.*

Proof. The 2NFA A′ is obtained from A by (1) adding a new state q_{restart}, (2) adding transitions returning $(q_{\mathsf{restart}}, -1)$ from q_{restart} on every symbol $\sigma \in \Sigma \cup \{\lhd\}$, and (3) adding a transition from q_{restart} returning $(q_{\mathsf{start}}, +1)$ on $\rhd$ where q_{start} is the initial state of both A′ and A.[7] Hence, from all configurations of the form $x \cdot q_{\mathsf{restart}} \cdot y$ with $xy = \rhd w \lhd$, the machine deterministically

[5] B is actually a *don't-care* automaton [9], namely an automaton with the self-verifying property on inputs from a restricted domain (here, inputs of the form $\mathsf{annot}(w)$ for some $w \in \Sigma^*$).
[6] L for "Left"; In the literature, they have sometimes been called "*Shepherdson's tables*".
[7] Remarkably, although useless for our purpose, the transformation preserves determinism.

reaches the configuration $\triangleright \cdot q_{\mathsf{start}} \cdot w \triangleleft$, namely the initial configuration of both A' and A, from where it performs a direct simulation of A. In particular, $t_u^{\mathsf{A}'}(q_{\mathsf{restart}}) = X_u^{\mathsf{A}'} = X_u^{\mathsf{A}}$ for all $u \in \Sigma^*$. Also, since the only transitions entering q_{restart} are left-move looping around q_{restart}, q_{restart} is inaccessible, and $(p, q_{\mathsf{restart}}) \notin t_u^{\mathsf{A}'}$ for every u and p. ◀

The key of Shepherdson's construction is that, given t_u^{A} and $\tau \in \Sigma$, it is possible to compute $t_{u\tau}^{\mathsf{A}}$ without accessing u. Indeed, an L-segment on $u\tau$ can be decomposed as a sequence alternating left-moves over τ and L-segments on u, followed by a final right-move over τ. This is formalized in Proposition 4.5 below, using the following notions. For $\tau \in \Sigma \cup \{\triangleleft\}$ and $k \geq 0$, we define the binary relation ${T_{u\tau}^{\mathsf{A}}}^k$ (resp. ${S_{u\tau}^{\mathsf{A}}}^k$) on Q as the set of pairs (p, q) for which there is a computational path that starts at position $|u| + 1$ in state p, ends at the same position in state q, visits only positions $j \leq |u| + 1$ in the meantime, and visits the position $|u| + 1$ exactly (resp. at most) $k + 1$ times (including the initial and final visits, in state p and q, respectively).

▶ **Definition 4.2.** *Let A be a 2NFA over Σ, $u \in \Sigma^*$, $\tau \in \Sigma \cup \{\triangleleft\}$, and $k \geq 0$. We define:*

$$
\begin{aligned}
{T_{u\tau}^{\mathsf{A}}}^0 &= \left\{ (p,p) \;\middle|\; p \in Q \right\} \\
{T_{u\tau}^{\mathsf{A}}}^{k+1} &= \left\{ (p,q) \;\middle|\; \exists s, r \in Q \text{ such that } (p,s) \in {T_{u\tau}^{\mathsf{A}}}^k,\ (r,-1) \in \delta(s,\tau) \text{ and } (r,q) \in t_u^{\mathsf{A}} \right\}.
\end{aligned}
$$

Furthermore we define ${S_{u\tau}^{\mathsf{A}}}^k = \bigcup_{j=0}^{k} {T_{u\tau}^{\mathsf{A}}}^j$ *and* ${S_{u\tau}^{\mathsf{A}}}^* = \bigcup_{j \in \mathbb{N}} {T_{u\tau}^{\mathsf{A}}}^j$.

▶ Remark 4.3. By definition, ${S_{u\tau}^{\mathsf{A}}}^*$ is the reflexive and transitive closure of ${T_{u\tau}^{\mathsf{A}}}^1$ which is characterized by:

$$
(p,q) \in {T_{u\tau}^{\mathsf{A}}}^1 \qquad \Longleftrightarrow \qquad \exists r \in Q \colon (r,q) \in t_u^{\mathsf{A}} \text{ and } (r,-1) \in \delta(p,\tau).
$$

Since the family $({S_{u\tau}^{\mathsf{A}}}^k)_k$ is an increasing sequence of subsets of Q^2 with respect to inclusion, it is ultimately constant.

▶ **Proposition 4.4.** *Let A be an n-state 2NFA over Σ, $u \in \Sigma^*$, and $\tau \in \Sigma \cup \{\triangleleft\}$. Then ${S_{u\tau}^{\mathsf{A}}}^* = {S_{u\tau}^{\mathsf{A}}}^j$ for all $j \geq n(n-1)$.*

Proof. By definition, for all $j \geq 0$, ${S_{u\tau}^{\mathsf{A}}}^j \subseteq {S_{u\tau}^{\mathsf{A}}}^{j+1} \subseteq {S_{u\tau}^{\mathsf{A}}}^* \subseteq Q^2$. Furthermore, by the inductive definition of ${T_u^{\mathsf{A}}}^k$, if ${S_u^{\mathsf{A}}}^k = {S_u^{\mathsf{A}}}^{k+1}$ for some k then ${S_u^{\mathsf{A}}}^k = {S_{u\tau}^{\mathsf{A}}}^*$. Since $\#{S_{u\tau}^{\mathsf{A}}}^0 = n$ and $\#{S_{u\tau}^{\mathsf{A}}}^* \leq n^2$,[8] this surely happens for $k \leq n(n-1)$, and hence ${S_{u\tau}^{\mathsf{A}}}^* = {S_{u\tau}^{\mathsf{A}}}^j$ for all $j \geq n(n-1)$. ◀

In order to build $t_{u\tau}^{\mathsf{A}}$ from t_u^{A}, we rely on the following property.

▶ **Proposition 4.5.** *Let $\mathsf{A} = \langle Q, \Sigma, \delta, q_{\mathsf{start}}, q_{\mathsf{f}} \rangle$ be a 2NFA, $u \in \Sigma^*$, and $\tau \in \Sigma$. Then $(p,q) \in t_{u\tau}^{\mathsf{A}}$ if and only if there exists r such that $(p,r) \in {S_{u\tau}^{\mathsf{A}}}^*$ and $(q,+1) \in \delta(r,\tau)$.*

A direct consequence of the above proposition is that $t_u^{\mathsf{A}} = t_v^{\mathsf{A}}$ implies $t_{uw}^{\mathsf{A}} = t_{vw}^{\mathsf{A}}$ for every w. Also, provided A is in the form of Lemma 4.1, we can decide whether $w \in \mathcal{L}(\mathsf{A})$ with the only information of ${S_{w\triangleleft}^{\mathsf{A}}}^*$, which is determined from t_w^{A}:

▶ **Proposition 4.6.** *Let A be a 2NFA over Σ in the form of Lemma 4.1, and $w \in \Sigma^*$. Then $w \in \mathcal{L}(\mathsf{A})$ if and only if $(q_{\mathsf{restart}}, q_{\mathsf{f}}) \in {S_{w\triangleleft}^{\mathsf{A}}}^*$ where q_{f} is the accepting state of A.*

---

[8] If A is in the form of Lemma 4.1, then $\#{S_{u\tau}^{\mathsf{A}}}^* \leq n(n-1)+1$ and thus ${S_{u\tau}^{\mathsf{A}}}^* = {S_{u\tau}^{\mathsf{A}}}^{n(n-2)+1}$.

Proof. By definition, a word $w \in \Sigma^*$ belongs to $\mathcal{L}(\mathsf{A})$ if and only if there exists a computational path that starts from the initial configuration and halts in the accepting state q_f with the head scanning the right endmarker. Every such path should thus visit the right endmarker at least once. Hence, $w \in \mathcal{L}(\mathsf{A})$ if and only if there exists p such that $p \in X_w^\mathsf{A}$ and $(p, q_\mathsf{f}) \in {S_{w\triangleleft}^{\mathsf{A}}}^*$. Since $X_w^\mathsf{A} = t_w^\mathsf{A}(q_\mathsf{restart})$ and $\delta(q_\mathsf{restart}, \triangleleft) = \{(q_\mathsf{restart}, -1)\}$, this is equivalent to $(q_\mathsf{restart}, q_\mathsf{f}) \in {S_{w\triangleleft}^{\mathsf{A}}}^*$. ◀

Not every binary relation $R \subseteq Q^2$ is equal to t_u^A for some u. Nevertheless, each $R \subseteq Q^2$ can be updated according to Proposition 4.5. We formalize this fact in the following, by introducing a variant $\mathsf{A}_{/R}$ of A so that $t_\varepsilon^{\mathsf{A}_{/R}} = R$.

▶ **Definition 4.7.** *Let A be a 2NFA with state set Q, and let $R \subseteq Q^2$. We define $\mathsf{A}_{/R}$ as the 2NFA obtained from A by overwriting its transitions on $\triangleright$ according to R. More precisely, $\mathsf{A}_{/R}$ has the same transitions as A on symbols distinct from $\triangleright$, and transitions from p to $(q, +1)$ on $\triangleright$ if and only if $(p, q) \in R$.*

Trivially, $t_\varepsilon^{\mathsf{A}_{/R}} = R$ and $\mathsf{A}_{/t_\varepsilon^\mathsf{A}} = \mathsf{A}$. Also, if $R = t_u^\mathsf{A}$ for some u then $t_v^{\mathsf{A}_{/R}} = t_{uv}^\mathsf{A}$ for every v.

In Shepherdson's construction, after reading a prefix u of the input, the simulating 1DFA stores the whole table t_u^A (along with the set X_u^A, which thanks to Lemma 4.1 is not needed in our presentation) in its finite control. Then, on symbol τ, it updates it to $t_{u\tau}^\mathsf{A}$ according to Proposition 4.5. Finally, it decides acceptance according to Proposition 4.6. This method comes with a cost: storing the L-tables in the finite control implies an exponential number of states (which, for the simulation of 2NFAs by 1DFAs, cannot be avoided in general; see [8] for a precise analysis). In order to keep the size of our simulating self-verifying 2NFA+cg polynomial, we do not store the successive L-tables in the finite control of the machine. We rather maintain their sizes as a state component, and encode *some* of them *on the tape*. Intuitively, the tape will be virtually divided into portions of length n^2 (possibly the last portion being shorter), so that the j-th portion stores on its annotation track an encoding of the table t_x^A where x is the input prefix of length jn^2. We naturally encode a relation $R \subseteq Q^2$ in a length-n^2 word $\mathsf{enc}(R)$ over $\{0,1\}$, as follows:[9]

$$\mathsf{enc}(R)[pn+q] = 1 \iff (p,q) \in R \qquad \text{for each } p, q \in Q = [n].$$

Notice that the encoding defines a bijection between binary relations on Q and length-n^2 words over $\{0,1\}$. We denote by dec the inverse of enc, *i.e.*, $\mathsf{dec} = \mathsf{enc}^{-1}$.

▶ **Definition 4.8.** *For $w \in \Sigma^*$, the annotation of w is the word $\mathsf{a}_w \in \{0,1\}^*$ of length $|w|$ defined as follows. Let $k \in \mathbb{N}$ and $r \in [n^2]$ such that $|w| = kn^2 + r$.*

- *If $k = r = 0$ (i.e., $w = \varepsilon$) then $\mathsf{a}_w = \varepsilon$, otherwise,*
- *If $r = 0$ then $\mathsf{a}_w = zy$ where $z = \mathsf{a}_{w[0,(k-1)n^2]}$ and $y = \mathsf{enc}(t_w^\mathsf{A})$,*
- *If $r > 0$ then $\mathsf{a}_w = zy$ where $z = \mathsf{a}_{w[0,kn^2]}$ and $y = 0^r$.*

The word $\mathsf{annot}(w)$ is the word x over $\Psi = \Sigma \times \{0,1\}$ such that $\pi_1(x) = w$ and $\pi_2(x) = \mathbf{a}_w$.

[9] Since the only relations over Q we encode are relations t_u^A for some u, and because they are included in $Q \times (Q \setminus \{q_\mathsf{restart}\})$ when assuming A normalized according to Lemma 4.1, a word of length $n(n-1)$ would be enough for encoding it. This would match the space needed for storing the pairs $(t_u^\mathsf{A}, X_u^\mathsf{A})$ (of the corresponding non-normalized $(n-1)$-state 2NFA) in Shepherdson's construction. For readability, we do not apply this minor optimization.

4.2 The automaton B

As in Section 3, a key ingredient in our construction is a subprocedure, implemented by a 2NFA, which, starting from and ending in some position $|v|$ for some prefix v of the input, nondeterministically simulates the computational paths of A on $\triangleright v$. Unlike those of 1NFAs described in Section 3, such computational paths are here L-segments or variants of L-segments. Again, this procedure is made possible by the use of annotations, which allow to keep the head close to the position $|v|$ during the simulation. This idea has already been used in [4] for proving a polynomial upper bound for the simulation of 1-LAs by halting 2NFA+cgs. However, in that construction the resulting machine is only able to check the inclusion of the encoded relations in the corresponding L-tables. In other words, the simulating 2NFA+cg is allowed to *"lose"* some pairs of the L-tables. Although this is sufficient for the simulated machine to recover acceptance of the input, it turns out to be insufficient if, as in the present work, we aim to recover *rejection* of the input. To address this lack of information, following the same strategy as in Section 3 and [3, 2], our construction uses inductive counting to check that every encoded relation is *equal* to the corresponding L-table, and finally detect whether the input belongs to $\mathcal{L}(\mathsf{A})$ or not.

4.2.1 Simulating L-segments using annotations.

The automaton B maintains a variable `rpos` ranging over $[2n^2]$ in its finite control, which is updated according to each head move as now explained. The variable `rpos` is incremented on right moves and decremented on left moves like a counter with the two following differences: decrementing from value 0 is forbidden (hence left-moves from a state in which $\texttt{rpos} = 0$ are forbidden), and incrementing from value $2n^2 - 1$ resets the counter to n^2. In the initial configuration, in which the head is at position 1, $\texttt{rpos} = n^2$. Hence, at any point of the computation with the head at position h, the value of `rpos` is congruent to $h - 1$ modulo n^2. We call *current window* the portion of the tape of length at most $2n^2$, going from position $\max(0, h - \texttt{rpos})$ to position $\min(\ell + 1, h - \texttt{rpos} + 2n^2 - 1)$ where ℓ is the input length. By *relative position* i, for $i \in [2n^2]$, we refer to the position i relatively to the current window, *i.e.*, the absolute position $\max(0, h - \texttt{rpos}) + i$. The window will slide from left to right along the input. Indeed, on the one hand, as decrementing `rpos` from value 0 is forbidden the head can never visit cells to the left of the current window. On the other hand, updating the value of `rpos` from $2n^2 - 1$ to n^2 on a right-move shifts the window to the right by n^2 cells in the following sense: after the shift, a cell that was at some relative position i before the shift is either not covered by the window if $i < n^2$, or at relative position $i - n^2$ otherwise. By using `rpos`, B can navigate within this window without getting lost. From now on, we assume the machine has always access to the value `rpos` without mentioning it explicitly. More generally, we say that a 2NFA is `rpos`-*aware* when it has access to `rpos`, and we do not count the underlying state component when analyzing its size. In a (partial) configuration of some `rpos`-aware 2NFA, $q_{[\texttt{rpos}=i]}$ indicates state q with $\texttt{rpos} = i$.

One of the key ingredients allowing our inductive counting is a nondeterministic subroutine `nsimul_t` (Procedure 5), which, assuming the annotation on the left side of the current window is correct, allows to simulate L-segments of A while keeping the head within the current window. In particular, on input w, if called from position $|x|$ for a prefix x of $\mathsf{annot}(w)$, the procedure eventually halts (if an L-segment on $\pi_1(x)$ is found) with the head at position $|x|+1$. The procedure can be implemented by a 2NFA of polynomial size in n, as stated in the following lemma (recall Definition 4.7).

Procedure 5 `nsimul_t`(p)

```
24 i ← rpos − n² + 1
25 p_curr ← p
26 while rpos < n² + i do
27     if rpos ≥ n² or read() = ▷ then
           // direct simulation of A reading the input track
28         choose (r, d) ∈ δ(p_curr, π₁(read()))
29         p_curr ← r
30         move the head according to d
31     else
           // recovering L-segment from the annotation track
32         while true do
33             if p_curr · n ≤ rpos < (p_curr + 1)n and π₂(read()) = 1 then
34                 choose b ∈ {true, false}
35                 if b then break
36             if rpos = 0 then abort
37             move the head leftward
38         p_curr ← rpos mod n
39         move the head rightward to relative position n²
40 return p_curr
```

▶ **Lemma 4.9.** *Let $i \in [n^2]$. There exists a* rpos*-aware* 2NFA C_i *with same state set* Q *as* A*, such that, for every* $x \in \{\triangleright\} \cup \Psi^{n^2}$*, every* $y \in \Psi^i$*, and every* $p, q \in Q$*:*

$$z \cdot p_{[\texttt{rpos}=n^2+i-1]} \cdot \sigma \vdash^{\mathsf{C}_i *} z\sigma \cdot q_{[\texttt{rpos}=n^2+i]} \qquad \Longleftrightarrow \qquad (p,q) \in t^{\mathsf{A}_{/R}}_{\pi_1(y\sigma)}$$

where $z\sigma = xy$ *with* $|\sigma| = 1$*, and* R *equals* $t^{\mathsf{A}}_{\varepsilon}$ *if* $x = \triangleright$ *or* $\mathsf{dec}(\pi_2(x))$ *otherwise. Furthermore,* C_i *has no transition outgoing configurations in which* $\texttt{rpos} \geq n^2 + i$*; in particular, the above configuration* $z\sigma \cdot q_{[\texttt{rpos}=n^2+i]}$ *is halting.*

Proof. The behavior of C_i is described in Procedure 5, where the value of i is deduced from the initial relative position (Line 24), and which uses a variable $\mathtt{p_{curr}}$ ranging over Q. The states of C_i are the valuations of this variable. The absence of outgoing transitions when $\texttt{rpos} \geq n^2 + i$ is ensured by the main loop condition (Line 26). Let x, y, z, σ, and R be as in the lemma statement. We also let $z'\sigma' = \triangleright\pi_1(y)$ with $|\sigma'| = 1$. Starting from the last position of $xy = z\sigma$ (thus scanning σ) with $\mathtt{p_{curr}} = p$ and $\texttt{rpos} = n^2 + i - 1$, C_i nondeterministically simulates a computational path of $\mathsf{A}_{/R}$ starting from configuration $z' \cdot p \cdot \sigma'$. Yet, C_i does not work on $z'\sigma'$ but on xy. Hence, in order to simulate $\mathsf{A}_{/R}$, C_i proceeds in two modes (mainly distinguished by whether $\texttt{rpos} \geq n^2$ or not, *c.f.* Line 27).

Mode 1. As far as it scans the portion containing y ($\texttt{rpos} \geq n^2$), it performs a direct simulation by reading the symbols from the input track. Since on this portion these symbols are distinct from $\triangleright$, the transition of $\mathsf{A}_{/R}$ are the same as those of A. Hence, C_i nondeterministically chooses one transition of A on the corresponding symbol, and then updates its state and moves its head accordingly (Lines 28–30).

Mode 2. As soon as the last position of the portion containing x is entered (from the right, detected as $\texttt{rpos} < n^2$), it simulates a transition of $\mathsf{A}_{/R}$ on $\triangleright$. If $x = \triangleright$ then $R = t^{\mathsf{A}}_{\varepsilon}$ and thus such transitions are the same as those of A. Hence the same simulation as in the previous case works (*c.f.* the second condition for entering the previous mode on Line 27). Otherwise, C_i scans backward the annotation track carrying $\pi_2(x)$ (Lines 32–39) in order to find some relative position $pn + q$ where p is the current value of $\mathtt{p_{curr}}$ (detected as $\mathtt{p_{curr}} \cdot n \leq \texttt{rpos} < (\mathtt{p_{curr}} + 1)n$ and such that $\pi_2(x[pn+q]) = 1$ (Line 33). Such a position indeed indicates that (p,q) belongs to R, or equivalently, that from state p scanning $\triangleright$,

$\mathsf{A}_{/R}$ may move its head rightward and enter q. Upon encountering such a position, C_i nondeterministically chooses either to ignore it or to select it (Line 34). In the former case it proceeds with the backward scan of $\pi_2(x)$, while in the latter it sets $\mathtt{p_{curr}}$ to q and moves the head rightward to relative position n^2 (Lines 38–39), so that the simulation may resume from there. If at some point $\mathtt{rpos} = 0$ and no position has been found and selected, then the machine aborts (Line 36).

By construction, C_i simulates $\mathsf{A}_{/R}$ on relative position less than $n^2 + i$. Next, its transitions are determined from the only information of $\mathtt{p_{curr}}$ that ranges over Q, $\mathtt{rpos}$ and the scanned symbol. Hence, C_i has n states (not counting the $\mathtt{rpos}$-component). ◀

Denote by C the disjoint union of C_i's over i, that is, the 2NFA implementing Procedure 5. A computation of C may fail for distinct reasons. First, it may follow a computational path of $\mathsf{A}_{/R}$ on $\pi_1(y)$ that gets stuck in some configuration with the head positioned on some symbol of $\triangleright\pi_1(y)$. This includes entering a configuration in which $\mathtt{rpos} = n^2 - 1$ (last position of x) with $\mathtt{p_{curr}}$ storing some state p for which $R(p) = \emptyset$. Second, from such a point, even if $R(p) \neq \emptyset$, C may choose none of the q of this set (by choosing $b = \mathsf{false}$ on Line 34) and finally end in abortion (on Line 36). Third, it may follow a computational path of $\mathsf{A}_{/R}$ on $\pi_1(y)$ that enters a loop. In such a case the simulation might loop forever. This last case is more problematic, as it cannot be easily detected. However, the following remark shows that, with a polynomial increase in size, it is possible to avoid such loops, while keeping the main property of the automaton.

▶ Remark 4.10 (About haltingness of C). Since, for every i, C_i works within the current window of length $2n^2$ and has n states, every computational path of C with more than $2n^3 - 1$ steps admits a loop (*i.e.*, visits twice the same configuration). Such a loop could have been cut, hence resulting in a shorter computational path with same starting and ending configurations. By adding a clock component to the finite control, as a variable ranging over $[2n^3]$, one may restrict computational paths of C to have length at most $2n^3 - 1$. This preserves all loop-free computational paths (and thus the main property of C) and ensures haltingness. Moreover, the size of the resulting $\mathtt{rpos}$-aware halting variant of C is still polynomial in n, namely, the number of states is in $\mathcal{O}(n^5)$.

In order to update the L-tables according to Proposition 4.5, we need to consider other relations, *e.g.* $S_{z\sigma}^{\mathsf{A}_{/R}\,j}$ for $j \in [n^2]$. The automaton C can be easily adapted so that the resulting 2NFA is able to find any computational path of $\mathsf{A}_{/R}$ witnessing the membership of a pair (p, q) to such relations.

▶ **Lemma 4.11.** *Let $i \in [n^2]$. There exists an $\mathtt{rpos}$-aware 2NFAs S_i with state set $Q \times [n^2]$ such that, for every $x \in \{\triangleright\} \cup \Psi^{n^2}$, $y \in \Psi^i$, $\sigma \in \Psi \cup \{\triangleleft\}$, $p, q \in Q$, and $j \in [n^2]$:*

$$xy \cdot (p, j)_{[\mathtt{rpos}=n^2+i]} \cdot \sigma \overset{\mathsf{S}_i}{\vdash}{}^{*} xy \cdot (q, 0)_{[\mathtt{rpos}=n^2+i]} \cdot \sigma \qquad \Longleftrightarrow \qquad (p, q) \in S_{\pi_1(y\sigma)}^{\mathsf{A}_{/R}\,j}$$

where R equals $t_\varepsilon^{\mathsf{A}}$ if $x = \triangleright$ and $\mathsf{dec}(\pi_2(x))$ otherwise. Furthermore, S_i forbids right moves from configurations in which $\mathtt{rpos} = n^2 + i$, and has no transition outgoing configurations in which $\mathtt{rpos} > n^2 + i$ or the state is $(q, 0)$ for some q; in particular the above configuration $xy \cdot (q, 0) \cdot \sigma$ is halting.

Proof. On input $xy\sigma$, based on Remark 4.3, an $\mathtt{rpos}$-aware 2NFA witnesses the membership of a pair to $T_{\pi_1(y\sigma)}^{\mathsf{A}_{/R}\,1}$ as follows. Starting from the relative position $n^2 + i$ (thus scanning σ), it simulates one left-move of $\mathsf{A}_{/R}$ over $\pi_1(\sigma)$, and then, using `nsimul_t` implemented by the automaton C_i from Lemma 4.9, it simulates an L-segment of $\mathsf{A}_{/R}$ on $\pi_1(y)$, thus ending

Procedure 6 enum_$t\,(m)$

```
(p_prev, q_prev) ← (−1, −1)
for k ← 1 to m do
    choose p_next in Q
    q_next ← nsimul_t(p_next)
    if (p_next, q_next) ≤ (p_prev, q_prev) then abort
    (p_prev, q_prev) ← (p_next, q_next)
    output (p_next, q_next)
```

Procedure 7 enum_$S\,(m,\,j)$

```
(p_prev, q_prev) ← (−1, −1)
for k ← 1 to m do
    choose p_next in Q
    q_next ← nsimul_S(p_next, j)
    if (p_next, q_next) ≤ (p_prev, q_prev) then abort
    (p_prev, q_prev) ← (p_next, q_next)
    output (p_next, q_next)
```

in relative position $|y\sigma|$. By repeating this process up to j times (using a counter ranging over $[j]$), it can witness membership of a pair to $S^{\mathsf{A}_{/R}\,j}_{\pi_1(y\sigma)}$. Only $\mathcal{O}(n^3)$ states are sufficient to implement this process, namely for storing the counter ranging over $[j] \subseteq [n^2]$, and the state of $\mathsf{A}_{/R}$ in the simulated computation. ◀

We let S be the disjoint union of S_i's over i. In subsequent procedures, the call to S is referred to as the procedure nsimul_S which takes two parameters, namely the starting state p and the value of j, and eventually returns a state $q \in S^{\mathsf{A}\,j}_{z\sigma}(p)$ when called from head position $|z\sigma|$.

Let Y be $t^{\mathsf{A}}_{v\tau}$ (resp. $S^{\mathsf{A}\,j}_{v\tau}$ for some j), and m denote the size of Y. Equipped with the procedure nsimul_t (resp. nsimul_S), a 2NFA can enumerate the elements of Y as soon as m is known. Indeed, in a similar way as Procedure 1 enumerates the elements of X^{A}_{v}, it is possible to repeat m calls to nsimul_t (resp. nsimul_S) and check that at each iteration but the first, the found pair is larger than the preceding one. The so-described enumeration procedure is named enum_t (resp. enum_S), and can be implemented by a rpos-aware 2NFA with $\mathcal{O}(n^8)$ (resp. $\mathcal{O}(n^{10})$) many states. In particular, enum_t has to store the value of the variable k for which it uses a state component of size at most n^2, and n-size state components for the values of p_{prev}, q_{prev}, p_{next}, and q_{next}. Notice that the variable q_{next} is not used during the execution of nsimul_t, and then overwritten with the value returned by nsimul_t, hence it is not needed to be stored in the finite control while executing nsimul_t. Therefore, that state component can be used to store the value of the internal state variable $\mathtt{p_{curr}}$ of nsimul_t. In addition, nsimul_t uses a state component of size n^2 for the variable i. So, summing up the sizes of the state components we end up with $\mathcal{O}(n^8)$ states. On the other hand, enum_S uses the same states as enum_t, plus a state component of size n^2 for the variable j.

4.2.1.1 Checking the annotation-track contents.

The above-presented procedures enum_t and enum_S, respectively, allow to enumerate the elements of $t^{\mathsf{A}_{/R}}_{\pi_1(y)}$ and $S^{\mathsf{A}_{/R}\,j}_{\pi_1(y)}$ for some y and j, where R is encoded over the annotation track on the left side of the current window (*i.e.*, on the cells at relative positions from 0 to n^2-1) which is scanned during the inner calls to nsimul_t and nsimul_S. In order to get membership to t^{A}_{v} and $S^{\mathsf{A}\,j}_{v}$ where v is the whole input-track contents to the left of the current position, we should ensure the correctness of R. More precisely, if $\rhd w \lhd$ is factorized as $\rhd w' x y w'' \lhd$ for some w' whose length is a multiple of n^2, some x of length n^2 such that $\mathsf{dec}(\pi_2(x)) = R = t^{\mathsf{A}}_{\pi_1(w'x)}$ some y of length n^2, and some w'', then B should check that $\mathsf{dec}(\pi_2(y)) = t^{\mathsf{A}}_{\pi_1(w'xy)} = t^{\mathsf{A}_{/R}}_{\pi_1(y)}$ before entering the first position of $w''\lhd$. Since, by induction hypothesis, the automaton will know $\#t^{\mathsf{A}}_{\pi_1(w'xy)}$ at that time (meaning that $\#t^{\mathsf{A}}_{\pi_1(w'xy)}$ will be stored in the finite control), it can enter a special mode to perform the check. The same idea as in the one-way case (see Procedure 2) can be used, namely, the automaton first checks that the annotation track contains exactly $\#t^{\mathsf{A}}_{\pi_1(w'xy)}$ many 1's and then, using the procedure enum_t, it checks that

Procedure 8 `get_S1_from_t(m)`

```
m_next ← n            // n for {(p,p) | p ∈ Q}
foreach (p,q) ∈ {(p,q) ∈ Q^2 | p ≠ q} do
    foreach (s,r) ∈ enum_t(m) do
        if r = q and (s,−1) ∈ δ(p, π_1(read())) then
            m_next ← m_next + 1
            break
return m_next;
```

Procedure 9 `get_t_from_S*(m)`

```
m_next ← 0
foreach (p,q) ∈ Q^2 do
    foreach (s,r) ∈ enum_S(m, n^2) do
        if s = p and (q,+1) ∈ δ(r, π_1(read())) then
            m_next ← m_next + 1
            break
return m_next;
```

the elements of $t^{\mathsf{A}}_{\pi_1(w'xy)}$ are exactly those annotated by 1 in the annotation track. This can be implemented by using in $\mathcal{O}(n^8)$ many states.

Although useless for the simulation, in order to fulfill the requirement of uniqueness of $\mathsf{annot}(w)$, when hitting the right endmarker B should check that the last r input cells were annotated by 0, where $r = |w| \mod n^2$ (*c.f.* Definition 4.8). This is easily done using `rpos`-awareness.

4.2.2 Inductively computing the size of L-tables.

We now explain how our 2NFA B inductively computes $\#t^{\mathsf{A}}_{\pi_1(v)}$ for each prefix v of w. More precisely, we inductively ensure the following invariant for every prefix v of w:

$$\begin{array}{c}\text{when } \mathsf{B} \text{ enters position } |v|+1 \text{ for the first time, } \#t^{\mathsf{A}}_v \text{ is stored in its finite control,}\\ \text{and } \pi_2(v') = \mathsf{enc}(t^{\mathsf{A}}_{\pi_1(v')}) \text{ for } v' \text{ the maximal prefix of } v \text{ such that } |v'| = 0 \mod n^2.\end{array} \quad (I_v)$$

Initially, for $v = \varepsilon$, $\#t^{\mathsf{A}}_v = \#t^{\mathsf{A}}_\varepsilon$ is a precomputed constant, encoded in the initial state of B. Let $v\sigma$ be a prefix of w with $|\sigma| = 1$. Assume that $\#t^{\mathsf{A}}_v$ is known (*i.e.*, stored in the finite control of B) and the head is on position $|v\sigma|$, scanning τ such that $\pi_1(\tau) = \sigma$. In order to compute $\#t^{\mathsf{A}}_{v\sigma}$, B follows the following steps:

Step 1. It computes $\#{S^{\mathsf{A}}_{v\sigma}}^1$ from $\#t^{\mathsf{A}}_v$ (Procedure 8);
Step 2. It then inductively computes $\#{S^{\mathsf{A}}_{v\sigma}}^*$ from $\#{S^{\mathsf{A}}_{v\sigma}}^1$ (Procedure 11, using Procedure 10);
Step 3. It finally computes $\#t^{\mathsf{A}}_{v\sigma}$ from $\#{S^{\mathsf{A}}_{v\sigma}}^*$ (Procedure 9).

Each of these steps is performed through a computational path of B that starts from position $|v\sigma|$, ends in position $|v\sigma|$, and visits only position $h \le |v\sigma|$ in the meantime.

Steps 1 and 3 are the simplest ones, as they follow from Remark 4.3 and Proposition 4.5, respectively. Indeed, for Step 1, B can detect whether a given pair (p,q) belongs to t^{A}_v given $\#t^{\mathsf{A}}_v$ using `member_t`. Thus, it can detect whether a pair belongs to ${S^{\mathsf{A}}_{v\sigma}}^1$ based on Remark 4.3, and, by trying all pairs and counting those that belong to ${S^{\mathsf{A}}_{v\sigma}}^1$, it can compute $\#{S^{\mathsf{A}}_{v\sigma}}^1$ (Procedure 8). Similarly for Step 3, B can detect whether a given pair (p,q) belongs to ${S^{\mathsf{A}}_{v\sigma}}^*$ given $\#{S^{\mathsf{A}}_{v\sigma}}^*$ using `member_S` with $j = n^2$ (so that ${S^{\mathsf{A}}_{v\sigma}}^j = {S^{\mathsf{A}}_{v\sigma}}^*$ by Proposition 4.4). Thus, it can detect whether a pair belongs to $t^{\mathsf{A}}_{v\sigma}$ based on Proposition 4.5, and by trying all pairs and counting those that belong to $t^{\mathsf{A}}_{v\sigma}$, it can compute $\#t^{\mathsf{A}}_{v\sigma}$ (Procedure 9). The resulting procedures `get_S1_from_t` and `get_t_from_S*` can be implemented using a polynomial number of states. Indeed, `member_t` (resp. `member_S`) is implemented using $f(n)$ states for some polynomial f, and thus `get_S1_from_t` (resp. `get_t_from_S*`) can be implemented using $\mathcal{O}(n^5 f(n))$ states (the $\mathcal{O}(n^5)$-size additional state component allows to store p, q, r, and m).

It thus remains to explain Step 2, which follows the same idea but with an inner induction. Indeed, one way of computing $\#{S^{\mathsf{A}}_{v\sigma}}^* = \#{S^{\mathsf{A}}_{v\sigma}}^{n^2}$ from $\#{S^{\mathsf{A}}_{v\sigma}}^1$ is to successively compute $\#{S^{\mathsf{A}}_{v\sigma}}^j$

Procedure 10 `get_next_S(m, j)`

```
m_next ← 0
foreach (p, q) ∈ Q^2 do
    foreach r ∈ Q do
        if member_S(p, r, m, j) and
          member_S(r, q, m, j) then
            m_next ← m_next + 1
            break
return m_next
```

Procedure 11 `get_S*_from_S1(m)`

```
m_next ← m
for j ← 1 to 2⌈log(n)⌉ do
    m_next ← get_next_S(m_next, 2^j)
return m_next
```

Procedure 12 `mainB()`

```
m ← #t_ε^A
while read() ≠ ◁ do
    m ← get_S1_from_t(m)
    m ← get_S*_from_S1(m)
    m ← get_t_from_S*(m)
    if rpos = 2n^2 − 1 then check_table(m)
    move the head to the right
if the suffix of length rpos − n^2 − 1 of π_2(w) is not in 0* then abort
m ← get_S1_from_t(m)
m ← get_S*_from_S1(m)
if enum_S(m, n^2) outputs (q_restart, q_f) then return true else return false
```

for $j = 1, \ldots, n^2$. However, we follow a cheaper and simpler big-step strategy, where we compute these cardinalities only for successive powers of 2, using the following observation:

$$(p, q) \in {S^{\mathsf{A}}_{v\sigma}}^{2j} \quad\iff\quad \exists r \in Q : \ (p, r) \in {S^{\mathsf{A}}_{v\sigma}}^{j} \text{ and } (r, q) \in {S^{\mathsf{A}}_{v\sigma}}^{j} \qquad \text{for all } j \geq 0. \tag{3}$$

Hence, since knowing $\#{S^{\mathsf{A}}_{v\sigma}}^{j}$ allows to detect which pairs of states belong to ${S^{\mathsf{A}}_{v\sigma}}^{j}$ using `member_S`, our automaton B is able to compute $\#{S^{\mathsf{A}}_{v\sigma}}^{2j}$ from $\#{S^{\mathsf{A}}_{v\sigma}}^{j}$. Just as `get_t_from_S*`, the resulting procedure `get_next_S` can be implemented using a polynomial number of states only (Procedure 10). By successively computing such cardinalities for $j = 1, \ldots, 2\lceil\log(n)\rceil$, we obtain $\#{S^{\mathsf{A}}_{v\sigma}}^{2^{2\lceil\log(n)\rceil}}$ which is equal to $\#{S^{\mathsf{A}}_{v\sigma}}^{*}$ by Proposition 4.4. The resulting procedure `get_S*_from_S1` (Procedure 11) can be implemented using a polynomial number of states in n only.

4.2.3 Gathering the mechanisms: the automaton B.

We are now ready to prove our main theorem, by concluding the presentation of B, whose high-level behavior is described by Procedure 12.

▶ **Lemma 4.12.** *Let A be an n-state 2NFA over Σ, and let $\Psi = \Sigma \times \{0, 1\}$. Then there exist a function $\mathsf{annot} : \Sigma^* \to \Psi^*$, and a 2NFA B of polynomial size in n[10] over Ψ, with two distinguished halting states q_{acc} and q_{rej} such that, on input $x \in \Psi^*$:*

- *B admits an initial computational path halting in q_{acc} if and only if $x = \mathsf{annot}(\pi_1(x))$ and $\pi_1(x) \in \mathcal{L}(\mathsf{A})$;*
- *B admits an initial computational path halting in q_{rej} if and only if $x = \mathsf{annot}(\pi_1(x))$ and $\pi_1(x) \notin \mathcal{L}(\mathsf{A})$.*

[10] In order to avoid technicalities, and because the polynomial order of our simulation is what matters here, we do not give a precise polynomial upper bound. A rough evaluation could show that $\mathcal{O}(n^{17}\log(n))$ states are sufficient.

Proof. The mapping annot is defined in Definition 4.8, while the 2NFA B implements Procedure 12 using and maintaining the `rpos` variable in its finite control, as explained in Section 4.2.1. We first show the correctness of B, based on the preliminary work, and then argue that its has polynomial size in n.

The behavior of B can be decomposed into two phases, one main loop for visiting all tape cells until hitting the right endmarker (Lines 81–86) while iteratively computing $\#t_v^{\mathsf{A}}$ for the corresponding prefix v, followed by a final phase deciding the acceptance or rejection of the input. In any phase, abortion of the computation, due to wrong choices or incorrect annotations are possible.

During the first phase, we ensure the invariant (I_v) where v is the input-track contents to the left of the head position, at the loop entrance. This prefix v is extended, symbol by symbol, at each iteration of the loop (see Line 86). Indeed, knowing $\#t_v^{\mathsf{A}}$ at the beginning of the loop with the head scanning $\tau \neq \lhd$ with $\pi_1(\tau) = \sigma$, the machine computes $\#t_{v\sigma}^{\mathsf{A}}$ by following the three steps that were presented in Section 4.2.2 (Lines 82–84). In order for the simulation to work properly, B checks the annotation track contents using the procedure `check_table` each time a factor of length n^2 have been completed (Line 85).

The second phase is entered when the head has reached the right endmarker. In order to ensure unicity of the annotation, it checks that the annotation-track contents ends with 0^r, where $r = |w| \mod n^2$ is known as $r = \texttt{rpos} - n^2 - 1$ (Line 87). Also, at that time, the state of B contains the information of $\#t_w^{\mathsf{A}}$. Hence, in a similar way as done during the loop, the machine can compute $\#S_{w\lhd}^{\mathsf{A}\,*}$ (Lines 88–89). It then decide acceptance by testing whether the pair $(q_{\mathsf{restart}}, q_{\mathsf{f}})$ belongs to $S_{w\lhd}^{\mathsf{A}\,*}$ or not according to Proposition 4.6, using `enum_`S with $j = n^2$ (Line 90).

The number of states of B is polynomial in n. This can easily be seen as the given procedure and sub-procedures use finitely many variables (including `rpos`), each ranging over at most n^2 values. Indeed a state of B roughly consists in a valuation of these variables, together with a mode specifying at which point (which procedure and which line) the simulation is. A rough analysis could show that $\mathcal{O}(n^{17}\log(n))$ states are sufficient (hopefully a finer analysis and/or design could decrease this exponent, possibly using further annotation symbols). ◀

Our main theorem then follows.

▶ **Theorem 4.13.** *Every n-state 2NFA has an equivalent self-verifying 2NFA+cg or 1-LA with a polynomial number of states in n, and 2 annotation symbols.*

5 Conclusion

We investigated the descriptional complexity of the complementation of 2NFAs. Our results show that, if we relax the target device so that, in addition to the usual nondeterminism it enjoys *common guess* (*i.e.*, it works on nondeterministically annotated inputs), then a polynomial cost can be met. Working on Remark 4.10 or applying the results from [4], the resulting 2NFA+cg can furthermore be made halting. A natural improvement of our result consists in a finer analysis of the upper-bounding polynomial, or –more promising– in the design of an alternative cheaper construction, possibly using more annotation symbols.

Another extension could be to consider more restricted forms of 2NFA+cgs as target devices. In particular, 2DFA+cgs are of particular interest. In contrast with 2NFA+cgs, their nondeterminism is limited to the annotation phase only. This limitation already allows to derive similar lower bounds for their simulation by 2NFAs, 1NFAs, 1DFAs, or deterministic 1-LAs (D1-LAs) as those obtained for the simulation of general 1-LAs [5]. A natural question is whether common guess is sufficient for capturing the usual form of nondeterminism up-to a polynomial size increase. This reduces to the question of the size cost of the conversion of 2NFAs into equivalent 2DFA+cgs, a weakening of [11, Problem 4]; see Figure 1. Hence, a future line of research consists in the investigation of these costs, as well as the related problem of the cost of complementing 2NFAs or 1NFAs by 2DFA+cgs.

References

1. Jean-Camille Birget. Intersection and union of regular languages and state complexity. *Information Processing Letters*, 43(4):185–190, September 1992. URL: http://dx.doi.org/10.1016/0020-0190(92)90198-5, doi:10.1016/0020-0190(92)90198-5.
2. Viliam Geffert, Bruno Guillon, and Giovanni Pighizzini. Two-way automata making choices only at the endmarkers. *Information and Computation*, 239:71–86, December 2014. URL: http://dx.doi.org/10.1016/j.ic.2014.08.009, doi:10.1016/j.ic.2014.08.009.
3. Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173–1187, August 2007. URL: http://dx.doi.org/10.1016/j.ic.2007.01.008, doi:10.1016/j.ic.2007.01.008.
4. Bruno Guillon and Luca Prigioniero. Linear-time limited automata. *Theoretical Computer Science*, 798:95–108, December 2019. URL: http://dx.doi.org/10.1016/j.tcs.2019.03.037, doi:10.1016/j.tcs.2019.03.037.
5. Bruno Guillon, Luca Prigioniero, and Javad Taheri. Nondeterminism makes unary 1-limited automata concise. apr 2025. URL: http://arxiv.org/abs/2504.08464v2, arXiv:2504.08464v2.
6. John E Hopcroft and Jeffrey D Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
7. Galina Jirásková. State complexity of some operations on binary regular languages. *Theoretical Computer Science*, 330(2):287–298, February 2005. URL: http://dx.doi.org/10.1016/j.tcs.2004.04.011, doi:10.1016/j.tcs.2004.04.011.
8. Christos Kapoutsis. *Removing Bidirectionality from Nondeterministic Finite Automata*, pages 544–555. Springer Berlin Heidelberg, 2005. URL: http://dx.doi.org/10.1007/11549345_47, doi:10.1007/11549345_47.
9. Nelma Moreira, Giovanni Pighizzini, and Rogério Reis. Optimal state reductions of automata with partially specified behaviors. *Theoretical Computer Science*, 658:235–245, January 2017. URL: http://dx.doi.org/10.1016/j.tcs.2016.05.002, doi:10.1016/j.tcs.2016.05.002.
10. Giovanni Pighizzini. Two-way finite automata: Old and recent results. *Fundam. Informaticae*, 126(2-3):225–246, 2013.
11. Giovanni Pighizzini. *Limited Automata: Properties, Complexity and Variants*, pages 57–73. Springer International Publishing, 2019. URL: http://dx.doi.org/10.1007/978-3-030-23247-4_4, doi:10.1007/978-3-030-23247-4_4.
12. Giovanni Pighizzini and Andrea Pisoni. Limited automata and regular languages. *International Journal of Foundations of Computer Science*, 25(07):897–916, November 2014. URL: http://dx.doi.org/10.1142/S0129054114400140, doi:10.1142/s0129054114400140.
13. Giovanni Pighizzini, Luca Prigioniero, and Simon Sádovský. Performing regular operations with 1-limited automata. *Theory Comput. Syst.*, 68(3):465–486, 2024.
14. William J Sakoda and Michael Sipser. Nondeterminism and the size of two way finite automata. *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 275–286, 1978.
15. J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, April 1959. URL: http://dx.doi.org/10.1147/rd.32.0198, doi:10.1147/rd.32.0198.

16 K Wagner and G Wechsung. *Computational Complexity*. Mathematics and its Applications. Springer, November 1986.