Decidability Results on the Existence of Lookahead Delegators for NFA

Christof Löding and Stefan Repke

Lehrstuhl für Informatik 7, RWTH Aachen, Aachen, Germany

Abstract

In this paper, we study lookahead delegators for nondeterministic finite automata (NFA), which are functions that deterministically choose transitions by additionally using a bounded lookahead on the input word. Of course, the delegator has to lead to an accepting state for each word that is accepted by the NFA. In the special case where no lookahead is allowed, a delegator coincides with a deterministic transition function that preserves the language.

Typical decision problems are to decide whether a delegator with a given fixed lookahead exists, or whether a delegator with some bounded lookahead exists for a given NFA. In a paper of Ravikumar and Santean from 2007, the complexity and decidability of these questions have been tackled, mainly for the case of unambiguous NFA. In this paper, we revisit the subject and provide results for the case of general NFA. First, we correct a complexity result from the above paper by showing that the existence of delegators with fixed lookahead can be decided in time polynomial in the number of states. We use two player games on graphs as a tool to obtain the result. As second contribution, we show that the problem becomes PSPACE-complete if the bound on the lookahead is a part of the input. The third result provides a bound on the maximal required amount of lookahead. We use this to show that the (previously open) problem of deciding the existence of a bounded lookahead delegator is also PSPACE-complete.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Automata, Lookahead Delegators, Safety Games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.327

1 Introduction

We revisit questions on the decidability of so called lookahead delegators for nondeterministic finite automata (NFA) that have been studied in [9]. A lookahead delegator for an NFA is a function that deterministically chooses one of the possible transitions for the next input symbol, based on a bounded lookahead in the input. The run that is constructed in this way by the delegator for an input word should be accepting if the input word belongs to the language accepted by the NFA.

Motivated from the composition of e-services that are modeled by finite automata, lookahead delegators have been studied in [4] in a slightly different context. In the setting of [4], instead of a single NFA, a tuple of deterministic finite automata (DFA) or NFAs is given, and the delegator has to decide for each input symbol by which subset of the automata this symbol has to be processed, such that in the end, all automata finish with a successful run. Given a bound k on the allowed lookahead, [4] presents an algorithm to compute a k-lookahead delegator (if one exists) for a tuple of DFAs. The algorithm is exponential in k and the number of DFAs in the list.

By taking the product of the tuple of the given automata, in which a transition nondeterministically chooses a subset of the automata that move on the next input symbol, the setting can be reduced to the question on lookahead delegators for a single given NFA. This

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 327–338

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

version of the problem has been analyzed in [9]. There are two main questions concerning the existence of lookahead delegators. For a given bound k on the lookahead, the decidability of the existence of a k-lookahead delegator is clear, because there are only finitely many possible candidates for a given NFA. So, for a given (or a fixed) k, the main question is the complexity of deciding the existence of a k-lookahead delegator (and computing one if it exists). If the bound is not given, the question is whether there exists some k such that a given NFA has a k-lookahead delegator.

In this paper, we will reconsider the three versions of the decision problem that were defined in [9, Section 4]:

a) k-DELEGATOR for a fixed number $k \in \mathbb{N}$: decide for a given NFA $\mathcal{A}$ whether $\mathcal{A}$ has a k-lookahead delegator.
b) DELEGATOR: decide for a given NFA $\mathcal{A}$ and $k \in \mathbb{N}$ whether $\mathcal{A}$ has a k-lookahead delegator.
c) BOUNDED-DELEGATOR: decide for a given NFA $\mathcal{A}$ whether $\mathcal{A}$ has a bounded lookahead delegator.

These problems have been solved for the restricted case of unambiguous NFAs in [9, Theorems 2 to 4]: A polynomial time algorithm is given for k-DELEGATOR, DELEGATOR is shown to be in co-NP, and BOUNDED-DELEGATOR is shown to be in PSPACE.

We study these problems here for the case of general NFAs. Our main contributions are as follows. We provide an algorithm that decides k-DELEGATOR in time polynomial in the number of states of a given NFA (and computes a k-lookahead delegator, if one exists) where k and the input alphabet are fixed. This generalizes [9, Theorem 2] from unambiguous to arbitrary NFAs.[1] As a main tool, we use two person games on graphs. This formulation of the problem yields a simple algorithm and correctness proof.

We furthermore show that the more general problem DELEGATOR, where the allowed lookahead k is a part of the input, is PSPACE-complete. The algorithm that runs in polynomial space is different from the one based on games for a fixed k, which is exponential in k and thus doubly exponential in the binary representation of k.

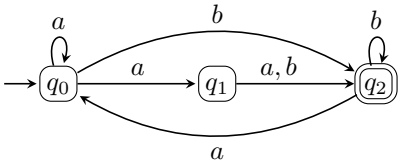
Finally, we prove that if an NFA $\mathcal{A}$ has a k-lookahead delegator for some k, then it also has a K-lookahead delegator for a bound K that is singly exponential in the size of $\mathcal{A}$. This shows the decidability of the problem BOUNDED-DELEGATOR which was left open in [9] and in [4]. In combination with the result for DELEGATOR, we also obtain PSPACE-completeness of this problem.

The remainder of the paper is structured as follows. Section 2 introduces basic terminology and results for later use. Then, in Sections 3, 4 and 5, we present solutions to the three problems k-DELEGATOR, DELEGATOR, and BOUNDED-DELEGATOR, respectively.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, \ldots\}$ be the set of non-negative integers. By $|S|$, we denote the cardinality of a set S. An *alphabet* Σ is a finite set of symbols. For $i \in \mathbb{N}$, Σ^i denotes the set of all sequences of Σ-symbols of length i. An element $w \in \Sigma^i$ is called a *word* and $|w| = i$ is its length. The *empty word*, denoted by ε, is the unique word of length $|\varepsilon| = 0$. Let $\Sigma^{\leq k} = \bigcup_{i=0}^{k} \Sigma^i$ for $k \in \mathbb{N}$ and $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$. A subset $L \subseteq \Sigma^*$ is called a *language*. For languages $L_1, L_2 \subseteq \Sigma^*$, let $L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$ be their concatenation.

[1] We note here that this corrects an error of [9, Theorem 5], which states that the problem is PSPACE-hard for arbitrary NFAs. The proof uses a reduction from an inclusion problem of the form $L(\mathcal{A}) \subseteq L_0$ for a fixed language L_0 and an NFA $\mathcal{A}$. However, this problem is not PSPACE-hard.



(a) NFA $\mathcal{A}$ with accepting states $F = \{q_2\}$

w	$(aw)^{-1}L_0$	$w^{-1}L_0$	$w^{-1}L_1$
ε	$L_0 \cup L_1$	L_0	L_1
a	$L_0 \cup L_1 \cup L_2$	$L_0 \cup L_1$	L_2
aa	$L_0 \cup L_1 \cup L_2$	$\underline{L_0 \cup L_1 \cup L_2}$	L_0
ab	L_2	$\underline{L_2}$	$\underline{L_2}$
b	L_2	$\underline{L_2}$	$\underline{L_2}$

(b) Left quotients of $\mathcal{A}$ (correct choices of the successor according to Lemma 3 are underlined)

Figure 1 NFA and its left quotients

Automata A *nondeterministic finite automaton* (*NFA* for short) $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ consists of a finite set of states Q, an alphabet Σ, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and an initial state $q_0 \in Q$ as well as a set $F \subseteq Q$ of accepting states. For a given input word $w \in \Sigma^*$, a *run* starting in q_0 is a sequence $q_0 q_1 \cdots q_{|w|}$ of states such that $(q_{i-1}, a_i, q_i) \in \Delta$ for each $i \in \{1, \ldots, |w|\}$ where $w = a_1 \cdots a_{|w|}$. We say that $\mathcal{A}$ *accepts* w if it has an *accepting* run from q_0, i.e., with last state accepting: $q_{|w|} \in F$. The language $L(\mathcal{A}) \subseteq \Sigma^*$ consists of those words that are accepted by $\mathcal{A}$.

For the rest of the paper, let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ be an NFA. We assume that each state $q \in Q$ is reachable from q_0 and that each state has at least one outgoing transition for each letter (missing transitions can be added to a sink state).

Lookahead Delegators A k-delegator has to choose a transition from Δ deterministically by looking ahead on the next k input symbols. It is further required to still accept $L(\mathcal{A})$.

▸ **Definition 1.** For $k \in \mathbb{N}$, a *k-lookahead delegator for $\mathcal{A}$* (or *k-delegator* for short) is a function $f : Q \times \Sigma\Sigma^{\leq k} \to Q$ such that

a) $\big(q, a, f(q, aw)\big) \in \Delta$ for each $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^{\leq k}$, and
b) $f^*(q_0, w) \in F$ for each $w \in L(\mathcal{A})$, where we define $f^* : Q \times \Sigma^* \to Q$ as follows: for $w = a_1 \cdots a_{|w|}$, let $q_i = f(q_{i-1}, a_i \cdots a_{\min(|w|, i+k)})$ for $0 < i \leq |w|$, and let $f^*(q_0, w) = q_{|w|}$.

Note that our notion of k-lookahead follows [4, 3] by counting the additional lookahead whereas in [9], this is understood as $(k+1)$-lookahead as they count the current input symbol as a part of the lookahead. Hence, in the setting of Definition 1, a 0-lookahead delegator for an NFA can be identified with a deterministic subset of the transitions such that the same language is accepted. We say that $\mathcal{A}$ *has a bounded lookahead delegator* if it has a k-lookahead delegator for some $k \in \mathbb{N}$.

▸ **Example 2.** For the alphabet $\Sigma = \{a, b\}$, consider the NFA $\mathcal{A}$ depicted in Figure 1a which accepts the language $L(\mathcal{A}) = \{aa\} \cup \Sigma^*\{b, aaa\}$. The only nondeterministic choice of a transition is at state q_0 for symbol a. A function $f : Q \times \Sigma\Sigma^{\leq 2} \to Q$ is a 2-delegator for $\mathcal{A}$ if $f(q_0, aa) = q_1$ and $f(q_0, aaa) = q_0$. For all other cases, it suffices that f is consistent with Δ, i.e., $\big(q_0, a, f(q_0, aw)\big) \in \Delta$ for $w \in \Sigma^{\leq 2}$. Example 4 will justify why this yields a 2-delegator.

Left Quotients For $u \in \Sigma^*$ and $L \subseteq \Sigma^*$, let $u^{-1}L = \{v \in \Sigma^* \mid uv \in L\}$ be the *left quotient* of u with L. This is the language containing each word that completes u to a word in L. Note that the composition of left quotients can be written as concatenation: $v^{-1}(u^{-1}L) = (uv)^{-1}L$. Further, we abbreviate $L = L(\mathcal{A})$ and $L_q = L(\mathcal{A}_q)$ for the language of $\mathcal{A}$ where q is taken as the initial state.

We now present a technical lemma that occurs as a key ingredient in the proofs and algorithms of the following three sections. A similar statement can be found in [9, Lemma 7] using a notion called blindness. It gives a language-theoretical characterization of the main property a k-delegator has to fulfill when it selects a transition. That is, whenever an a-successor p has to be chosen for some lookahead w and state q, then $(aw)^{-1}L_q = w^{-1}L_p$, i.e., $awv \in L_q$ iff $wv \in L_p$ for all $v \in \Sigma^*$. Note that the inclusion $(aw)^{-1}L_q \supseteq w^{-1}L_p$ holds generally, since $(q, a, p) \in \Delta$.

▸ **Lemma 3.** *$\mathcal{A}$ has a k-delegator iff there exists a set $Q' \subseteq Q$ such that $q_0 \in Q'$ and for each $q \in Q'$, $a \in \Sigma$, $w \in \Sigma^k$, there exists $p \in Q'$ such that $(q, a, p) \in \Delta$ and $(aw)^{-1}L_q = w^{-1}L_p$ hold.*

Proof. For the direction from left to right, let f be a k-delegator and Q' be the set of states reachable by f from q_0 with full lookahead, i.e., the smallest set such that $q_0 \in Q'$ and $f(q, aw) \in Q'$ for each $q \in Q'$, $a \in \Sigma$, and $w \in \Sigma^k$. For a contradiction, assume there are $q \in Q'$, $a \in \Sigma$, and $w \in \Sigma^k$ such that $(aw)^{-1}L_q \neq w^{-1}L_p$ for all $p \in Q'$ with $(q, a, p) \in \Delta$. Since q is reachable from q_0, fix a word $u \in \Sigma^*$ such that f leads to q and assume w.l.o.g. that q is the first state on that run with the above property. Let $p = f(q, aw)$. Then, there is a word $v \in (aw)^{-1}L_q \setminus w^{-1}L_p$, i.e., $awv \in L_q$ but $wv \notin L_p$. Hence, we have that $f^*(q_0, uawv) = f^*(q, awv) = f^*(p, wv) \notin F$ whereas $uawv \in L$ in contradiction to the definition of a k-delegator.

For the other direction, we construct a k-delegator f from a set Q' with the above properties. For each $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^{\leq k}$, we set $f(q, aw) = p$ as follows.

a) If $|w| = k$ and $q \in Q'$, then $p \in Q'$ becomes some state such that $(q, a, p) \in \Delta$ and $(aw)^{-1}L_q = w^{-1}L_p$ as directly guaranteed by the property.
b) If $|w| < k$ and $aw \in L_q$, then there is an a-successor $p \in Q$ of q such that $w \in L_p$.
c) If $|w| < k$ and $aw \notin L_q$, then fix some arbitrary a-successor $p \in Q$ of q.

Note that the case $|w| = k$ and $q \notin Q'$ cannot occur due to the above property. It easily follows by the definition of f^* that $f^*(q_0, v) \in F$ if $v \in L$. Hence, f is a k-delegator for $\mathcal{A}$. ◂

▸ **Example 4.** Let us reconsider the NFA $\mathcal{A}$ of Example 2 from the perspective of Lemma 3. Since q_0 is the initial state, a set Q' satisfying the property of Lemma 3 must contain q_0. The only nondeterministic choice happens at q_0 with letter a. The two a-successors of q_0 are q_0 itself and q_1. The left quotients that are relevant for Lemma 3 are listed in Figure 1b for some example words w, where L_i stands for L_{q_i}. For $w = \varepsilon$, one can see that neither the language $w^{-1}L_0$ nor $w^{-1}L_1$ is equivalent to $(aw)^{-1}L_0$ which indicates, according to Lemma 3, that there is no 0-delegator for $\mathcal{A}$. The row for $w = a$ shows analogously that there is also no 1-delegator. When the lookahead is increased to 2, then the condition of Lemma 3 is finally fulfilled. For $w = aa$, the left quotients are the same if and only if a delegator chooses q_0 as a-successor of q_0. The remaining two lines show that the choice does not matter for $w \in \{ab, b\}$.

3 Fixed Lookahead

In this section, we present for an arbitrary fixed number $k \in \mathbb{N}$ an algorithm for the problem k-Delegator, which is to decide the existence of a k-delegator for a given NFA $\mathcal{A}$ (and to compute such a delegator if it exists). The special case 0-Delegator corresponds to deciding whether the NFA $\mathcal{A}$ can be turned into an equivalent DFA just by removing transitions of the NFA. The polynomial time decidability of this very problem has already been mentioned in the survey article [2, Theorem 15] without a proof.

The rough idea behind our approach is to construct a game that simulates the delegation. The two players play a sequence of actions in alternation. One player has to choose the letters of an input word while the other has to choose appropriate transitions. The goal of the player in charge of the transitions is to play an accepting run if a word contained in the language $L(\mathcal{A})$ is formed by the player in charge of the input.

Before we proceed with the detailed definition of this game, we first introduce some terminology concerning 2-player games which follows [5].

Games Formally, a *safety game* $\mathcal{G} = (V, V_0, E, S)$ is played between Player 0 and Player 1, and consists of

a) a finite directed graph (V, E),
b) a partition of the vertices into sets $V_0 \subseteq V$ for Player 0 and $V_1 = V \setminus V_0$ for Player 1, and
c) a set $S \subseteq V$ of safe vertices.

A *play* in $\mathcal{G}$ is a finite or infinite sequence $v_0 v_1 v_2 \cdots$ of vertices. It starts in some initial vertex $v_0 \in V$ and then, for all v_i, there has to be an edge $(v_i, v_{i+1}) \in E$ to its successor v_{i+1} which is chosen by Player 0 if $v_i \in V_0$ or by Player 1 if $v_i \in V_1$. Then, the safety condition states that the play is *won* by Player 0 if all vertices are safe: $v_i \in S$ for all i. Otherwise Player 1 wins the play. We note here that usually a player that cannot move (because there are no outgoing edges) loses. For our purpose, the above condition is more natural, that is, if a maximal play is finite because it ends in a vertex without outgoing edges, then Player 0 wins if all vertices of the play are safe, independent of whether the last vertex is in V_0 or in V_1. One can obtain the standard setting by adding self-loops to terminal vertices of Player 0.

For $\sigma \in \{0, 1\}$, let $V'_\sigma = \left\{ v \in V_\sigma \,\middle|\, (v, v') \in E \text{ for some } v' \in V \right\}$ be the set of non-terminal vertices of Player σ. A *strategy for Player* σ is a function $s : V^* V'_\sigma \to V$ that chooses a successor vertex $v_{i+1} = s(v_0 \cdots v_i)$ with $(v_i, v_{i+1}) \in E$ for each finite play $v_0 \cdots v_i \in V^* V'_\sigma$ ending in a non-terminal vertex v_i of Player σ. We say that s is *winning from a vertex* v_0 if Player σ wins every maximal play $v_0 v_1 v_2 \cdots$ resulting from this strategy, i.e., with $v_{i+1} = s(v_0 \cdots v_i)$ for every $v_i \in V'_\sigma$. Finally, we say that Player σ can *win* $\mathcal{G}$ *from a vertex* v_0 if he has a winning strategy from there.

Safety games are *determined*, i.e., either Player 0 or Player 1 has a winning strategy (see [5]). Further, the respective Player σ can always win with a *positional* strategy s, which means that the choice of the next move only depends on the current vertex, i.e., $s(v_0 \cdots v_i) = s(u_0 \cdots u_j)$ holds for any two finite plays $v_0 \cdots v_i, u_0 \cdots u_j \in V^* V'_\sigma$ with $v_i = u_j$. We then consider a strategy as a function $s : V'_\sigma \to V$.

Delegation Game Now, we can define our game $\mathcal{G}_{\mathcal{A},k}$ according to the main idea explained at the beginning of this section, where Player 1 has to give the input word and Player 0 has to choose transitions. The positions of the game store the corresponding information: a lookahead of k letters (or less if the play goes towards the end of the input word) provided by Player 1, and the current state of $\mathcal{A}$ reached by Player 0. The goal of this construction is to show later in Lemma 7 that $\mathcal{A}$ has a k-delegator if and only if Player 0 has a winning strategy. Since this strategy will occur to be positional, we can directly use it as a function that deterministically chooses transitions, i.e., as a delegator.

The main challenge is now to create a safety game the number of states of which is polynomial in the number of automaton states. The winning condition should express that the state of Player 0 is accepting if the input word played by Player 1 is in the language, which depends not on the current position alone but on the whole play instead. Naïvely, one can implement this as a safety condition by additionally keeping track of the set of reachable

states by the input played by Player 1. However, this leads to a blowup that is exponential in the size of the automaton.

To solve this problem, we modify our game in such a way that Player 1 also has to choose a transition for each input, but after Player 0 has chosen one. We show that, since Player 0 has to make the choice of the transition first, the additional information on the transition chosen by Player 1 does not help Player 0 (because basically, Player 0 has to choose a transition according to Lemma 3, which only depends on the current state of Player 0). This means that in this modified game, a winning strategy for Player 0 still corresponds to a k-delegator.

To summarize, the game $\mathcal{G}_{\mathcal{A},k}$ goes as follows. First, Player 1 gives the initial content of the lookahead. Then, both players play in alternation. Player 0 chooses a transition for the next input letter. Afterwards, Player 1 also chooses a transition for it and simultaneously removes this input symbol (as both players have just processed it) and appends a new letter to the content of the lookahead, or he does not refill it if the input word should end. Consequently, a game position encodes the content of the lookahead as well as one state for each player. The safety condition for Player 0 now simply states that such vertices have to be avoided, where the state of Player 1 is accepting and the state of Player 0 is non-accepting although the lookahead is empty.

▸ **Definition 5.** Given an NFA $\mathcal{A}$ and $k \in \mathbb{N}$, we define the two player safety game $\mathcal{G}_{\mathcal{A},k} = (V, V_0, E, S)$ as follows:

a) $V = \{\top\} \cup \Big(\{0,1\} \times \Sigma^{\leq k+1} \times Q \times Q\Big)$, (initial vertex and simulation vertices)

b) $V_0 = \Big(\{0\} \times \Sigma^{\leq k+1} \times Q \times Q\Big)$,

c) $E \subseteq V \times V$ containing the following edges:

i) $\Big(\top, (0, w, q_0, q_0)\Big)$ for $w \in \Sigma^{\leq k+1}$, (initiate buffer)

ii) $\Big((0, aw, q, p), (1, aw, q', p)\Big)$ for $(q, a, q') \in \Delta$ and $w \in \Sigma^{\leq k}$, (Player 0 applying transition)

iii) $\Big((1, aw, q', p), (0, wb, q', p')\Big)$ for $(p, a, p') \in \Delta$, $w \in \Sigma^k$, and $a, b \in \Sigma$, (Player 1 applying transition, removing leftmost symbol, and refilling lookahead)

iv) $\Big((1, aw, q', p), (0, w, q', p')\Big)$ for $(p, a, p') \in \Delta$, $w \in \Sigma^{\leq k}$, and $a \in \Sigma$, (Player 1 applying transition and removing leftmost symbol without refilling)

d) $S = V \setminus \Big\{(0, \varepsilon, q, p) \;\Big|\; q \notin F \wedge p \in F\Big\}$. (Player 0 has to avoid $\overline{F} \times F$)

The number of vertices of $\mathcal{G}_{\mathcal{A},k}$ is in $\mathcal{O}\big((k+1) \cdot |\Sigma|^{k+1} \cdot |Q|^2\big)$ which is polynomial in $|Q|$ and $|\Sigma|$ for a fixed k. It is easy to see that the game can be constructed in time polynomial in the number of vertices.

▸ **Example 6.** Let us reconsider the NFA $\mathcal{A}$ of Example 2. A part of the 1-delegator game $\mathcal{G}_{\mathcal{A},1}$, that is reachable from the initial vertex $\top$, is depicted in Figure 2, in such a way that for Player 1, only one edge is enabled from each vertex whereas for Player 0, all successors are considered. This deterministic choice corresponds to a positional strategy for Player 1. The strategy always forces the play to an unsafe vertex no matter how Player 0 reacts. Hence, it is a positional winning strategy for Player 1 in $\mathcal{G}_{\mathcal{A},1}$ from $\top$. We show next that this is because there exists no 1-delegator for $\mathcal{A}$ (cf. Example 4).

▸ **Lemma 7.** *Player 0 has a positional winning strategy for $\mathcal{G}_{\mathcal{A},k}$ from $\top$ iff $\mathcal{A}$ has a k-lookahead delegator.*

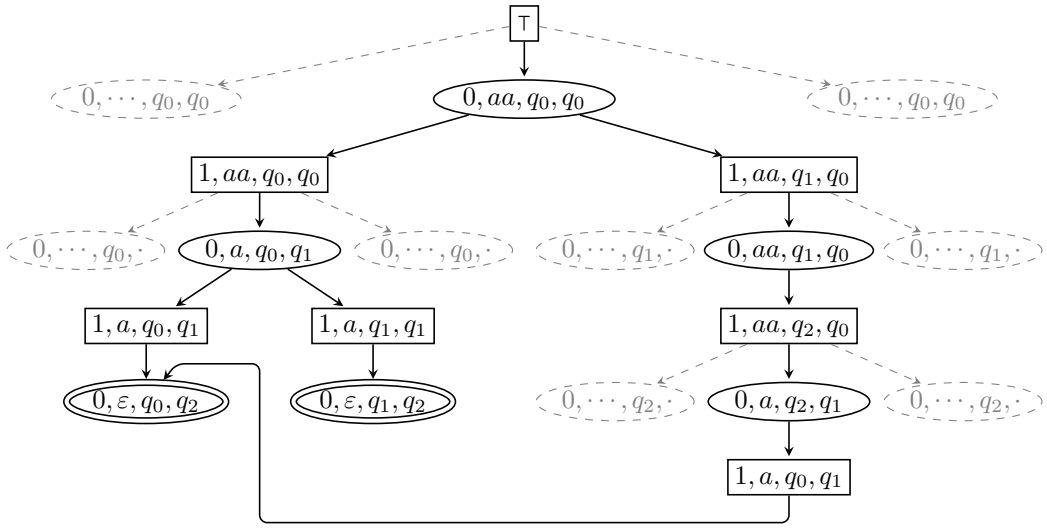


Figure 2 The (partial) 1-delegator game $\mathcal{G}_{\mathcal{A},1}$ showing a positional winning strategy for Player 1 (circled vertices belong to Player 0, boxed ones to Player 1, and doubly circled vertices are unsafe)

Proof. The key observation is that for (a strategy of) Player 0, it is not important to known which states Player 1 chooses since Lemma 3 states that left quotients are important rather than exact states. As long as the property is fulfilled locally, Player 1 can dually be assumed w.l.o.g. to just copy the choices of his opponent.

For the left to right direction, suppose Player 0 has a positional winning strategy s. We show the existence of a k-delegator for $\mathcal{A}$ by proving that the condition from Lemma 3 is satisfied. For this purpose, let Q' be the set consisting of all states $q \in Q$ such that a vertex of the form $(0, aw, q, q)$, with $a \in \Sigma$ and $w \in \Sigma^k$, can be reached with s for some sequence of moves of Player 1. Since trivially $q_0 \in Q'$, it remains to show that for each such vertex, there is a successor $(1, aw, p, q)$ with $(aw)^{-1}L_q = w^{-1}L_p$. Assume, to the contrary, that $(aw)^{-1}L_q \neq w^{-1}L_p$ for each $p \in Q$ with $(q, a, p) \in \Delta$. Then, no matter which p is chosen by Player 0, there is a word $v \in (aw)^{-1}L_q \setminus w^{-1}L_p$, i.e., $awv \in L_q$ but $wv \notin L_p$. Player 1 can win from $(1, aw, p, q)$ by continuing to play the word v since there is a sequence of states that accepts awv from q but none that accepts wv from p. This contradicts the property that s is a winning strategy for Player 0.

For the other direction, suppose $\mathcal{A}$ has a k-delegator f. We can naturally use it to define a positional winning strategy $s : V_0' \to V$ for Player 0 where $s(0, aw, q, p) = \big(1, aw, f(q, aw), p\big)$. One can easily see by the construction of $\mathcal{G}_{\mathcal{A},k}$ and s that $q = f^*(q_0, x)$ holds whenever a terminal vertex $(0, \varepsilon, q, p)$ is reached after Player 1 has played a complete word $x \in \Sigma^*$. Player 0 wins because $p \in F$ implies $x \in L$ and hence, $q \in F$. ◂

By combining Lemma 7 with the linear-time determinacy of safety games (see [5, Theorem 4.1]), we get the main result of this section. Linear-time determinacy means that for the winning player, there exists a positional winning strategy which can be computed in linear time (using a fixed-point algorithm that successively identifies the vertices from which Player 0 is losing, starting with the set of unsafe vertices).

Consequently, the existence of a k-lookahead delegator for $\mathcal{A}$ can be decided in time $\mathcal{O}\big((k+1) \cdot |\Sigma|^{k+1} \cdot |Q|^2\big)$ for a given NFA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ and a positive number k. This yields polynomial running time for a fixed k that we consider in this section.

▸ **Corollary 8.** *For each $k \in \mathbb{N}$, the problem k-*Delegator *can be solved in polynomial time.*

This generalizes [9, Theorem 2] where polynomial time decidability of k-Delegator for each fixed k is shown for unambiguous NFA.

As explained in Section 1, we consider the input to be a single NFA whereas in the original motivation, the input consists of several DFAs. It is shown in [7] that the problem 0-Delegator is EXPTIME-complete in the original setting. This is caused by the fact that the construction of a product of the DFAs yields an NFA that is exponentially larger.

4 Given Lookahead

We now consider the complexity of the problem Delegator, where an NFA and a bound k are given. Note that for deciding whether $\mathcal{A}$ has a k-lookahead delegator, the game-based approach from the previous section yields an algorithm that runs in time that is doubly exponential in the binary representation of k. However, using a different algorithm, we can show that the problem can be solved in polynomial space. The idea of the algorithm is to check whether the property of Lemma 3 holds. The main problem in checking this condition in polynomial space is that we cannot enumerate all words $w \in \Sigma^k$ because their length is exponential in the binary representation of k.

We therefore first introduce transition profiles, which can be used to circumvent this problem. Intuitively, a transition profile of a word w for a given NFA $\mathcal{A}$ describes the possible state transformations induced by w on $\mathcal{A}$, that is, the transition profile for w contains all pairs of states (p, q) such that there is a w-labeled path from p to q.

▸ **Definition 9.** For an NFA $\mathcal{A}$ and a word $w \in \Sigma^*$, we define the *transition profile* $\Delta_w \subseteq Q^2$:

$$
\begin{aligned}
&(q,q) \in \Delta_\varepsilon && \text{for each } q \in Q,\\
&(q,p) \in \Delta_a \;\Leftrightarrow\; (q,a,p) \in \Delta && \text{for each } q,p \in Q,\ a \in \Sigma,\\
&(q,p) \in \Delta_{wa} \;\Leftrightarrow\; \exists r \in Q : (q,r) \in \Delta_w \text{ and } (r,p) \in \Delta_a && \text{for each } q,p \in Q,\ a \in \Sigma,\ w \in \Sigma^*.
\end{aligned}
$$

The main idea for checking the condition of Lemma 3 in polynomial space is to use transition profiles that are induced by words of length k, instead of working directly with the words. This is justified by the simple observation that words with the same profile also have the same left quotient.

▸ **Lemma 10.** *Let $x, y \in \Sigma^*$ be such that $\Delta_x = \Delta_y$. Then, $x^{-1}L_q = y^{-1}L_q$ for all $q \in Q$.*

Proof. A trivial consequence of Definition 9 is that $\Delta_{xw} = \Delta_{yw}$ for all $w \in \Sigma^*$. Then,

$$
\begin{aligned}
w \in x^{-1}L_q \;\Leftrightarrow\; xw \in L_q \;&\Leftrightarrow\; \exists p \in F : (q,p) \in \Delta_{xw}\\
&\Leftrightarrow\; \exists p \in F : (q,p) \in \Delta_{yw} \;\Leftrightarrow\; yw \in L_q \;\Leftrightarrow\; w \in y^{-1}L_q.
\end{aligned}
$$

◂

▸ **Theorem 11.** *The problem* Delegator *is in PSPACE.*

Proof. Let an NFA $\mathcal{A}$ (with the usual components) and k be given. We show that for each $Q' \subseteq Q$, there is a nondeterministic PSPACE algorithm that checks whether the property of Lemma 3 is satisfied. Savitch's theorem (see [8, Theorem 7.5]) implies that there is also a deterministic PSPACE algorithm.

So let $Q' \subseteq Q$ with $q_0 \in Q$. The algorithm tests for each q and each a, whether for each word $w \in \Sigma^k$ there is an a-successor p of q such that $(aw)^{-1}L_q = w^{-1}L_p$. As mentioned above, we cannot enumerate all words $w \in \Sigma^k$ because their length is exponential in the binary

representation of k. Instead, we work with the transition profiles induced by the words w. Each such transition profile is of size polynomial in $\mathcal{A}$ and contains sufficient information to test $(aw)^{-1}L_q = w^{-1}L_p$. Lemma 10 allows us to restrict the test to transition profiles, as words with the same transition profile induce the same left quotient.

We now describe the algorithm. Given Q', $q \in Q'$, and $a \in \Sigma$, the algorithm proceeds as follows. For each transition profile $\tau \in 2^{Q \times Q}$:

a) Check if $\tau = \Delta_w$ for some word w of length k. If it is the case, do the following. Otherwise, move on to the next transition profile.
b) Let $p_1, \ldots, p_n$ be the a-successors of q in Q'. For $i \in \{1, \ldots, n\}$, let $R_i = \{p \in Q \mid (p_i, p) \in \tau\}$ be the set of states that are reached from p_i in the profile τ. Let $R = \bigcup_{1 \le i \le n} R_i$. Note that $L_{R_i} = w^{-1}L_{p_i}$ and $L_R = (aw)^{-1}L_q$, where for $S \subseteq Q$, we let $L_S = \bigcup_{s \in S} L_s$.
c) Check if there is an index $i \in \{1, \ldots, n\}$ such that $L_R = L_{R_i}$.

If the last test fails (meaning that there is no such index i), then Q' does not satisfy the property of Lemma 3. If the test passes for all q, all a, and all the relevant transition profiles (those passing the first test), then Q' has the desired property and thus, $\mathcal{A}$ has a k-lookahead delegator.

It remains to verify that the steps of the algorithm can be carried out in polynomial space. The first test uses the idea of checking reachability in a directed graph in logarithmic space. In our setting, we use a counter for counting up to k (note that the number of bits needed for the counter corresponds to the size of the binary representation of k), and successively guess k steps to reach the transition profile τ. That is, we start with the transition profile Δ_ε of the empty word. In each step, we guess a letter $b \in \Sigma$ and extend the current transition profile Δ_v to Δ_{vb}. After k steps, we check whether the resulting profile Δ_w is equal to τ. At each moment, we only need to store the counter and the intermediate transition profile, which requires polynomial space.

The second step just computes (in LOGSPACE) some sets from the transition profile τ.

The third step requires us to test n equivalences $L_R = L_{R_i}$, where the languages are given by NFAs with the sets R and R_i as initial states, respectively. Since equivalence of NFAs can be tested in polynomial space (see [1]), this step is also in PSPACE. ◂

▸ **Theorem 12.** *The problem* Delegator *is PSPACE-complete.*

Proof. The upper bound follows from Theorem 11.

For the lower bound, let M be some polynomially space bounded Turing machine that solves a PSPACE-hard problem. We show that the word problem for M can be reduced to the problem of the existence of a bounded lookahead delegator. The word problem for M is to decide for a given word whether M accepts w, which is clearly PSPACE-hard because M solves a PSPACE-hard problem.

Let h be the polynomial for the space bound of M. Given a word w, we construct an NFA $\mathcal{A}$ that has a $\big(2h(n) + 2\big)$-lookahead delegator iff M rejects w, where $n = |w|$.

As usual, we encode configurations of M by words of the form $\kappa = usv$, where uv is the content of the tape of M, and s the current control state. The head of M in configuration usv is on the first position of v. We can assume that $|uv| = h(n)$. A computation of M is then encoded by a word of the form $\#\kappa_0\#\kappa_1\#\cdots\kappa_\ell\#$, where κ_0 is the initial configuration of M on w, each κ_{i+1} is the successor configuration of κ_i, and κ_ℓ is an accepting configuration.

The core of the reduction is an NFA $\mathcal{A}_w$ that accepts a word if it does *not* encode an accepting computation of M on w (see [1, Lemma 10.2] for such a construction for regular expressions instead of NFAs). For this purpose, $\mathcal{A}_w$ uses a product of automata testing the following properties:

a) The word is not of the required form $\#\kappa_0\#\kappa_1\#\cdots\kappa_\ell\#$ where each κ_i is of the form $u_i s_i v_i$ with $|u_i v_i| = h(n)$.
b) The first configuration is not the initial configuration of M on w.
c) The last configuration is not an accepting configuration.
d) There is an i such that κ_{i+1} is not the M-successor configuration of κ_i.

The first three properties can be easily checked by DFAs of size linear in $h(n)$. The last property can be checked by an NFA that guesses at some symbol # that this corresponds to the index i, and then guesses a position j in κ_i and tests whether κ_{i+1} has been updated in a wrong way at position j (to detect this, the three symbols at positions $j-1$, j, and $j+1$ are sufficient). The size of such an NFA is also linear in $h(n)$ (it needs to count up to $h(n)$ for finding the corresponding cell j in κ_{i+1}). All the automata can be constructed in logarithmic space from M and w. The automaton $\mathcal{A}_w$ is the product of these four automata that accepts if one of its components accepts. Note that $\mathcal{A}_w$ has a $\big(2h(n)+2\big)$-lookahead delegator because it is sufficient to know the next two configurations to decide which transition to take in the NFA for the last property.

Further, note that $\mathcal{A}_w$ accepts all words if there is no accepting computation of M on w. And if there is such an accepting computation, then $\mathcal{A}_w$ does not accept the word encoding this computation.

We now embed $\mathcal{A}_w$ into an NFA $\mathcal{A}$ to obtain the desired reduction. Let Σ be the alphabet of $\mathcal{A}_w$, and let X, Y, Z be new letters. Define the languages

$$L_1 := X^* \cdot L(\mathcal{A}_w) \cdot \{Y, Z\} \text{ and } L_2 := X^* \cdot \Sigma^* \cdot \{Y\}.$$

Note that $L_2 \subseteq L_1$ iff $L(\mathcal{A}_w) = \Sigma^*$ iff M rejects w.

We construct $\mathcal{A}$ to accept the language $X \cdot (L_1 \cup L_2)$. For this purpose, $\mathcal{A}$ nondeterministically chooses from its initial state on the first X to either go to an automaton $\mathcal{A}_1$ for L_1 or to an automaton $\mathcal{A}_2$ for L_2. The automaton $\mathcal{A}_1$ is a simple extension of $\mathcal{A}_w$ by an X-loop at the beginning and transitions for processing the last Y or Z. The automaton $\mathcal{A}_2$ just consists of an X-loop, followed by a Σ-loop, followed by a transition for Y into an accepting state.

Now assume that M rejects w. Then, $L_2 \subseteq L_1$, as noted above, and a lookahead delegator for $\mathcal{A}$ can always choose the transition going to $\mathcal{A}_1$ from the initial state. We already noted that $\mathcal{A}_w$ has a $\big(2h(n)+2\big)$-lookahead delegator. Overall, we obtain a $\big(2h(n)+2\big)$-lookahead delegator for $\mathcal{A}$ in this case.

Now assume that M accepts w and that $\mathcal{A}$ has a k-lookahead delegator f for some k. Consider the decision of f on the lookahead word X^k. If f moves to $\mathcal{A}_1$, then pick the word v encoding the accepting computation of M on w, followed by the letter Y. $\mathcal{A}_1$ does not accept this word and therefore, f cannot be a k-lookahead delegator because $X^k v Y \in L$.

If f moves to $\mathcal{A}_2$, then consider any word v accepted by $\mathcal{A}_w$ followed by Z. Then, $X^k v Z \in L_1$ but $\mathcal{A}_2$ only accepts words ending with Y. Hence, also in this case, f cannot be a k-lookahead delegator.

This shows that $\mathcal{A}$ has a k-lookahead delegator for some k iff M rejects w. Furthermore, k can be chosen as $2h(n)+2$. ◂

We note that in [9, Theorem 3], it is shown that the problem DELEGATOR for unambiguous NFA is contained in co-NP.

5 Bounded Lookahead

We now turn to the problem of deciding, given an NFA $\mathcal{A}$, whether there exists a k such that $\mathcal{A}$ has a k-delegator, that is, we consider the problem BOUNDED-DELEGATOR. We show

that if $\mathcal{A}$ has some k-delegator, then $\mathcal{A}$ also has some K-delegator for a number K that is singly exponential in the size of $\mathcal{A}$. In the combination with the results from Section 4, we then obtain that Bounded-Delegator is PSPACE-complete, too.

The proof showing that there is this bound K uses a technique inspired by [6], where two player games with lookahead for one of the players are considered. In our setting, the main idea is the following. If the lookahead K is big enough, then each word that can occur as lookahead contains an infix that can be pumped such that the considered lookahead word can be extended to a word of length k (with k and K as explained above where we assume w.l.o.g. $k > K$). On this longer lookahead word of length k, one can query the existing delegator and use the same decision for a delegator with the smaller lookahead K.

The required pumping argument that we just mentioned is formalized by an extension of Lemma 10 where we use transition profiles (cf. Definition 9) again.

▸ **Lemma 13.** *Let $x, y, z \in \Sigma^*$ be such that $\Delta_x = \Delta_{xy}$. Then, $(xyz)^{-1}L_q = (xy^iz)^{-1}L_q$ for all $q \in Q$ and $i \in \mathbb{N}$.*

Proof. An easy induction shows that $\Delta_{xy} = \Delta_{xy^i}$ for all $i \in \mathbb{N}$. Consequently, $\Delta_{xyz} = \Delta_{xy^iz}$ holds and the claim follows directly by Lemma 10. ◂

Using a simple counting argument, we can show that each word of a certain length has a decomposition xyz as in Lemma 13.

▸ **Lemma 14.** *For the bound $K = 2^{|Q|^2}$, each word $w \in \Sigma^K$ can be decomposed as $w = xyz$ with $y \neq \varepsilon$ and $\Delta_x = \Delta_{xy}$.*

Proof. A word of length $2^{|Q|^2}$ has $2^{|Q|^2} + 1$ prefixes. Two different prefixes must have the same transition profile since there are at most $2^{|Q|^2}$ transition profiles. This implies the existence of the claimed decomposition. ◂

We now combine Lemma 13 and Lemma 14 to prove that the bound $K = 2^{|Q|^2}$ is the maximal "useful" lookahead.

▸ **Theorem 15.** *$\mathcal{A}$ has a K-lookahead delegator if it has a bounded lookahead delegator.*

Proof. Let f be a k-delegator for $\mathcal{A}$ where $k > K$ w.l.o.g. We show that the property on the right hand side of Lemma 3, which holds for k by assumption, also holds for K for the same set $Q' \subseteq Q$. To this end, we have to show that for every $q \in Q'$, $a \in \Sigma$, and $w \in \Sigma^K$, there is some $p \in Q'$ with $(q, a, p) \in \Delta$ and $(aw)^{-1}L_q = w^{-1}L_p$. We know that there is a decomposition $w = xyz$ with $y \neq \varepsilon$ and $\Delta_x = \Delta_{xy}$. Choose $i \in \mathbb{N}$ and a proper prefix y' of y such that $|xy^iy'| = k$. Let y'' be such that $y = y'y''$. Finally, we pick some $p \in Q'$ with $(q, a, p) \in \Delta$ such that $(axy^iy')^{-1}L_q = (xy^iy')^{-1}L_p$ the existence of which is guaranteed by Lemma 3 for k-lookahead. With Lemma 13, we can now show that the desired property holds for K-lookahead:

$$
\begin{aligned}
(axyz)^{-1}L_q &= (axy^{i+1}z)^{-1}L_q && \text{(Lemma 13)}\\
&= (axy^iy'y''z)^{-1}L_q && (y = y'y'')\\
&= (y''z)^{-1}\big((axy^iy')^{-1}L_q\big) && \text{(composition of left quotients)}\\
&= (y''z)^{-1}\big((xy^iy')^{-1}L_p\big) && \text{(Lemma 3 for } k\text{-lookahead)}\\
&= (xy^iy'y''z)^{-1}L_p && \text{(composition of left quotients)}\\
&= (xy^{i+1}z)^{-1}L_p && (y = y'y'')\\
&= (xyz)^{-1}L_p && \text{(Lemma 13)}
\end{aligned}
$$

◂

Since the bound K is singly exponential in the number of states of $\mathcal{A}$ and therefore has a binary representation that is polynomial in the size of $\mathcal{A}$, Theorem 11 implies that Bounded-Delegator also is in PSPACE. Furthermore, our reduction showing that Delegator is PSPACE-hard also shows that Bounded-Delegator is PSPACE-hard (see proof of Theorem 12).

▸ **Corollary 16.** *The problem* Bounded-Delegator *is PSPACE-complete.*

In [9, Theorem 4], it is shown that Bounded-Delegator is in PSPACE for unambiguous NFA. This result is generalized by Corollary 16. However, the NFA constructed in the proof of Theorem 12 for the PSPACE lower bound is ambiguous, in general, and therefore, our completeness result does not extend to unambiguous automata.

6 Conclusion

We have shown that the existence of a k-lookahead delegator for an NFA $\mathcal{A}$ can be decided in $\mathcal{O}\big((k+1)\cdot|\Sigma|^{k+1}\cdot|Q|^2\big)$, where Σ is the input alphabet, and Q the state set of the given NFA. In particular, for a fixed k, the problem can be solved in polynomial time. We have furthermore shown that the problem becomes PSPACE-complete if the bound is either a part of the input or no bound is given. This gives a complete picture for the complexities of the decision problems for lookahead delegators for NFA.

As further research, we would like to analyze whether our techniques can be extended to decide the existence of lookahead delegators for some classes of infinite state automata (like counter or pushdown automata), as it has been considered in [3].

Acknowledgements With thanks to the anonymous reviewers for their detailed feedback.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, New York, 1974.
2 Thomas Colcombet. Forms of Determinism for Automata (Invited Talk). In *STACS 2012*, volume 14 of *LIPIcs*, pages 1–23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
3 Zhe Dang, Oscar H. Ibarra, and Jianwen Su. Composability of Infinite-State Activity Automata. In *ISAAC*, pages 377–388, 2004.
4 Cagdas Evren Gerede, Richard Hull, Oscar H. Ibarra, and Jianwen Su. Automated composition of e-services: lookaheads. In *ICSOC*, pages 252–262, 2004.
5 Erich Grädel. Back and Forth Between Logics and Games. In *Lectures in Game Theory for Computer Scientists*, pages 99–145. Springer, 2011.
6 Michael Holtmann, Lukasz Kaiser, and Wolfgang Thomas. Degrees of Lookahead in Regular Infinite Games. *Logical Methods in Computer Science*, 8(3), 2012.
7 Anca Muscholl and Igor Walukiewicz. A Lower Bound on Web Services Composition. *Logical Methods in Computer Science*, 4(2), 2008.
8 Christos H. Papadimitriou. *Complexity Theory*. Addison Wesley, 1994.
9 Bala Ravikumar and Nicolae Santean. On the Existence of Lookahead Delegators for NFA. *Int. J. Found. Comput. Sci.*, 18(5):949–973, 2007.