The Pumping Lemma for Regular Languages is Hard

Hermann Gruber[1], Markus Holzer[2], and Christian Rauch[2]

[1] Planerio GmbH, Gewürzmühlstr. 11, 80538 München
h.gruber@planerio.de
[2] Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
{holzer,christian.rauch}@informatik.uni-giessen.de

Abstract. We investigate the computational complexity of the PUMPING-PROBLEM, that is, for a given finite automaton A and a value p, to decide whether the language $L(A)$ satisfies a previously fixed pumping lemma w.r.t. the value p. Here we concentrate on two different pumping lemmata from the literature. It turns out that this problem is intractable, namely, it is already coNP-complete, even for deterministic finite automata (DFAs), and it becomes PSPACE-complete for nondeterministic finite state devices (NFAs), for at least one of the considered pumping lemmata. In addition we show that the minimal pumping constant for the considered particular pumping lemmata cannot be approximated within a factor of $o(n^{1-\delta})$ with $0 \leq \delta \leq 1/2$, for a given n-state NFA, unless the Exponential Time Hypothesis (ETH) fails.

1 Introduction

The syllabus on courses of automata theory and formal languages certainly contains the introduction of pumping lemmata for regular and context-free languages in order to prove non-regularity or non-context freeness, respectively, of certain languages. Depending on the preferences of the course instructor and the used monograph, different variants of pumping lemmata are taught. For instance, consider the following pumping lemma, or iteration lemma, that can be found in [11, page 70, Theorem 11.1], which describes a necessary condition for languages to be regular.

Lemma 1. *Let L be a regular language over Σ. Then, there is a constant p (depending on L) such that the following holds: If $w \in L$ and $|w| \geq p$, then there are words $x \in \Sigma^*$, $y \in \Sigma^+$, and $z \in \Sigma^*$ such that $w = xyz$ and $xy^tz \in L$ for $t \geq 0$—it is then said that y can be* pumped *in w.*

A lesser-known pumping lemma, attributed to Jaffe [10], characterizes the regular languages, by describing a necessary and sufficient condition for languages to be regular. For other pumping lemmata see, e.g., the annotated bibliography on pumping [12]:

Lemma 2. *A language L is regular if and only if there is a constant p (depending on L) such that the following holds: If $w \in \Sigma^*$ and $|w| = p$, then there are words $x \in \Sigma^*$, $y \in \Sigma^+$, and $z \in \Sigma^*$ such that $w = xyz$ and*[3]

$$wv = xyzv \in L \iff xy^t zv \in L$$

for all $t \geq 0$ and each $v \in \Sigma^*$.

For a regular language L the value of p in Lemma 1 can always be chosen to be the number of states of a finite automaton, regardless whether it is deterministic or nondeterministic, accepting L. Consider the unary language $a^n a^*$, where all values p with $0 \leq p \leq n$ do *not* satisfy the properties of Lemma 1, but $p = n+1$ does. A closer look on some example languages reveals that sometimes a much smaller value suffices. For instance, consider the language

$$L = a^* + a^*bb^* + a^*bb^*aa^* + a^*bb^*aa^*bb^*,$$

which is accepted by a (minimal) deterministic finite automaton with five states, the sink state included—see Figure 1. Already for $p = 1$ the statement of

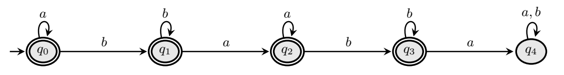


Fig. 1: The minimal deterministic finite automaton A accepting the language L.

Lemma 1 is satisfied since regardless whether the considered word starts with a or b, this letter can be readily pumped. Thus, the minimal pumping constant satisfying the statement of Lemma 1 for the language L is 1, because the case $p = 0$ is equivalent to $L = \emptyset$ [4]. This already shows that the minimal pumping constant w.r.t. Lemma 1 is non-recursively related to the deterministic state complexity of a language. For the pumping constant w.r.t. Lemma 2 the situation is even more involved. Here the value of p can only be chosen to be the number of states of a deterministic finite automaton accepting the language in question. In fact, the gap between the nondeterministic state complexity and the minimal pumping constant p satisfying Lemma 2 for a specific language can be arbitrarily large—cf. Theorem 3. Moreover, the relation between the minimal pumping constant p w.r.t. Lemma 2 is related to the deterministic state complexity n of a language L over the alphabet Σ by the inequality $p \leq n \leq \sum_{i=0}^{p-1} |\Sigma|^i$, which was recently shown in [7]. A careful inspection of the language L mentioned above reveals that the minimal pumping constant p when considering Jaffe's pumping lemma is equal to five. Any word of length at least five uses a loop-transition in its computation—see Figure 1—and hence the letter on this loop-transition

[3] Observe that the words $w = xyz$ and $xy^t z$, for all $t \geq 0$, belong to the same Myhill-Nerode equivalence class of the language L. Thus, one can say that the pumping of the word y in w *respects equivalence classes*.

can be pumped in the word under consideration. Thus $\texttt{mpe}(L) \leq 5$. On the other hand, the word *baba* cannot be pumped such that the Myhill-Nerode equivalence classes are respected, because the word xz belongs to a different equivalence class than xyz, for every decomposition of *baba* into x, y, and z with $|y| \geq 1$—again, see Figure 1. Therefore, $\texttt{mpe}(L) > 4$ and thus, $\texttt{mpe}(L) = 5$. This example shows that the minimal pumping constant is *not* equivalent to the length of the shortest accepting path in (the minimal) automaton accepting the language. The authors experienced that this is the most common misconception of the minimal pumping constant when discussing this for the first time.

This gives rise to the following definition of a *minimal pumping constant*: for a regular language L let $\texttt{mpc}(L)$ ($\texttt{mpe}(L)$, respectively) refer to the minimal number p satisfying the conditions of Lemma 1 (Lemma 2, respectively). Recently, in [4, 7] minimal pumping constants w.r.t. the above two lemmata were investigated from a descriptional complexity perspective. Here we focus more on the computational complexity of pumping, a problem that to our knowledge was not considered so far. This is even more surprising, since pumping lemmata are omnipresent in theoretical courses in automata theory and formal languages. We will consider the following problem related to the pumping lemma stated above:

Language-Pumping-Problem or for short Pumping-Problem:
- Input: a finite automaton A and a natural number p, i.e., an encoding $\langle A, 1^p \rangle$.
- Output: Yes, if and only if the statement from Lemma 1 holds for the language $L(A)$ w.r.t. the value p.

A similar definition applies when considering the condition of Lemma 2 instead. We summarize our findings on the computational complexity of the Pumping-Problem in Table 1.

2 Preliminaries

We assume the reader to be familiar with the basics in computational complexity theory [13]. In particular we recall the inclusion chain: $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$. Here P (NP, respectively) denotes the class of problems solvable by deterministic (nondeterministic, respectively) Turing machines in polytime, and PSPACE refers to the class of languages accepted by deterministic or nondeterministic Turing machines in polynomial space [16]. As usual,

	Pumping-Problem w.r.t. ...	
	Lemma 1	Lemma 2
DFA	coNP-complete	
NFA	coNP-hard contained in Π_2^{P}	PSPACE-compl.
	No det. $2^{o(s^{\delta})}$-time approx. within $o(s^{1-\delta})$, unless ETH fails.	

Table 1: Complexity of the Pumping-Problem for variants of finite state devices.

the prefix co refers to the complement class. For instance, coNP is the class of problems that are complements of NP problems. Moreover, recall the complexity class Π_2^{P} from the polynomial hierarchy, which can be described by polynomial time bounded oracle Turing machines. Here $\Pi_2^P = \mathsf{coNP}^{\mathsf{NP}}$, where $\mathsf{coNP}^{\mathsf{A}}$ is the set of decision problems solvable in polynomial time by a universal Turing machine with an oracle for some complete problem in class A. The class NP^{A} is defined analogously. Completeness and hardness are always meant with respect to deterministic many-one logspace reducibilities ($\leq_m^{\log}$) unless stated otherwise.

Next we fix some definitions on finite automata—cf. [6]. A *nondeterministic finite automaton* (NFA) is a quintuple $A = (Q, \Sigma, \cdot, q_0, F)$, where Q is the finite set of *states*, Σ is the finite set of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states*, and the *transition function* $\cdot$ maps $Q \times \Sigma$ to 2^Q. Here 2^Q refers to the powerset of Q. The *language accepted* by the NFA A is defined as $L(A) = \{\, w \in \Sigma^* \mid (q_0 \cdot w) \cap F \neq \emptyset \,\}$, where the transition function is recursively extended to a mapping $Q \times \Sigma^* \to 2^Q$ in the usual way. An NFA A is said to be *partial deterministic* if $|q \cdot a| \leq 1$ and *deterministic* (DFA) if $|q \cdot a| = 1$ for all $q \in Q$ and $a \in \Sigma$. In these cases we simply write $q \cdot a = p$ instead of $q \cdot a = \{p\}$. Note that every partial DFA can be made complete by introducing a non-accepting sink state that collects all non-specified transitions. For a DFA, obviously every letter $a \in \Sigma$ induces a mapping from the state set Q to Q by $q \mapsto q \cdot a$, for every $q \in Q$. Finally, a finite automaton is *unary* if the input alphabet Σ is a singleton set, that is, $\Sigma = \{a\}$, for some input symbol a.

The *deterministic state complexity of a finite automaton* A with state set Q is referred to as $\mathtt{sc}(A) := |Q|$ and the *deterministic state complexity of a regular language* L is defined as

$$\mathtt{sc}(L) = \min\{\, \mathtt{sc}(A) \mid A \text{ is a DFA accepting } L\text{, i.e., } L = L(A) \,\}.$$

A similar definition applies for the *nondeterministic state complexity of a regular language* by changing DFA to NFA in the definition, which we refer to as $\mathtt{nsc}(L)$. It is well known that

$$\mathtt{nsc}(L) \leq \mathtt{sc}(L) \leq 2^{\mathtt{nsc}(L)},$$

for every regular language L.

A finite automaton is *minimal* if its number of states is minimal with respect to the accepted language. It is well known that each minimal DFA is isomorphic to the DFA induced by the Myhill-Nerode equivalence relation. The *Myhill-Nerode* equivalence relation $\sim_L$ for a language $L \subseteq \Sigma^*$ is defined as follows: for $u, v \in \Sigma^*$ let $u \sim_L v$ if and only if $uw \in L \iff vw \in L$, for all $w \in \Sigma^*$. The equivalence class of u is referred to as $[u]_L$ or simply $[u]$ if the language is clear from the context and it is the set of all words that are equivalent to u w.r.t. the relation $\sim_L$, i.e., $[u]_L = \{\, v \mid u \sim_L v \,\}$. Therefore we refer to the automaton induced by the Myhill-Nerode equivalence relation $\sim_L$ as the minimal DFAfor the language L. On the other hand there may be minimal non-isomorphic NFAs for L.

3 The Complexity of Pumping

Before we start with the investigation of the complexity of pumping problems we list some simple facts about the minimal pumping constants known from the literature: (1) $\mathtt{mpc}(L) = 0$ if and only if $L = \emptyset$, (2) for every non-empty finite language L we have $\mathtt{mpc}(L) = 1 + \max\{\, |w| \mid w \in L \,\}$, (3) $\mathtt{mpc}(L) = 1$ implies for the empty word λ that $\lambda \in L$, and (4) $\mathtt{mpe}(L) = 1$ if and only if $L = \emptyset$ or $L = \Sigma^*$. Also the inequalities $\mathtt{mpc}(L) \leq \mathtt{sc}(L) \leq \mathtt{nsc}(L)$ hold—see, e.g., [4, 7]. For a finite languages we have $\mathtt{mpe}(L) = 2+\max\{|w| \mid w \in L\}$ if each w with $|w| = \max\{|w| \mid w \in L\}$ is contained in L. Otherwise $\mathtt{mpe}(L) = 1 + \max\{|w| \mid w \in L\}$ holds. The relation of $\mathtt{mpe}(L)$ and the state complexities is more subtle, namely for a regular language over the alphabet Σ it was shown in [7] that

$$\mathtt{mpe}(L) \leq \mathtt{sc}(L) \leq \sum_{i=0}^{\mathtt{mpe}(L)-1} |\Sigma|^i.$$

On the other hand, $\mathtt{mpe}(L)$ and $\mathtt{nsc}(L)$ are incomparable in general as we see next—the automaton used to prove the second statement of the following Theorem is shown in Figure 2:

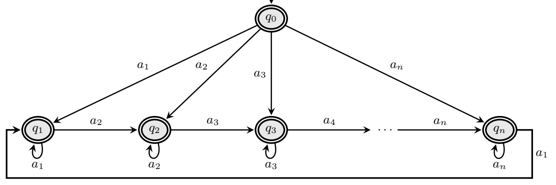


Fig. 2: The nondeterministic finite automaton A_n accepting $L_n = L(A_n)$ satisfying $\mathtt{mpe}(L_n) = 3$ and $\mathtt{nsc}(L_n) \geq n$.

Theorem 3. *The following statements hold: (1) There is a family of unary regular languages $(L_i)_{i\geq 2}$ such that the inequality $\mathtt{nsc}(L_i) < \mathtt{mpe}(L_i)$ holds. (2) There is a family of regular languages $(L_i)_{i\geq 4}$ over a growing size alphabet such that $3 = \mathtt{mpe}(L_i) < i \leq \mathtt{nsc}(L_i)$.*

Observe, that the automaton depicted in Figure 2 is a partial DFA, where only the non-accepting sink state is missing. It is worth mentioning that this finite state device shows that even for DFAs the $\mathtt{mpe}$-measure and the *longest path* of the automaton, that is, a simple[4] directed path of maximum length starting in the initial state of the automaton, are different measures in general. Nevertheless, the following observation is immediate by Jaffe's proof, cf. [10]:

[4] Here a path is called *simple* if it does not have repeated states/vertices.

Lemma 4. *Let A be a DFA and $L := L(A)$. Then $\mathtt{mpe}(L) \leq \ell_A + 1$, where ℓ_A is the length, i.e., number of transitions, of the longest path of the automaton A. If L is a unary language, then $\mathtt{mpe}(L) = \mathtt{sc}(L)$.* □

3.1 The Pumping-Problem for NFAs

We start our journey with results on the complexity of the Pumping-Problem for NFAs. First we consider the problem w.r.t. Lemma 1 and later in the subsection w.r.t. Lemma 2.

Theorem 5. *Given an NFA A and a natural number p, it can be decided in Π_2^{P} whether for the language $L(A)$ the statement of Lemma 1 holds for the value p.*

In order to prove the theorem, we establish an auxiliary lemma.

Lemma 6. *Given an NFA $A = (Q, \Sigma, \cdot, q_0, F)$ and a word w over Σ, the language inclusion problem for w^* in $L(A)$ is coNP-complete. The variant of the problem where A is deterministic is L-complete.*

Now we are ready to prove the upper bound for the Pumping-Problem:

Proof (of Theorem 5). We show that the pumping problem for NFAs belongs to Π_2^{P}. Let $\langle A, p\rangle$ be an input instance of the problem in question, where Q is the state set of A. We construct a coNP Turing machine M with access to a coNP oracle: first the device M deterministically verifies whether $p \geq |Q|$, and if so halts and accepts. Otherwise the computation universally guesses ($\forall$-states) a word w with $p \leq |w| < |Q|$. On that particular branch M checks deterministically if w belongs to $L(A)$. If this is *not* the case the computation halts and accepts. Otherwise, M deterministically cycles through all valid decompositions $w = xyz$ with $|y| \geq 1$. Then it constructs a finite automaton B accepting the language quotient $(x^{-1} \cdot L(A)) \cdot z^{-1}$. Here, if A is deterministic, then so is B. Then M decides whether $y^* \subseteq L(B)$ with the help of the coNP oracle—compare Lemma 6. If $y^* \subseteq L(B)$, then the cycling through the valid decompositions is stopped, and the device M halts and accepts. Notice that the latter is the case iff $xy^*z \subseteq L(A)$. Otherwise, i.e., if $y^* \not\subseteq L(B)$, the Turing machine M continues with the next decomposition in the enumeration cycle. Finally, if the cycle computation finishes, the Turing machine halts and rejects, because no valid decomposition of w was found that allows for pumping. In summary, the Turing machine operates universally, runs in polynomial time, and uses a coNP oracle. Thus, the containment within Π_2^{P} follows. □

Next we show that the problem in question is coNP-hard and gives us a nice non-approximability result under the assumption of the so-called *Exponential-Time Hypothesis (ETH)* [3, 9]: there is no deterministic algorithm that solves 3SAT in time $2^{o(n+m)}$, where n and m are the number of variables and clauses, respectively.

Note that the unary regular language $T_p = a^{p-1}a^*$ satisfies $\mathtt{mpc}(T_p) = p$. The languages T_p will be a basic building block for our reduction. We build upon the classical coNP-completeness proof of the inequality problem for regular expressions without star given in [8, Thm. 2.3]. We modify the reduction a bit, since we want to deal with only one parameter in the analysis to come, and that is the number of clauses.

Theorem 7. *Let φ be a formula in 3DNF with n variables and m clauses. Then a nondeterministic finite automaton A_φ can be computed in time $O(m^2)$ such that the language $Z = L(A_\varphi)$ is homogeneous*[5] *and Z equals $\{0,1\}^{3m}$ if and only if φ is a tautology. Furthermore, A_φ has $O(m^2)$ states.*

The last $3m - n$ positions of the words in the homogeneous language used for the above reduction do not convey any information; they simply serve the purpose of avoiding a parameterization by n. In order to finish our reduction, we embed Z into the language $Y = Z \cdot \# \cdot \Sigma^* + \{0,1\}^{3m} \cdot \# \cdot T_p$, for some carefully chosen p, where $\#$ is a new letter not contained in Σ. This reduction runs in polynomial time.

Lemma 8. *Let φ be a formula in 3DNF with n variables and m clauses and let A_φ be the NFA constructed in Theorem 7. Furthermore, let*

$$Y = Z \cdot \# \cdot \Sigma^* + \{0,1\}^{3m} \cdot \# \cdot T_p,$$

for $p \geq 2$. Then the minimal pumping constant of Y w.r.t. Lemma 1 is equal to $3m+2$, if $Z = \{0,1\}^{3m}$, and is equal to $3m+1+p$ otherwise.

As a direct corollary we obtain:

Corollary 9. *Given an NFA A and a natural number p in unary, it is* coNP-*hard to decide whether for the language $L(A)$ the statement from Lemma 1 holds for the value p.* □

Next we prove the following main result:

Theorem 10. *Let A be an NFA with s states, and let δ be a constant such that $0 < \delta \leq 1/2$. Then no deterministic $2^{o(s^\delta)}$-time algorithm can approximate the minimal pumping constant w.r.t. Lemma 1 within a factor of $o(s^{1-\delta})$, unless ETH fails.*

Proof. We give a reduction from the 3DNF tautology problem as in Lemma 8. That is, given a formula φ in 3DNF with n variables and m clauses, we construct an NFA A that accepts the language $Y = Z \cdot \# \cdot \Sigma^* + \{0,1\}^{3m} \cdot \# \cdot T_p$. Here, the set Y features some carefully chosen parameter p, which will be fixed later on. For now, we only assume $p \geq 2$.

By Lemma 8, the reduction is correct in the sense that if φ is a tautology, then the minimal pumping constant w.r.t. Lemma 1 is strictly smaller than in the case where it admits a falsifying assignment.

[5] A language $L \subseteq \Sigma^*$ is *homogeneous* if all words in L are of same length.

Observe that the running time of the reduction is linear in the number of states of the constructed NFA describing Y. It remains to estimate that number of states. Recall from Theorem 7 that the number of states in the NFA A_φ is of order $O(m^2)$. The set $Z \cdot \# \cdot \Sigma^*$ thus admits an NFA with $O(m^2)$ states; and the language $\{0,1\}^{3m} \cdot \# \cdot T_p$ can be accepted by an NFA with $O(m+p)$ states. Altogether, the number of states is in $O(m^2+p)$.

Now we need to fix the parameter p in our reduction; let us pick $p = m^{\frac{1}{\delta}}$, where it is understood that we round up to the next integer. Given that δ is constant, for large enough m, we can ensure that $p \geq 2$. So this is a valid choice for the parameter p—in the sense that the reduction remains correct.

Towards a contradiction with the ETH, assume that there is an algorithm A_δ approximating the pumping constant within $o(s^{1-\delta})$ running in time $2^{o(s^\delta)}$. Then algorithm A_δ could be used to decide whether $Z = \{0,1\}^{3m}$ as follows: the putative approximation algorithm A_δ returns a cost C that is at most $o(s^{1-\delta})$ times the optimal cost C^*, that is, $C = o(s^{1-\delta}) \cdot C^*$.

We consider two cases: if $Z = \{0,1\}^{3m}$, then the pumping constant is in $O(m)$ by Lemma 8. In this case, the hypothetical approximation algorithm A_δ returns a cost C with

$$\begin{aligned} C = o(m \cdot s^{1-\delta}) &= o\left(m \cdot O\left(m^2 + m^{\frac{1}{\delta}}\right)^{1-\delta}\right) \\ &= o\left(m \cdot O\left(m^{\frac{1}{\delta}}\right)^{1-\delta}\right) = o(m^{\frac{1}{\delta}}) = o\left(m^{\frac{1}{\delta}} + m\right) = o\left(m+p\right); \end{aligned}$$

where in the second step of the above calculation, we used the fact that $s = O(m^2+p) = O(m^2 + m^{\frac{1}{\delta}})$; in the third step, we applied the inequality $\frac{1}{\delta} \geq 2$ to see that the term $m^{\frac{1}{\delta}}$ asymptotically dominates the term m^2; the fourth step is a simple term transformation; and the last two steps apply these facts in reverse order.

On the other hand, in case Z is not full, then Lemma 8 states that the pumping constant is in $\Omega(m+p)$. Using the constants implied by the O-notation, the size returned by algorithm A_δ could thus be used to decide, for large enough m, whether Z is full, and thus by Theorem 7 whether the 3DNF formula φ is a tautology.

It remains to show that the running time of A_δ in terms of the number of clauses m is in $2^{o(m)}$, which contradicts the ETH. Recall that $s = O(m^2+p)$ and $p = m^{\frac{1}{\delta}}$ with $\frac{1}{\delta} \geq 2$; we thus can express the running time of the algorithm A_δ in terms of the number of clauses m, namely,

$$2^{o(s^\delta)} = 2^{o\left((m^2+p)^\delta\right)} = 2^{o\left((m^{\frac{1}{\delta}})^\delta\right)} = 2^{o(m)},$$

which yields the desired contradiction to the ETH. □

A careful inspection of Lemma 8 and Theorem 10 reveals that both results remain valid if one considers the minimal pumping constant w.r.t. Lemma 2, since $\texttt{mpe}(T_p) = p$ as the interested reader may easily verify.

Although, we have to leave open the exact complexity of the Pumping-Problem for NFAs w.r.t. Lemma 1—coNP-hard and contained in Π_2^{P}, we can give a precise answer if we consider Jaffe's pumping lemma instead. First we establish an auxiliary theorem.

Theorem 11. *Given a DFA $A = (Q, \Sigma, \cdot, q_0, F)$ and a deterministic or non-deterministic finite automaton B, deciding whether every word $w \in L(B)$ describes the same equivalence class w.r.t. the Myhill-Nerode relation $\sim_{L(A)}$, is* NL*-complete. If the automaton A is an NFA, the problem becomes* PSPACE*-complete.*

This allows us to prove the following PSPACE-completeness:

Theorem 12. *Given an NFA A and a natural number p in unary, it is* PSPACE*-complete to decide whether for the language $L(A)$ the statement from Lemma 2 holds for the value p.*

3.2 The Pumping-Problem for DFAs

Here we find for both pumping lemmata under consideration that the corresponding Pumping-Problem for DFAs is coNP-complete. First let us prove the upper bound:

Theorem 13. *Given a DFA A and a natural number p in unary. To decide whether for the language $L(A)$ the statement from Lemma 1 holds for the value p can be solved in* coNP*. The same upper bound applies if Lemma 2 is considered.*

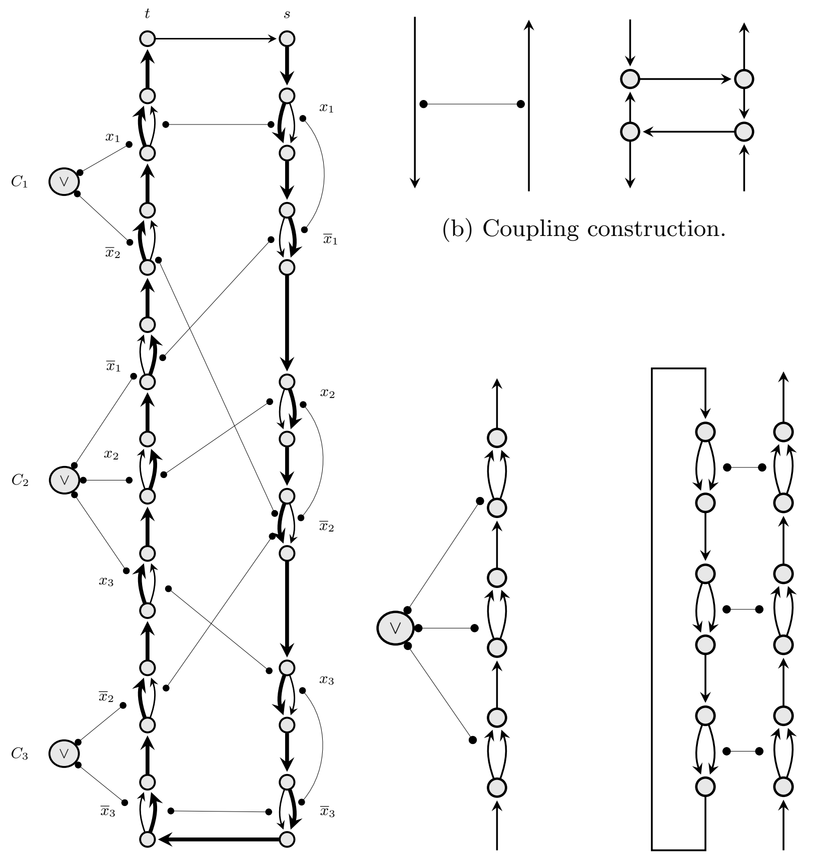
In fact, both problems are coNP-hard. To this end, we utilize the construction of [15] of a directed (planar) graph from a 3SAT instance φ that has a Hamiltonian cycle if and only if φ is satisfiable. Assume that $\varphi = \bigwedge_{i=1}^{m} C_i$ is a 3SAT formula with n variables $x_1, x_2, \ldots, x_n$ and m clauses. Without loss of generality we may assume that every variable occurs at most four times in φ and no variable appears in only one polarity (pure literal). Let us briefly recall the construction of [15], slightly adapted to our needs,[6] which is illustrated in Figure 3a for the formula

$$\varphi = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3).$$

The conversion of the given skeleton graph into a directed graph is achieved through the utilization of couplings as demonstrated in Figure 3b and clause gadgets depicted in Figure 3c—clause gadgets for a single literal or two literals are constructed straight-forwardly. We require that the value of x_i (or $\overline{x}_i$) is *true* if a given Hamiltonian s-t-path of the directed graph indicated in Figure 3a contains the left edge from every pair of edge assigned to x_i (or $\overline{x}_i$, respectively). Otherwise the value is assumed to be *false*.

The directed graph G_φ that has been constructed will possess a Hamiltonian s-t-path if and only if the Boolean formula φ is satisfiable. Additionally, the Hamiltonian s-t-path must satisfy the following conditions:

[6] Instead of a Hamiltonian cycle we ask for a Hamiltonian s-t-path.



(a) Skeleton graph for φ.

(b) Coupling construction.

(c) Clause gadget.

Fig. 3: Schematic drawing of the skeleton graph constructed for the 3SAT instance φ. The obtained direct graph has a Hamiltonian s-t-path if and only if φ is satisfiable. Here, formula φ is satisfiable, because $\varphi|_{x_1=1,x_2=0,x_3=1} = 1$. A Hamiltonian s-t-path (without traversing the coupling connections) is indicated with a boldface line.

1. The s-t-path must pass through all left edges assigned to x_i (or $\overline{x}_i$), or all right edges assigned to x_i ($\overline{x}_i$, respectively).
2. It is not permitted for the s-t-path to pass through both a left (right) edge of x_i and a left (right) edge of $\overline{x}_i$ at the same time.
3. There must be at least one left edge present in every clause C_j.

Now we are ready to state our next theorem:

Theorem 14. *Given a DFA A and a natural number p, it is* coNP*-hard to decide whether for the language $L(A)$ the statement of Lemma 1 (Lemma 2, respectively) holds for the value p.* □

Proof. Let φ be a 3SAT formula with n variables and m clauses. Then by the above construction we obtain a skeleton graph and in turn a directed graph G_φ that has a Hamiltonian s-t-path if and only if φ is satisfiable. It remains to construct a *partial* DFA A_φ out of the skeleton/directed graph by giving an appropriate labeling of the edges such that the Hamiltonian s-t-path can only be pumped trivially. The vertices of the directed graph become the states of the automaton A_φ and the edges become transitions with appropriate labels described below. Moreover, the initial state of A_φ is the state s and the sole accepting state is set to t.

In the skeleton graph, an edge labeled a (or b, respectively) is coupled by the construction illustrated in Figure 4 with an edge labeled b (or a, respectively). Observe, that in the resulting graph both connecting edges (these are the back

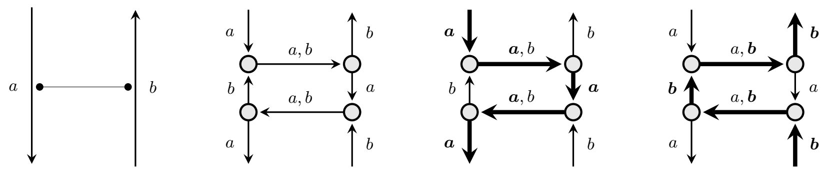


Fig. 4: Coupling of two edges labeled a and b and two possible traversals.

and forth edges) carry the labels a and b simultaneously. Then a left (right, respectively) bended edge of a variable x_i is labeled a (b, respectively). For $\overline{x}_i$ this is exactly the other way around. The determinism of the automaton together with the coupling thus induces the a-b-labeling of all other bended edges. Finally, every (vertical) straight edge, except for the edge connecting t and s, is labeled by the letter #. The t-s-edge is labeled with all letters, i.e., a, b, and #. Thus, the input alphabet is equal to $\Sigma = \{a, b\} \cup \{\#\}$. This completes the description of the partial DFA A_φ—see Figure 5 for a partial drawing of the automaton.

First we show that A_φ is minimal, if completed.

Claim 1. The partial DFA A_φ is bideterministic[7] and if completed, i.e., by introducing a non-accepting sink state that collects all non-specified transitions, it is the minimal (ordinary) DFA.

Next we consider the minimal pumping constants induced by the language accepted by A_φ.

Claim 2. Let L be the language accepted by the partial DFA A_φ. Then we have $\texttt{mpe}(L) = \texttt{sc}(A_\varphi)$ if and only if the Boolean formula φ is satisfiable. The same holds true for the measure $\texttt{mpc}(L)$.

[7] A finite automaton A is *bideterministic* if it is both partially deterministic and partially co-deterministic and has a sole accepting state. Here A is *partially co-deterministic* if the reversed automaton obtained by reversing the transitions of A is partially deterministic.

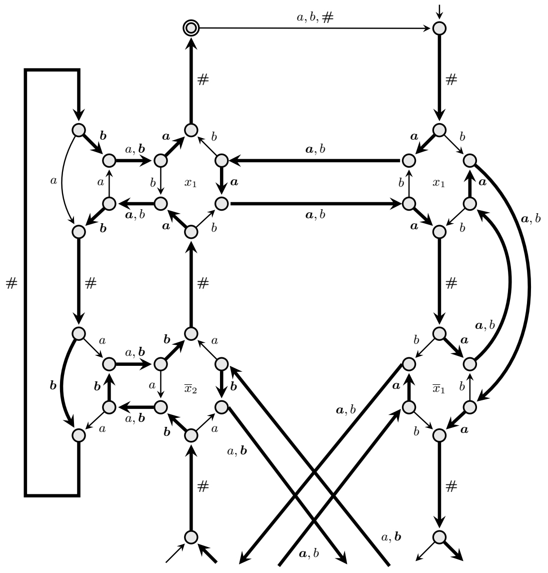


Fig. 5: Part of the partial DFA constructed from the skeleton graph with the help of coupling and clause gadgets. Here $n = 3$ and $m = 3$. A Hamiltonian s-t-path p is indicated with boldface transitions and any word w that fits to p is depicted with boldface letters on the transitions.

The complete automaton for A_φ and $\mathsf{sc}(A_\varphi)$ is a no instance of the Pumping-Problem w.r.t. Lemma 1 (Lemma 2, respectively) if and only if the Boolean formula φ is unsatisfiable. Thus the Pumping-Problem for DFAs is coNP-hard. □

Putting the results together we get:

Corollary 15. *Given a DFA A and a natural number p, it is* coNP*-complete to decide whether for the language $L(A)$ the statement of Lemma 1 (Lemma 2, respectively) holds for the value p.* □

References

1. Angluin, D.: Inference of reversible languages. J. ACM **29**(3), 741–765 (1982)
2. Birget, J.C.: Intersection and union of regular languages and state complexity. Inform. Process. Lett. **43**, 185–190 (1992)
3. Cygan, M., Fomin, F., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms, chap. Lower Bounds Based on the Exponential-Time Hypothesis, pp. 467–521. Springer (2015).
4. Dassow, J., Jecker, I.: Operational complexity and pumping lemmas. Acta Inform. **59**, 337—355 (2022)

5. Glaister, I., Shallit, J.: A lower bound technique for the size of nondeterministic finite automata. Inform. Process. Lett. **59**, 75–77 (1996)
6. Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley (1978)
7. Holzer, M., Rauch, C.: On Jaffe's pumping lemma, revisited. In: DCFS. LNCS, Springer, Potsdam, Germany (2023), accepted for publication
8. Hunt, III, H.B.: On the time and tape complexity of languages I. In: Proceedings of the 5th Annual ACM Symposium on Theory of Computing. pp. 10–19. ACM, Austin, Texas, USA (1973)
9. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. System Sci. **63**(4), 512–530 (2001).
10. Jaffe, J.: A necessary and sufficient pumping lemma for regular languages. SIGACT News **10**(2), 48–49 (Sommer 1978)
11. Kozen, D.C.: Automata and Computability. Undergraduate Texts in Computer Science, Springer (1997)
12. Nijholt, A.: YABBER—yet another bibliography: Pumping lemma's. An annotated bibliography of pumping. Bull. EATCS **17**, 34–53 (1982)
13. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
14. Pin, J.E.: On reversible automata. In: LATIN. pp. 401–416. No. 583 in LNCS, Springer, São Paulo, Brazil (1992)
15. Plesník, J.: The NP-completeness of the hamiltonian cycle problem in planar digraphs with degree bound two. Inform. Process. Lett. **8**(4), 199–201 (1978)
16. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. System Sci. **4**(2), 177–192 (1970)
17. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Proceedings of the 5th Symposium on Theory of Computing. pp. 1–9 (1973)

Appendix

Proof (of Theorem 3). For the first statement we argue as follows: let p_i be the ith prime number (counting 2 as the first). Then consider the unary language

$$L_i = a(a^{p_i})^* + a(a^{p_{i+1}})^*.$$

The language L_i is accepted by a $(1+p_i+p_{i+1})$-state automaton A which consists of an initial state that is connected to two cycles of p_i and p_{i+1} states, where the first state in the cycles is accepting. Hence, $\mathtt{nsc}(L_i) \leq 1 + p_i + p_{i+1}$. On the other hand, L_i is accepted by a minimal DFA with $p_i \cdot p_{i+1}$ states which consists of a single cycle and properly chosen final states. Since L_i is unary, this implies that $\mathtt{mpe}(L_i) = \mathtt{sc}(L_i) = p_i \cdot p_{i+1}$. For $i \geq 2$ it holds $1 + p_i + p_{i+1} < p_i \cdot p_{i+1}$ which implies the stated claim.

The second statement is seen by considering the nondeterministic state devices $A_n = (Q_n, \Sigma_n, \cdot_n, q_0, Q_n)$, where the set of states $Q_n = \{q_0, q_1, \ldots, q_n\}$, the input alphabet $\Sigma_n = \{\, a_i \mid 1 \leq i \leq n \,\}$, the initial state q_0, and the transition function is specified by

- $q_0 \cdot_n a_i = \{q_i\}$, for $1 \leq i \leq n$,
- $q_i \cdot_n a_i = \{q_i\}$, for $1 \leq i \leq n$,
- $q_i \cdot_n a_{i+1} = \{q_{i+1}\}$, for $1 \leq i < n$, and $q_n \cdot_n a_1 = \{q_1\}$.

The NFA A_n is depicted in Figure 2; note that A_n is in fact partially deterministic.

Let $L_n = L(A_n)$. First we show that $\mathtt{mpe}(L_n) = 3$. Observe, that by construction, any word that reaches a state q_i with $1 \leq i \leq n$ after two letters can be pumped using the first letter. Thus, such a word starts with $a_i a_i$, $a_i a_{i+1}$, or $a_n a_1$. If the word under consideration does not start with any of these prefixes, then after reading the second letter the computation of the NFA cannot be continued and the word is *not* accepted. But none of these two-letter words can be pumped (not by the first letter, the second, nor by the whole word of length two). In this case a third letter is required in order to get a word that can be pumped, namely by its third letter. Thus, $\mathtt{mpe}(L_n) = 3$ as required.

In order to prove the result on the nondeterministic state complexity we use the so called extended fooling set technique, which was introduced in [2] and is defined as follows—compare also with [5]: A set $S = \{\, (x_i, y_i) \mid 1 \leq i \leq n \,\}$ is an *extended fooling set* for the regular language $L \subseteq \Sigma^*$, if

1. $x_i y_i \in L$ for $1 \leq i \leq n$, and
2. $i \neq j$ implies $x_i y_j \notin L$ or $x_j y_i \notin L$, for $1 \leq i, j \leq n$.

Then any NFA accepting the language L has at least n states, i.e., $\mathtt{nsc}(L) \geq n$.

Consider the following set of pairs of words

$$S = \{\, (a_i, a_i) \mid 1 \leq i \leq n \,\}.$$

Obviously $a_i a_i$ is a member of L_n, for $1 \leq i \leq n$. It remains to verify the second property of being an extended fooling set. To this end assume that $i < j$.

Consider any two pairs (a_i, a_i) and (a_j, a_j) with $1 \leq i < j \leq n$. Observe that all words $a_j a_i$ are not in L_n for $j \neq n$ or $i \neq 1$. For $j = n$ and $i = 1$ we see that the word $a_1 a_n$ is not in L_n since $n \geq 3$. Thus, S is an extended fooling set for L_n of size n. Therefore any NFA accepting L_n requires at least n states. This proves the stated claim. $\square$

Proof (of Lemma 6). To show that the problem is in coNP, we define the unary NFA

$$A_w = (Q, \{a\}, \cdot_w, q_0', F')$$

with $q_0' = q_0$, and for $p, q \in Q$ let $p \cdot_w a \ni q$, if $p \cdot y \ni p$, and finally $F' = F$. By construction of A_w we find

$$w^t \in L(A) \quad \text{if and only if} \quad a^t \in L(A_w),$$

for $t \geq 0$. Thus, if $L(A_w)$ is universal, i.e., $L(A_w) = a^*$, then every word w^t, for $t \geq 0$ belongs to the language $L(A)$, and therefore the word w can be pumped. Otherwise, there is a t such that a^t is *not* in $L(A_w)$, which implies that w^t is not in $L(A)$. Thus, in turn

$$w^t \in L(A), \text{ for every } t \geq 0, \quad \text{if and only if} \quad L(A_w) = a^*.$$

We note, that for the hardness the special case $\Sigma = \{a\}$ and $w = a$ is equal to the universality problem for unary NFAs, which is coNP-complete by [17].

For the variant of the problem where A is deterministic, we note for the upper bound that the construction of A_w gives a unary DFA in this case, and for the complexity lower bound that the universality problem for unary DFAs is well known to be L-complete. $\square$

Proof (of Theorem 7). Let $\varphi = \bigvee_{i=1}^{m} C_i$ be a formula in 3DNF, and let n denote the number of variables. We can assume without loss of generality that no clause C_i contains both x_j and $\overline{x}_j$ as a literal, and that each variable occurs in at least one of the clauses, which implies $3m \geq n$.

For each clause C_i, let $\zeta_i = \zeta_{i1}\zeta_{i2}\cdots\zeta_{in} \cdot \{0,1\}^{3m-n} \subseteq \{0,1\}^{3m}$, where

$$\zeta_{ij} = \begin{cases} (0+1) & \text{if both } x_j \text{ and } \overline{x}_j \text{ are not literals in } C_i, \\ 0 & \text{if } \overline{x}_j \text{ is a literal in } C_i, \\ 1 & \text{if } x_j \text{ is a literal in } C_i. \end{cases}$$

Let $\zeta = \zeta_1 + \zeta_2 + \cdots + \zeta_m$. It is easy to construct NFAs A_{ζ_i} accepting the language ζ_i, each having $3m+1$ states. Merging the start states of these NFAs into a single state yields a new NFA A_φ accepting the union of these languages. This yields a total of $O(m^2)$ states.

Clearly, $Z = L(A_\varphi) \subseteq \{0,1\}^{3m}$. Let w in $\{0,1\}^n$. Then w is the prefix of a word in Z if and only if w satisfies some clause C_i. Thus $Z = \{0,1\}^{3m}$ if and only if φ is a tautology. This completes the reduction. $\square$

Proof (of Lemma 8). We distinguish two cases:

1. If φ is a tautology, then we have $Y = \{0,1\}^{3m} \cdot \# \cdot \Sigma^*$ and therefore the minimal pumping constant is $3m+2$, because the smallest word is of length $3m+1$ and ends with $\#$ but pumping cannot be performed since pumping a subword within the prefix not containing $\#$ would alter the length of the prefix, which must be of length $3m$. Pumping a subword containing the $\#$ symbol would reproduce it several times, or zero times. In all cases the word obtained by pumping is not in Y. In contrast to that all words of length at least $3m+2$ can be pumped by the subword which contains only the last symbol. Therefore the minimal pumping constant w.r.t. Lemma 1 is $3m+2$.
2. On the other hand, if φ is *not* a tautology, then there is an assignment under which φ evaluates to *false*. Hence, there is a word $w \in \{0,1\}^n$ that corresponds to that assignment and is *not* prefix of a member of $L(A_\varphi)$. But then any word in the set $w\{0,1\}^{3m-n}\#T_p$ requires length at least $3m+1+p$ to become pumpable w.r.t. Lemma 1. Since all words in Y whose length n prefix describes a satisfying assignment are pumpable if they are at least of length $3m+2$ by a similar argument as in the first case. Thus, the minimal pumping constant for Y w.r.t. Lemma 1 is $3m+1+p$ in this case.

This completes this proof. □

Proof (of Theorem 11). First consider the case where A is a DFA. Since NL is closed under complementation it suffices to prove that there are two words w_1 and w_2 in $L(B)$ such that the states $q_0 \cdot w_1$ and $q_0 \cdot w_2$ are distinguishable in A. This means that w_1 and w_2 are not equivalent w.r.t. the Myhill-Nerode relation $\sim_{L(A)}$. This can be done by nondeterministically guessing these two words in logspace in a letter-by-letter fashion and determining $q_0 \cdot w_1$ and $q_0 \cdot w_2$ and finally asking for non-equivalence of these two states, which is known to be an NL-complete problem. Thus the containment within NL follows. For the NL-hardness we argue as follows: observe, that $\langle A, B\rangle$ with $L(B) = \Sigma^*$ is a positive instance of the problem in question if and only if $L(A) = \Sigma^*$. Hence this is a reduction from the NL-complete universality problem for DFAs. Thus, the stated claim follows.

If A is an NFA, we determinize it with the well-known powerset construction in order to obtain an equivalent DFA and run the above NL-algorithm. Since the determinized automaton is of exponential size and can be reconstructed on demand, the PSPACE upper bound follows. The PSPACE-hardness follows with the same reduction as given above, since the universality problem for NFAs is PSPACE-complete. □

Proof (of Theorem 12). The containment of the problem within $\mathsf{PSPACE} = \mathsf{AP}$ is seen as follows: on input $\langle A, 1^p\rangle$ an alternating polynomial time Turing machine first universally guesses a word w of length p. Then it existentially guesses a decomposition $w = xyz$ with $x \in \Sigma^*$, $y \in \Sigma^+$, and $z \in \Sigma^*$ and constructs an

NFA B accepting the set xy^*z. Finally, it verifies whether $\langle A, B\rangle$ is a positive instance of the generalized state equivalence problem. By Theorem 11 this can be done in alternating polynomial time, since A is an NFA. If this is a positive instance the Turing machine halts and accepts; otherwise it halts and rejects.

Finally we show that the PSPACE-hardness follows from the PSPACE-complete universality problem for NFAs. To this end, let A be the instance of the universality problem. Then we verify in deterministic logspace, whether the empty word belongs to $L(A)$. If this is the case the reduction outputs A and value $p = 1$; otherwise it outputs an automaton accepting the language $\{\lambda\}$ and also the value $p = 1$, where λ is the empty word. Then by the fact that $\mathtt{mpe}(L) = 1$ if and only if L is equal to the empty set or the full set, as mentioned at the beginning of this section, the desired hardness follows. □

Proof (of Theorem 13). First let us consider the Pumping-Problem w.r.t. Lemma 1. Recall the proof of Theorem 5, where on an NFA input a coNP Turing machine with a coNP oracle for the inclusion problem of some w^* in a language accepted by an NFA is used. In the case where A is a DFA, we still need a coNP Turing machine but now the access to a logspace oracle by Lemma 6 suffices, thus putting the problem in $\mathsf{coNP}^{\mathsf{L}} = \mathsf{coNP}$.

Next we turn our attention to the Pumping-Problem w.r.t. Jaffe's pumping lemma. A Turing machine solves the problem in question as follows: first it universally guesses a word w of length p. Then it cycles through all decompositions $w = xyz$ with $w \in \Sigma^*$, $y \in \Sigma^+$, and $z \in \Sigma^*$, constructs an NFA B accepting xy^*z, and verifies in deterministic polynomial time whether $\langle A, B\rangle$ is a positive instance of the generalized state equivalence problem—see Theorem 11. If this is the case the Turing machine halts and accepts; otherwise it continues with the next decomposition in the enumeration cycle. In case the cycle ends without finding a positive instance, then the Turing machine halts and rejects. This proves the containment within coNP. □

Proof (of Claim 1). The automaton A_φ is partially deterministic and a close inspection of its construction reveals that it is co-deterministic, too. Moreover, since it has unique accepting state it is bideterministic. Then the automaton obtained from A_φ by introducing a non-accepting sink state that collects all non-specified transitions, is minimal according to [1, 14]. □

Proof (of Claim 2). Recall that the Boolean formula φ is satisfiable if and only if there is a Hamiltonian cycle or equivalently a Hamiltonian s-t-path in the directed graph associated with the skeleton graph G_φ. Note that the Hamiltonian property always applies to the partial automaton.

Assume first that there is *no* Hamiltonian s-t-path. Then the longest path of the completed automaton A_φ (starting in its initial state) is strictly less than $\ell := \mathtt{sc}(A_\varphi) - 1$. This is seen as follows: for the longest path in the complete automaton A_φ starting in the initial state there are two possibilities, namely, the path ends in the accepting state or in the non-accepting sink state. We distinguish these two cases:

1. In the former case, we have an s-t-path but it is *not* Hamiltonian by assumption. Thus, the path length is strictly less than $\mathtt{sc}(A_\varphi)-1$. The non-accepting sink state cannot be used to prolong this path.
2. For the latter case, where the longest path ends in the non-accepting sink state, we can exclude the accepting state to be part of this path: namely, each transition leaving t enters the start state by construction of the automaton A_φ, and the start state can occur only once in the path. Now let q denote the state that precedes the accepting state, i.e., q is the state satisfying $q \cdot_{A_\varphi} \# = t$. In the sub-case where q is part of the path, then it must precede the non-accepting sink state in the path, since each transition leaving q enters either the sink state, or the accepting state, and we have just ruled out that the latter is part of the path. Hence we can alter the path by changing the last step from q now towards t instead of moving towards the non-accepting sink state without changing the path length. Thus, we have constructed another longest path, this time ending in t. From the discussion of Case (1) above the length of a longest path is strictly less than $\mathtt{sc}(A_\varphi)-1$ in this sub-case. In the other sub-case, where q is not part of the path, then it may contain all states of A_φ, except for t and q, together with the non-accepting sink state. These are $\mathtt{sc}(A_\varphi)-2$ states. Hence the length of the path is strictly less than $\mathtt{sc}(A_\varphi)-1$. Thus, by Lemma 4 we deduce that $\mathtt{mpe}(L) < \ell + 1 = \mathtt{sc}(A_\varphi)$. On the other hand, if there is a Hamiltonian s-t-path we conclude that $\mathtt{mpe}(L) \leq \mathtt{sc}(A_\varphi)$, since the longest path of the partial device A_φ (and also of its completed version) is equal to $\mathtt{sc}(A_\varphi)-1$ in this case.

In order to prove the statement that the Boolean formula φ is satisfiable if and only if there is a Hamiltonian s-t-path if and only if $\mathtt{mpe}(L) = \mathtt{sc}(A_\varphi)$ it suffices to give a witness showing that $\mathtt{mpe}(L) \geq \mathtt{sc}(A_\varphi)$. Now assume that there is a Hamiltonian s-t-path p. Every path p induces several words, if an edge with both labels a and b is traversed. We choose a word w that fits the path p as follows: whenever the path p traverses an edge with a sole letter, this symbol is used, while for edges with two letters a and b the letter that corresponds to the previous letter induced by the path p is preferred. If we apply these rules, then the word w induced by the Hamiltonian s-t path indicated in Figure 3 (partially also shown in Figure 5) reads as follows:

$$\begin{aligned} w = \#a^5\#a^5 \cdot \#b^5\#b^9 \cdot \#a^5\#a^5 \cdot \#a^5\#bba\#a^5\#aab \\ \cdot \#aab\#a^5\#b^5\#bba\#b^5\#a^5 \cdot \#bbbbb\#aab\#b\#bba\#. \end{aligned}$$

By construction this belongs to the language $L(A_\varphi)$. We prove that the word w of length $\mathtt{sc}(A_\varphi)-1$ is such a desired witness—the following argument does not require that the word w is constructed as described above.

Consider any decomposition of w into $x \in \Sigma^*$, $y \in \Sigma^+$, and $z \in \Sigma^*$ with $w = xyz$. We show that y fails to be pumped in w. To this end let $q_x = s \cdot_{A_\varphi} x$ and $q_{xy} = s \cdot_{A_\varphi} xy$. Observe, that q_x is different from q_{xy}, since otherwise w does not fit to a Hamiltonian s-t-path. Because w is a member of $L(A_\varphi)$ by the computation

$$s \cdot_{A_\varphi} w = q_x \cdot_{A_\varphi} yz = q_{xy} \cdot_{A_\varphi} z = t,$$

it suffices to show that xz is not in $L(A_\varphi)$. Assume to the contrary that xz belongs to the language under consideration. Then $s \cdot_{A_\varphi} xz = q_x \cdot_{A_\varphi} z = t$. Because, A_φ is bideterministic the word z can only be accepted from the state q_{xy}. Hence, we conclude that $q_x = q_{xy}$. This is a contradiction to the property of q_x and q_{xy} of being different states. Hence, any non-empty sub-word of w cannot be pumped. Therefore, $\texttt{mpe}(L) > |w| = \texttt{sc}(A_\varphi) - 1$, which gives $\texttt{mpe}(L) \geq \texttt{sc}(A_\varphi)$ and proves the stated claim.

Since $\texttt{mpc}(L) \leq \texttt{mpe}(L)$ holds for every regular language, and the above argumentation applies to a word w accepted by A_φ, the statement is also valid for the other pumping constant w.r.t. Lemma 1. $\square$