

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

Bruno Guillon   

Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Luca Prigioniero   

Department of Computer Science, Loughborough University, UK

Javad Taheri  

Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 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.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Models of computation; Theory of computation → Regular languages

Keywords and phrases descriptional complexity, inductive counting, common-guess

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.48

Related Version *Full Version*: <https://arxiv.org/abs/2507.11209> [7]

Acknowledgements The authors are very grateful to Giovanni Pighizzini for his useful comments and bibliographic help during the writing process.

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

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.



deterministic finite automata (1DFAs), because they admit a unique computational path on each input, and thus existential and universal quantifications on computational paths coincide. 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 transforming 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 [18, 1, 10].

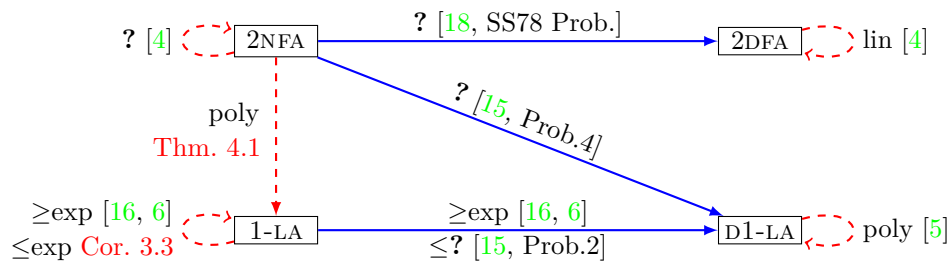
Yet, the corresponding question remains unsolved for other regular language recognizers, and in particular when dealing with *two-way finite automata*, an extension of finite automata allowing the machine to move its head both back and forth, and which still characterizes regular languages. Indeed, although complementing two-way deterministic automata (2DFAs) has been non-trivially² shown to cost linear only [4], 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*” [18] (see *e.g.* [14] for a survey). As shown in [4], 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* and *letter-bounded* settings [4, 2], or when the 2NFA makes a restricted use of nondeterminism, known as *outer-nondeterminism* [3].

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 by equipping them with some extra features while keeping the expressive power unchanged. More precisely, we use as target device a machine called *1-limited automaton* (1-LA), which is an extension of 2NFAs with some rewriting capability.

■ **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. The cost of complementing 2NFAs is an open problem related to the “the Sakoda and Sipser conjecture” [18] (abbreviated “SS78” in the table). The best-known upper bound for this problem is exponential and is derived from constructions that eliminate two-wayness, see, *e.g.*, [21, 12]. 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.

model	cost	model	cost
1DFA	trivial	1NFA	exp [18]
2DFA	linear [4]	2NFA	??? (related to SS78, <i>via</i> [4])
D1-LA	poly [5]	1-LA	exp (lower bound in [16, 6], and upper bound in Corollary 3.3)

² The main challenge when complementing 2DFAs is to detect infinite computations induced by loops.



■ **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” [18].

Technically, each time the machine visits a cell for the first time it is allowed to change its contents. This model is not more powerful than 2NFAs [22, Thm. 12.1], *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 [15]). 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) [17].

In the survey [15], the author identifies some problems regarding the descriptorial complexity of 1-LAS. In particular, the question of the cost of the conversion of 2NFAs into equivalent D1-LAS [15, Problem 4], as well as those of the cost of determinizing 1-LAS [15, 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 4.1). The property of being self-verifying means that the devices are able to recognize both the language and its complement (the formal definition is given in Section 2), see, *e.g.*, [11]. In both constructions, the resulting devices are 1-LAS of a 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 [5, 6] for details and results on this model. Although a polynomial simulation of 1NFAs by self-verifying 1-LAS is implied by Theorem 4.1, 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 [16], 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 [6], 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. In Section 2 we gather 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.*, [8]).

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 $[n] = \{0, \dots, n-1\}$; 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 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, $0 \leq i, j < |w|$. 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) \mathbf{A} is a tuple $\langle Q, \Sigma, \delta, q_{\text{start}}, q_{\text{f}} \rangle$, where Q is the finite set of states, Σ is the input alphabet, $q_{\text{start}} \in Q$ is the initial state, q_{f} is the final state,³ and $\delta : Q \times \Sigma_{\triangleright, \triangleleft} \rightarrow 2^{Q \times \{-1, +1\}}$ 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, \mathbf{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 \mathbf{A} is denoted by $\mathcal{L}(\mathbf{A})$.

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 $\delta : Q \times \Sigma \rightarrow 2^Q$ (that is, the endmarkers and the instruction for head direction are irrelevant).

The above models are all read-only machines. We now define *1-limited automata* (1-LAs, for short) that extend 2NFAs with a limited write ability. Just as 2NFAs, 1-LAs have a finite set of states, an initial and an accepting state, and work both ways on a tape that initially contains the input surrounded by the two endmarkers. However, they are allowed to replace the contents of each tape cell (except for those containing the endmarkers) when the head visits the cell for the first time – during subsequent visits to the cell, the contents cannot be changed any longer. Acceptance for 1-LAs is defined exactly as for 2NFAs, and the language accepted by a given 1-LA \mathbf{A} is denoted by $\mathcal{L}(\mathbf{A})$. It is known that 1-LAs recognize regular languages only [22, Thm 12.1].

³ 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.

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 $M = \langle A, \Sigma, \Gamma \rangle$ where Σ is the input alphabet, Γ is the annotation alphabet, and A is a 2NFA over the product alphabet $\Sigma \times \Gamma$.

The language recognized by M is:

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

In [5], 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 -state 1-LA. For a recent discussion on this model, we refer the reader to [6].

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 $\triangleright \cdot q_{\text{start}} \cdot w \triangleleft$, and an *accepting configuration* is a configuration of the form $\triangleright x \cdot q_f \triangleleft$. The transition relation between configurations of M is denoted by $\stackrel{M}{\rightarrow}$, and its reflexive-transitive closure by $\stackrel{M}{\rightarrow}^*$. A *computational path* of M is a sequence of configurations c_0, \dots, c_r such that $c_i \stackrel{M}{\rightarrow} c_{i+1}$ 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 $\stackrel{M}{\rightarrow}$ and $\stackrel{M}{\rightarrow}^*$, whence the notion of computational path, naturally extend onto partial configurations.

Self-verifyingness. A nondeterministic state machine M is said to be *self-verifying* if it has two different final states, one *accepting* and one *rejecting*, and satisfies the following property. Each string must have at least one computational path ending in a final state, and cannot admit both an *accepting* (i.e., ending in the accepting state) and a *rejecting* (i.e., ending in the rejecting state) computational path. As a consequence the set of words recognized by M is partitioned into those admitting an accepting computational path, and those admitting a rejecting computational path. Observe that it is still possible to have computational paths that halt in non-final states; they are said *aborted*. The set of words admitting an accepting computational path is denoted as $\mathcal{L}(M)$, and the set of words admitting a rejecting computational path is its complement.

Procedure 1 `enum_X(m)`.

```

1  $q_{\text{prev}} \leftarrow -1$ 
2 for  $i \leftarrow 1$  to  $m$  do
3    $q_{\text{next}} \leftarrow \text{nsimul\_A}()$ 
4   if  $q_{\text{next}} \leq q_{\text{prev}}$  then abort
5    $q_{\text{prev}} \leftarrow q_{\text{next}}$ 
6   output  $q_{\text{next}}$ 

```

Procedure 2 `check_annot(m)`.

```

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

```

Procedure 3 `member_X(q_t, m)`.

```

14 foreach  $q \in \text{enum\_X}(m)$  do
15   if  $q = q_t$  then return true
16 return false

```

Procedure 4 `count_next_X(m)`.

```

17  $m_{\text{next}} \leftarrow 0$ 
18 foreach  $p \in Q$  do
19   foreach  $r \in \text{enum\_X}(m)$  do
20     if  $p \in \delta(x, \text{read}())$  then
21        $m_{\text{next}} \leftarrow m_{\text{next}} + 1$ 
22       break
23 return  $m_{\text{next}}$ 

```

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 using 2NFAs with common guess

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 [4, 3], our simulation is based on the *inductive counting* technique, which originates from the proof of the well-celebrated Immerman–Szelepcsényi theorem [9, 20].

Let $A = \langle Q, \Sigma, \delta, q_{\text{start}}, q_f \rangle$ be an n -state 1NFA. We assume $Q = [n]$ which is thus equipped with the natural ordering. For $w \in \Sigma^*$, we define X_w^A to be the set of states the automaton A reaches after reading w , *i.e.*, $X_w^A = \delta(q_{\text{start}}, w)$. Working on w , a 2NFA can iteratively simulate nondeterministic computation paths of A by bringing the head back to the left endmarker before starting a new 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.*, $m = \#X_w^A$. 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 all the m states of X_w^A . This process is described in Procedure 1, where `nsimul_A` stands for the nondeterministic simulation of A from the initial configuration.⁴ Hence, according to whether one or none of them is the accepting state, the 2NFA recognizes whether w belongs to $\mathcal{L}(A)$ 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 (q_{prev}), and the state in the current simulation (q_{next}). One strategy to detect whether an input w belongs to the complement of $\mathcal{L}(A)$ would therefore be to first compute $m = \#X_w^A$, and then run the 2NFA B_m . Yet, computing $\#X_w^A$ is not an easy task.

⁴ 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.

Suppose now that $\#X_w^A$ 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 $q \in X_u^A$. Then a 2NFA equipped with `nsimul_A` may inductively compute $\#X_u^A$ for each successive prefix u of w as follows. Initially, $\#X_\varepsilon^A = 1$ since $X_\varepsilon^A = \{q_{\text{start}}\}$ by definition. Let $u\sigma$ be a prefix of w . In order to compute $\#X_{u\sigma}^A$ from $\#X_u^A$, the automaton tests for each state p whether it belongs to $X_{u\sigma}^A$, 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 $X_{u\sigma}^A$ is based on the following basic observation:

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

Using the knowledge of $m = \#X_u^A$ 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 $p \in X_{u\sigma}^A$ 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 $p \notin X_{u\sigma}^A$ has been obtained. Once the number of states in $X_{u\sigma}^A$ 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 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 $X_{w[0, i \cdot n - 1]}^A$. 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 $X_{\pi_1(x_1 \cdots x_i)}^A = X_{w[0, i \cdot n - 1]}^A$. 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 $p \in X_{w[0, i \cdot n - 1]}^A$ – for $i = 0$, we assume $x_i = \triangleright$ from which the machine extracts $p = q_{\text{start}}$. 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 $X_{w[0, i \cdot n + j - 1]}^A$ since $p \in X_{w[0, i \cdot n - 1]}^A = \delta(q_{\text{start}}, w[0, i \cdot n - 1])$.

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

$$\text{enc}(S)[p] = 1 \quad \iff \quad p \in S.$$

Hence, provided $\pi_2(x_i) = \text{enc}(X_{w[0, i \cdot n]}^A)$, in order for `nsimul_A` to select $p \in X_{w[0, i \cdot n]}^A$ 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 $y = \text{enc}(X_w^A)$,
- if $r > 0$ then $\mathbf{a}_w = zy$, where $z = \mathbf{a}_{w[0, kn]}$, and $y = 0^r$.

The word $\text{annot}(w)$ 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 \mathbf{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:

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

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

The design of \mathbf{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, \mathbf{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 `nsimul_A`, the latter information is an integer less than $2n$.

Not only \mathbf{B} has to behave correctly on inputs $\text{annot}(w)$: it also has to detect ill-formed inputs, namely words $x \in \Psi^*$ for which $x \neq \text{annot}(\pi_1(x))$. To this end, each time the length of the prefix z under consideration is a positive multiple of n , before considering the next prefix, \mathbf{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 $m = \#X_{\pi_1(z)}^A$ has been computed. Hence, the automaton can check that $\pi_2(x_i)$ has m occurrences of 1, and, using `enum_X`, that $\pi_2(x_i[p]) = 1$ for each $p \in X_{\pi_1(z)}^A$.

Evaluating the size of \mathbf{B} , we get that $\mathcal{O}(n^3)$ states are sufficient for implementing `nsimul_A`, including the two above-mentioned state components relative to the head position. Next, \mathbf{B} further uses a $\mathcal{O}(n^4)$ -state component for storing $\mathbf{m} = \#X_u^A$, $\mathbf{m}_{\text{next}} \leq \#X_{u\tau}^A$, and two intermediate states \mathbf{q} and \mathbf{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 an exponential number of states in n and $3\#\Sigma$ work symbols which recognizes the complement of L .

Proof. From [16, 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 [6], 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 is *deterministic*.⁵

4 Complementing 2NFAs using 2NFAs with common guess

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 2 annotation symbols.

► **Theorem 4.1.** *Every n -state 2NFA has an equivalent self-verifying 2NFA+cg or 1-LA with a polynomial number of states in n , and annotation alphabet $\{0, 1\}$.*

The self-verifyingness of the obtained 2NFA+cg M is actually implied by a stronger condition which is satisfied by its underlying 2NFA B over $\Psi = \Sigma \times \{0, 1\}$. This is stated in the following lemma whose proof is our main technical contribution.

► **Lemma 4.2.** *Let A be an n -state 2NFA over Σ , and let $\Psi = \Sigma \times \{0, 1\}$. Then there exist a function $\text{annot} : \Sigma^* \rightarrow \Psi^*$, and a 2NFA B of polynomial size in n over Ψ , with two distinguished halting states q_{acc} and q_{rej} such that, on input $x \in \Psi^*$:*

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

Notice that, in the above lemma, B itself is not a self-verifying 2NFA. Indeed, on inputs not belonging to $\text{annot}(\Sigma^*)$, B neither accepts nor rejects.⁶ Yet, as the goal is to check whether a word $w \in \Sigma^*$ belongs to $\mathcal{L}(A)$, and since each such word has an annotated version $\text{annot}(w)$ that is either accepted or rejected by B , we obtain self-verifyingness for the 2NFA+cg $M = \langle B, \Sigma, \{0, 1\} \rangle$.

The rest of the section is devoted to the proof of Lemma 4.2, and is structured as follows. In Section 4.1, we recall the principle of Shepherdson’s construction [19], introducing the basic concepts and properties on which such a simulation as much as ours relies. The definition of $\text{annot}(w)$ is given at the end of the section. The automaton B is then designed in Section 4.2, where the proof of Lemma 4.2 is finally completed.

4.1 L-tables

In [19], 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.*, [12, 16]. 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*,⁷ 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 $\triangleright uv \triangleleft$ for some $v \in \Sigma^*$ that starts from tape position $|u|$ (hence reading the last symbol of $\triangleright u$) and ends in position $|u| + 1$ (hence, reading the first symbol of $v \triangleleft$) 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:

$$t_u^A = \{(p, q) \in Q^2 \mid x \cdot p \cdot \sigma \stackrel{A^*}{\vdash} \triangleright u \cdot q\}, \quad \text{where } x\sigma = \triangleright u \text{ with } |\sigma| = 1.$$

⁵ Remark that 2DFA+cg’s are nondeterministic devices, since the annotation phase is nondeterministic.

⁶ B is actually a *don’t-care* automaton [13], namely an automaton with the self-verifying property on inputs from a restricted domain (here, inputs of the form $\text{annot}(w)$ for some $w \in \Sigma^*$).

⁷ L for “Left”; In the literature, they have sometimes been called “*transition tables*”.

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, $X_u^A = \{q \in Q \mid \triangleright \cdot q_{\text{start}} \cdot u \stackrel{A}{\vdash} \triangleright u \cdot q\}$. 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 $u \in \mathcal{L}(A)$. In order to simplify, we slightly modify the simulated 2NFA, so that knowing t_u^A will be sufficient.

► **Lemma 4.3.** *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 $(q_{\text{restart}}, p) \in t_u^{A'}$ if and only if $p \in X_u^{A'}$, and on the other hand $(p, q_{\text{restart}}) \notin t_u^{A'}$. Furthermore, $\delta'(q_{\text{restart}}, \triangleleft) = \{(q_{\text{restart}}, -1)\}$ where δ' is the transition function of A' .*

The key of Shepherdson's construction is that, given t_u^A and $\tau \in \Sigma$, it is possible to compute $t_{u\tau}^A$ 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 $T_{u\tau}^{A^k}$ (resp. $S_{u\tau}^{A^k}$) 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). Formally: $T_{u\tau}^{A^0} = \{(p, p) \mid p \in Q\}$ and

$$T_{u\tau}^{A^{k+1}} = \left\{ (p, q) \mid \exists s, r \in Q \text{ such that } (p, s) \in T_{u\tau}^{A^k}, (r, -1) \in \delta(s, \tau) \text{ and } (r, q) \in t_u^A \right\},$$

and $S_{u\tau}^{A^k} = \bigcup_{j=0}^k T_{u\tau}^{A^j}$. We furthermore let $S_{u\tau}^{A^*}$ denote the relation $\bigcup_{j \in \mathbb{N}} T_{u\tau}^{A^j}$.

The following directly follow from the above definitions:

- **Remark 4.4.** Given a 2NFA A , $u \in \Sigma^*$ and $\tau \in \Sigma \cup \{\triangleleft\}$:
1. $S_{u\tau}^{A^*}$ is the reflexive and transitive closure of $T_{u\tau}^{A^1}$;
 2. $(p, q) \in T_{u\tau}^{A^1}$ if and only if there exists $r \in Q$ such that $(r, q) \in t_u^A$ and $(r, -1) \in \delta(p, \tau)$;
 3. for every $j \geq n(n - 1)$, it holds $S_{u\tau}^{A^*} = S_{u\tau}^{A^j}$.

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

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

A direct consequence of the above proposition is that $t_u^A = t_v^A$ implies $t_{uw}^A = t_{vw}^A$ for every w . Also, provided A is in the form of Lemma 4.3, we can decide whether $w \in \mathcal{L}(A)$ with the only information of $S_{w\triangleleft}^{A^*}$, which is determined from t_w^A thanks to Remark 4.4:

► **Proposition 4.6.** *Let A be a 2NFA over Σ in the form of Lemma 4.3, and $w \in \Sigma^*$. Then $w \in \mathcal{L}(A)$ if and only if $(q_{\text{restart}}, q_f) \in S_{w\triangleleft}^{A^*}$ where q_f is the accepting state of 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 A/R of A so that $t_{\varepsilon}^{A/R} = R$.

► **Definition 4.7.** *Let A be a 2NFA with state set Q , and let $R \subseteq Q^2$. We define A/R as the 2NFA obtained from A by overwriting its transitions on \triangleright according to R . More precisely, 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, $t_{\varepsilon}^{A/R} = R$ and $A/t_{\varepsilon}^A = A$. Also, if $R = t_u^A$ for some u then $t_v^{A/R} = t_{uv}^A$ 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 4.3 is not needed in our presentation) in its finite control. Then, on symbol τ , it updates it to $t_{u\tau}^A$ 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, e.g., [12]). 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 successively store the number of elements belonging to the L-tables as a state component, and encode *some* of these tables *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 $\text{enc}(R)$ over $\{0, 1\}$, as follows:

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

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., $\text{dec} = \text{enc}^{-1}$.

► **Definition 4.8.** For $w \in \Sigma^*$, the *annotation of w* is the word $\mathbf{a}_w \in \{0, 1\}^*$ 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 $\mathbf{a}_w = \varepsilon$, otherwise,
- If $r = 0$ then $\mathbf{a}_w = zy$ where $z = \mathbf{a}_{w[0, (k-1)n^2]}$ and $y = \text{enc}(t_w^A)$,
- If $r > 0$ then $\mathbf{a}_w = zy$ where $z = \mathbf{a}_{w[0, kn^2]}$ and $y = 0^r$.

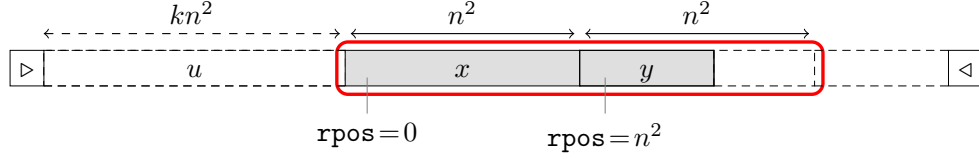
The word $\text{annot}(w)$ is the word x over $\Psi = \Sigma \times \{0, 1\}$ such that $\pi_1(x) = w$ and $\pi_2(x) = \mathbf{a}_w$.

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 [5] for proving a polynomial upper bound for the simulation of 1-LAS by halting 2NFA+cg. 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 [4, 3], 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 $\mathcal{L}(A)$ or not.

4.2.1 Simulating L-segments using annotations

The automaton B maintains a variable \mathbf{rpos} ranging over $[2n^2]$ in its finite control, which is updated according to each head move as now explained. The variable \mathbf{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 $\mathbf{rpos} = 0$



■ **Figure 2** The virtual window when the input prefix under consideration is $uxy \in \Psi^*$ with $|u| = kn^2$ for some $k \geq 0$, $|x| = n^2$ and $|y| = r < n^2$. The automaton already has checked $\pi_2(x) = \text{enc}(t_{\pi_1(ux)}^A)$ and can thus simulate L-segments of A with respect to $\pi_1(uxy)$ without exiting the window.

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, $\text{rpos} = n^2$. 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 $\max(0, h - \text{rpos})$ to position $\min(\ell + 1, h - \text{rpos} + 2n^2 - 1)$ where ℓ is the input length; see Figure 2. 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 $\max(0, h - \text{rpos}) + i$. 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, $q_{[\text{rpos}=i]}$ indicates state q with $\text{rpos} = i$.

Our construction based on inductive counting relies on 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 $\text{annot}(w)$, 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).

► **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 $p, q \in Q$:*

$$z \cdot p_{[\text{rpos}=n^2+i-1]} \cdot \sigma \stackrel{C_i}{\vdash} z\sigma \cdot q_{[\text{rpos}=n^2+i]} \iff (p, q) \in t_{\pi_1(y\sigma)}^{A/R}$$

where $z\sigma = xy$ with $|\sigma| = 1$, and R equals t_ε^A if $x = \triangleright$ or $\text{dec}(\pi_2(x))$ otherwise.

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 p_{curr} ranging over Q . The states of C_i are the valuations of this variable. The absence of outgoing transitions when $\text{rpos} \geq n^2 + i$ 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 $\text{p}_{\text{curr}} = p$ and $\text{rpos} = n^2 + i - 1$, C_i nondeterministically simulates a computational path of 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 A/R , C_i proceeds in two modes (mainly distinguished by whether $\text{rpos} \geq n^2$ or not, *c.f.* Line 27).

■ **Procedure 5** $\text{nsimul_t}(p)$.

```

24  $i \leftarrow \text{rpos} - n^2 + 1$ 
25  $p_{\text{curr}} \leftarrow p$ 
26 while  $\text{rpos} < n^2 + i$  do
27   if  $\text{rpos} \geq n^2$  or  $\text{read}() = \triangleright$  then
28     // direct simulation of A reading the input track
29     choose  $(r, d) \in \delta(p_{\text{curr}}, \pi_1(\text{read}()))$ 
30      $p_{\text{curr}} \leftarrow r$ 
31     move the head according to  $d$ 
32   else
33     // recovering L-segment from the annotation track
34     while true do
35       if  $p_{\text{curr}} \cdot n \leq \text{rpos} < (p_{\text{curr}} + 1)n$  and  $\pi_2(\text{read}()) = 1$  then
36         choose  $b \in \{\text{true}, \text{false}\}$ 
37         if  $b$  then break
38       if  $\text{rpos} = 0$  then abort
39       move the head leftward
40      $p_{\text{curr}} \leftarrow \text{rpos} \bmod n$ 
41     move the head rightward to relative position  $n^2$ 
42 return  $p_{\text{curr}}$ 

```

Mode 1 As far as it scans the portion containing y ($\text{rpos} \geq n^2$), 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 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 $\text{rpos} < n^2$), it simulates a transition of A/R on \triangleright . If $x = \triangleright$ then $R = t_\varepsilon^A$ 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 p_{curr} (detected as $p_{\text{curr}} \cdot n \leq \text{rpos} < (p_{\text{curr}} + 1)n$) 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 , 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 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 $\text{rpos} = 0$ and no position has been found and selected, then the machine aborts (Line 36).

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

In order to update the L-tables according to Proposition 4.5, we need to consider other relations, *e.g.* $S_{2\sigma}^{A/R^j}$ 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 A/R witnessing the membership of a pair (p, q) to such relations.

► **Lemma 4.10.** *Let $i \in [n^2]$. There exists an 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 $j \in [n^2]$:*

$$xy \cdot (p, j)_{[\text{rpos}=n^2+i]} \cdot \sigma \stackrel{S_i^*}{\vdash} xy \cdot (q, 0)_{[\text{rpos}=n^2+i]} \cdot \sigma \iff (p, q) \in S_{\pi_1(y)\sigma}^{A/R^j}$$

where R equals t_ε^A if $x = \triangleright$ and $\text{dec}(\pi_2(x))$ otherwise.

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 $q \in S_{z\sigma}^A(j)(p)$ when called from head position $|z\sigma|$.

Let Y be $t_{v\tau}^A$ (resp. $S_{v\tau}^A(j)$ 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_v^A , 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 polynomially many states in n .

Checking the annotation-track contents. The whole machinery above relies on the procedure `nsimul_t`, which in turn relies on the correctness of the annotation on the left side of the current window. Hence, as in Section 3, we have to check that the annotation track contents do encode the expected L-tables. Because our strategy requires to compute the size m of $t_{\pi_1(v)}^A$ for the successive prefixes v of the correctly-annotated input, our automaton B can trigger a special mode, on each time the length of such a prefix v is a positive multiple of n^2 , just after having computed $m = \#t_{\pi_1(v)}^A$. This special mode is responsible of checking that the n^2 -length suffix y of v carries the correct annotation symbols, namely, that $\pi_2(y) = \text{enc}(t_{\pi_1(y)}^A)$. (Technically, it checks that $\pi_2(y) = t_{\pi_1(y)}^{A/R}$ where $R = t_\varepsilon^A$ if $v = y$ or $R = \text{dec}(\pi_2(x))$ if $v = zxy$ for some $|x| = n^2$. Since $\pi_2(x) = R = t_{\pi_1(zx)}^A$ is assumed to have already been checked by induction, it follows $\pi_2(y) = t_{\pi_1(v)}^A$.) Because at the time this special mode is entered $m = \#t_{\pi_1(v)}^A$ is known (*i.e.*, is stored within the finite control of B), the verification can be easily obtained as follows. First the device checks that $\pi_2(y)$ has exactly m bits equal to 1. Second, it enumerates the pair $(p, q) \in t_v^A$ using `enum_t` and checks for each of them that $\pi_2(y)[p + qn] = 1$. It is routine to implement this procedure, to which we refer as `check_table`, by a 2NFA of size polynomial in n .

► **Lemma 4.11.** *There exists an `rpos`-aware 2NFA T with polynomially many states in n , and two injections `start` and `end` from $[n^2]$ to the states of T such that, for every $x \in \{\triangleright\} \cup \Psi^{n^2}$, $y \in \Psi^{n^2-1}$, and $\sigma \in \Psi$:*

$$xy \cdot \text{start}(m) \cdot \sigma \stackrel{T^*}{=} xy \cdot \text{end}(m) \cdot \sigma \quad \iff \quad \pi_2(y\sigma) = \text{enc}(t_{\pi_1(y\sigma)}^{A/R})$$

where R equals t_ε^A if $x = \triangleright$ and $\text{dec}(\pi_2(x))$ otherwise, and $m = \#t_{\pi_1(y\sigma)}^{A/R}$.

4.2.2 Inductively computing the size of L-tables

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

$$\begin{aligned} & \text{when } B \text{ enters position } |v| + 1 \text{ for the first time,} \\ & \#t_{\pi_1(v)}^A \text{ is stored in its finite control, and, if } |v| \geq n^2 \text{ then } \pi_2(v') = \text{enc}(t_{\pi_1(v')}^A) \quad (I_v) \\ & \text{for } v' \text{ the maximal prefix of } v \text{ whose length is a positive multiple of } n^2. \end{aligned}$$

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

Step 1 It computes $\#S_{\pi_1(v\sigma)}^{A^1}$ from $\#t_{\pi_1(v)}^A$;

Step 2 It then inductively computes $\#S_{\pi_1(v\sigma)}^{A^*}$ from $\#S_{\pi_1(v\sigma)}^{A^1}$;

Step 3 It finally computes $\#t_{\pi_1(v\sigma)}^A$ from $\#S_{\pi_1(v\sigma)}^{A^*}$.

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 $h \leq |v\sigma|$ in the meantime.

Steps 1 and 3 are the simplest ones, as they follow from Remark 4.4 and Proposition 4.5, respectively. Indeed, for Step 1, **B** can detect whether a given pair (p, q) belongs to $t_{\pi_1(v)}^A$ given $\#t_{\pi_1(v)}^A$ by enumerating the elements of $t_{\pi_1(v)}^A$ using `enum_t` (using the same idea as in Procedure 3). Thus, it can detect whether a pair belongs to $S_{\pi_1(v\sigma)}^{A^1}$ based on Item 2 of Remark 4.4, and, by trying all pairs and counting those that belong to $S_{\pi_1(v\sigma)}^{A^1}$, it can compute $\#S_{\pi_1(v\sigma)}^{A^1}$. Similarly for Step 3, **B** can detect whether a given pair (p, q) belongs to $S_{\pi_1(v\sigma)}^{A^*}$ given $\#S_{\pi_1(v\sigma)}^{A^*}$ by enumerating the elements of $S_{\pi_1(v\sigma)}^{A^*}$ using `enum_S` with $j = n^2$ (so that $S_{\pi_1(v\sigma)}^{A^j} = S_{\pi_1(v\sigma)}^{A^*}$ by Item 3 of Remark 4.4). Thus, it can detect whether a pair belongs to $t_{\pi_1(v\sigma)}^A$ based on Proposition 4.5, and by trying all pairs and counting those that belong to $t_{\pi_1(v\sigma)}^A$, it can compute $\#t_{\pi_1(v\sigma)}^A$. The resulting procedures `get_S1_from_t` and `get_t_from_S*` can be implemented using a polynomial number of states (as stated in Lemma 4.12 below).

It thus remains to explain Step 2, which follows the same idea but with an inner induction. Indeed, one way of computing $\#S_{\pi_1(v\sigma)}^{A^*} = \#S_{\pi_1(v\sigma)}^{A^{n^2}}$ from $\#S_{\pi_1(v\sigma)}^{A^1}$ is to successively compute $\#S_{\pi_1(v\sigma)}^{A^j}$ for $j = 1, \dots, n^2$. 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:

$$(p, q) \in S_{\pi_1(v\sigma)}^{A^{2^j}} \iff \exists r \in Q: (p, r) \in S_{\pi_1(v\sigma)}^{A^j} \text{ and } (r, q) \in S_{\pi_1(v\sigma)}^{A^j} \quad \text{for all } j \geq 0.$$

Hence, since knowing $\#S_{\pi_1(v\sigma)}^{A^j}$ allows to detect which pairs of states belong to $S_{\pi_1(v\sigma)}^{A^j}$ using `enum_S`, our automaton **B** is able to compute $\#S_{\pi_1(v\sigma)}^{A^{2^j}}$ from $\#S_{\pi_1(v\sigma)}^{A^j}$. The resulting procedure `get_next_S` (Procedure 6) can be implemented using a polynomial number of states only. By successively computing such cardinalities for $j = 1, \dots, 2\lceil \log(n) \rceil$, we obtain $\#S_{\pi_1(v\sigma)}^{A^{2^{2\lceil \log(n) \rceil}}}$ which is equal to $\#S_{\pi_1(v\sigma)}^{A^*}$ by Remark 4.4. The resulting procedure `get_S*_from_S1` (Procedure 7) can be implemented using a polynomial number of states in n only.

► **Lemma 4.12.** *Let $i \in [n^2]$. There exist three rpos-aware 2NFAs D_i, E_i, F_i of size polynomial in n , each provided with two mappings `start` and `end` from $[n^2]$ to their respective set of states, such that for every $x \in \{\triangleright\} \cup \Psi^{n^2}$, $y \in \Psi^i$, $\sigma \in \Psi \cup \{\triangleleft\}$, and $m, m' \in [n^2]$:*

$$\begin{aligned} m = \#t_{\pi_1(y)}^{A/R} &\iff (xy \cdot \text{start}(m) \cdot \sigma \stackrel{D_i^*}{=} xy \cdot \text{end}(m') \cdot \sigma \iff m' = \#S_{\pi_1(y\sigma)}^{A/R^1}) \\ m = \#S_{\pi_1(y\sigma)}^{A/R^1} &\iff (xy \cdot \text{start}(m) \cdot \sigma \stackrel{E_i^*}{=} xy \cdot \text{end}(m') \cdot \sigma \iff m' = \#S_{\pi_1(y\sigma)}^{A/R^*}) \\ m = \#S_{\pi_1(y\sigma)}^{A/R^*} &\iff (xy \cdot \text{start}(m) \cdot \sigma \stackrel{F_i^*}{=} xy \cdot \text{end}(m') \cdot \sigma \iff m' = \#t_{\pi_1(y\sigma)}^{A/R}) \end{aligned}$$

where R equals t_ε^A if $x = \triangleright$ and $\text{dec}(\pi_2(x))$ otherwise.

4.2.3 Gathering the mechanisms: the automaton **B**

We are now ready to prove Lemma 4.2 by concluding the presentation of **B** whose high-level behavior is described by Procedure 8.

Procedure 6 `get_next_S(m, j)`.

```

41  $m_{\text{next}} \leftarrow 0$ 
42 foreach  $(p, q) \in Q^2$  do
43   foreach  $r \in Q$  do
44     if  $(p, r) \in \text{enum\_S}(m, j)$  and
45        $(r, q) \in \text{enum\_S}(m, j)$  then
46        $m_{\text{next}} \leftarrow m_{\text{next}} + 1$ 
47       break
47 return  $m_{\text{next}}$ 

```

Procedure 7 `get_S*_from_S1(m)`.

```

48  $m_{\text{next}} \leftarrow m$ 
49 for  $j \leftarrow 1$  to  $2 \lceil \log(n) \rceil$  do
50    $m_{\text{next}} \leftarrow \text{get\_next\_S}(m_{\text{next}}, 2^j)$ 
51 return  $m_{\text{next}}$ 

```

Procedure 8 `mainB()`.

```

52  $m \leftarrow \#t_\epsilon^A$ 
53 while  $\text{read}() \neq \triangleleft$  do
54    $m \leftarrow \text{get\_S1\_from\_t}(m)$ 
55    $m \leftarrow \text{get\_S*\_from\_S1}(m)$ 
56    $m \leftarrow \text{get\_t\_from\_S*}(m)$ 
57   if  $\text{rpos} = 2n^2 - 1$  then check_table(m)
58   move the head to the right
59 if the suffix of length  $\text{rpos} - n^2 - 1$  of  $\pi_2(w)$  is
60   not in  $0^*$  then abort
60  $m \leftarrow \text{get\_S1\_from\_t}(m)$ 
61  $m \leftarrow \text{get\_S*\_from\_S1}(m)$ 
62 if enum_S(m, n2) outputs  $(q_{\text{restart}}, q_f)$  then
63   return true
64 else
65   return false

```

Proof of Lemma 4.2. The mapping `annot` is defined in Definition 4.8, while the 2NFA **B** implements Procedure 8 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. First, it has one main loop for visiting all tape cells until hitting the right endmarker (Lines 53–58) while iteratively computing $\#t_v^A$ for the corresponding prefix v . Then, a final phase decides the acceptance or the rejection of the input. In any phase, abortion of the computation, due to wrong choices or incorrect annotation 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 58). Indeed, knowing $\#t_v^A$ at the beginning of the loop with the head scanning $\tau \neq \triangleleft$ with $\pi_1(\tau) = \sigma$, the machine computes $\#t_{v\sigma}^A$ by following the three steps that were presented in Section 4.2.2 (Lines 54–56). 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 57).

The second phase is entered when the head has reached \triangleleft . To ensure unicity of the annotation, **B** checks that the annotation-track contents ends with 0^r , where $r = |w| \bmod n^2$ is known as $r = \text{rpos} - n^2 - 1$ (Line 59). Also, at that time, the state of **B** contains the information of $\#t_w^A$. Hence, in a similar way as done during the loop, the machine can compute $\#S_{w\triangleleft}^{A*}$ (Lines 60–61). It then decide acceptance by testing whether the pair $(q_{\text{restart}}, q_f)$ belongs to $S_{w\triangleleft}^{A*}$ or not according to Proposition 4.6, using `enum_S` with $j = n^2$ (Line 62).

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 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). ◀

5 Conclusion

We investigated the descriptonal 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. By adapting our construction or applying the results from [5], the resulting $2\text{NFA}+\text{cg}$ 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. In the opposite direction, deriving non-trivial (*e.g.*, quadratic) upper bounds for the transformation is left as a challenging open problem.

Another extension could be to consider more restricted forms of $2\text{NFA}+\text{cgs}$ as target devices. In particular, $2\text{DFA}+\text{cgs}$ are of particular interest. In contrast with $2\text{NFA}+\text{cgs}$, 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 [6]. 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 $2\text{DFA}+\text{cgs}$, a weakening of [15, 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 $2\text{DFA}+\text{cgs}$.

References

- 1 Jean-Camille Birget. Intersection and union of regular languages and state complexity. *Information Processing Letters*, 43(4):185–190, 1992. doi:10.1016/0020-0190(92)90198-5.
- 2 Alessandro Clerici Lorenzini, Giovanni Pighizzini, and Luca Prigioniero. Two-way automata and bounded languages. In *Conference on Implementation and Application of Automata CIAA 2025, Proceedings*, volume 15981 of *Lecture Notes in Computer Science*, pages 73–85. Springer, 2025. doi:10.1007/978-3-032-02602-6_6.
- 3 Viliam Geffert, Bruno Guillon, and Giovanni Pighizzini. Two-way automata making choices only at the endmarkers. *Information and Computation*, 239:71–86, 2014. doi:10.1016/j.ic.2014.08.009.
- 4 Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173–1187, 2007. doi:10.1016/j.ic.2007.01.008.
- 5 Bruno Guillon and Luca Prigioniero. Linear-time limited automata. *Theoretical Computer Science*, 798:95–108, 2019. doi:10.1016/j.tcs.2019.03.037.
- 6 Bruno Guillon, Luca Prigioniero, and Javad Taheri. Nondeterminism makes unary 1-limited automata concise. In *Developments in Language Theory DLT 2025, Proceedings*, volume 16036 of *Lecture Notes in Computer Science*, pages 153–165. Springer, 2025. doi:10.1007/978-3-032-01475-7_11.
- 7 Bruno Guillon, Luca Prigioniero, and Javad Taheri. Polynomial complementation of non-deterministic 2-way finite automata by 1-limited automata. *CoRR*, abs/2507.11209, 2025. doi:10.48550/arXiv.2507.11209.
- 8 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
- 9 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988. doi:10.1137/0217058.
- 10 Galina Jirásková. State complexity of some operations on binary regular languages. *Theoretical Computer Science*, 330(2):287–298, 2005. doi:10.1016/j.tcs.2004.04.011.
- 11 Galina Jirásková and Giovanni Pighizzini. Optimal simulation of self-verifying automata by deterministic automata. *Information and Computation*, 209(3):528–535, 2011. doi:10.1016/j.ic.2010.11.017.

- 12 Christos A. Kapoutsis. Removing bidirectionality from nondeterministic finite automata. In *Mathematical Foundations of Computer Science MFCS 2005, Proceedings*, volume 3618 of *Lecture Notes in Computer Science*, pages 544–555. Springer, 2005. doi:10.1007/11549345_47.
- 13 Nelma Moreira, Giovanni Pighizzini, and Rogério Reis. Optimal state reductions of automata with partially specified behaviors. *Theoretical Computer Science*, 658:235–245, 2017. doi:10.1016/j.tcs.2016.05.002.
- 14 Giovanni Pighizzini. Two-way finite automata: Old and recent results. *Fundamenta Informaticae*, 126(2-3):225–246, 2013. doi:10.3233/FI-2013-879.
- 15 Giovanni Pighizzini. Limited automata: Properties, complexity and variants. In *Descriptive Complexity of Formal Systems DCFS 2019, Proceedings*, volume 11612 of *Lecture Notes in Computer Science*, pages 57–73. Springer, 2019. doi:10.1007/978-3-030-23247-4_4.
- 16 Giovanni Pighizzini and Andrea Pisoni. Limited automata and regular languages. *International Journal of Foundations of Computer Science*, 25(7):897–916, 2014. doi:10.1142/S0129054114400140.
- 17 Giovanni Pighizzini, Luca Prigioniero, and Simon Sádovský. Performing regular operations with 1-limited automata. *Theory of Computing Systems*, 68(3):465–486, 2024. doi:10.1007/s00224-024-10163-1.
- 18 William J. Sakoda and Michael Sipser. Nondeterminism and the size of two way finite automata. In *Symposium on Theory of Computing STOC 1978, Proceedings*, pages 275–286. ACM, 1978. doi:10.1145/800133.804357.
- 19 John C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959. doi:10.1147/rd.32.0198.
- 20 Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988. doi:10.1007/BF00299636.
- 21 Moshe Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30(5):261–264, 1989. doi:10.1016/0020-0190(89)90205-6.
- 22 Klaus W. Wagner and Gerd Wechsung. *Computational Complexity. Mathematics and its Applications*. Springer, 1986.