Completely Reachable Automata: A Polynomial Solution and Quadratic Bounds for the Subset Reachability Problem

Robert Ferens
Institute of Computer Science, University of Wrocław, Wrocław, Poland

Marek Szykuła
Institute of Computer Science, University of Wrocław, Wrocław, Poland

Abstract

A deterministic finite automaton (DFA) with a set of states \( Q \) is completely reachable if every non-empty subset of \( Q \) is the image of the action of some word applied to \( Q \). The concept was first introduced by Bondar and Volkov (2016), who also raised the question of the complexity of deciding if an automaton is completely reachable. We develop a polynomial-time algorithm for this problem, which is based on a complement-intersecting technique for finding an extending word for a subset of states. Additionally, we prove a weak Don’s conjecture for this class of automata: a subset of size \( k \) is reachable with a word of length at most \( 2(|Q| − k)|Q| \). This implies a quadratic upper bound in \(|Q|\) on the length of the shortest synchronizing words (reset threshold) and generalizes earlier upper bounds derived for subclasses of completely reachable automata.

1 Introduction

The concept of completely reachable automata originates from synchronizing automata. A deterministic finite automaton is synchronizing if starting from the set of all states, after reading a suitable reset word, we can narrow the set of possible states to a singleton. On the other hand, an automaton is completely reachable if starting from the set of all states, we can reach its every non-empty subset of states. Thus, every completely reachable automaton is synchronizing, and the class of completely reachable automata forms a remarkable subclass of (synchronizing) automata.

Synchronizing automata are most famous due to the longstanding open problem: the Černý conjecture, which states that a synchronizing \( n \)-state automaton admits a reset word of length at most \((n − 1)^2\). The currently best upper bound is cubic \([13, 15, 16]\).

The concept of complete reachability was introduced in 2016 by Bodnar and Volkov \([4]\), who asked about the complexity of the computational problem of deciding whether a given automaton is completely reachable. Later studies revealed a connection to the so-called Rystsov graph, whose generalization is used to characterize completely reachable automata \([3, 5]\). However, this does not yet lead to an effective algorithm, as it is unknown how to compute these graphs. Recently, the case of binary automata was solved with a quasilinear-time algorithm \([6]\), which strongly relies on the specificity of both letters if they ensure the complete reachability of the automaton. Other studies include connections with the properties of its syntactic semigroup \([3, 12]\).

The class of completely reachable automata is not as small as it might look so. In particular, it contains the Černý automata \([7]\) and some other from the so-called slowly
synchronizing series [1], automata with the full transition monoid [10], and synchronizing automata with simple idempotents [14]. For these subclasses, quadratic upper bounds instead of a cubic one were established for the Černý ’s synchronization problem. However, for the whole class of completely reachable automata, only cubic bounds were known (though better than in the general case) [3].

In connection to the bounds, a remarkable conjecture and a generalization of the Černý one is Don’s conjecture [8, Conjecture 18], which states that for an n-state automaton, if a subset of states of size 1 ≤ k < n is reachable, then it can be reached with a word of length at most n(n − k). This conjecture was disproved in general [11] but weaker versions were proposed: restricting to completely reachable automata [11, Problem 4] and another, in relation to avoiding words [9, Conjecture 15].

1.1 Contribution

We design a polynomial-time algorithm for the problem of deciding whether an automaton is completely reachable, thus solving the 6-year-old open question [4]. For this solution, we develop a complement-intersecting technique, which allows finding a (short) extending word for a given subset of states (i.e., the preimage is larger).

Based on the discovered properties, we also prove that every non-empty subset of k < n states in a completely reachable n-state automaton is reachable with a word of length smaller than 2n(n − k). This is a weaker version (by the factor 2) of Don’s conjecture stated for completely reachable automata [11, Problem 4].

It follows that a completely reachable n-state automaton has a reset word of length at most 2n^2 − n ln n − 4n + 2 (for n ≥ 3). This generalizes and slightly improves the previous bounds obtained for proper subclasses of completely reachable automata: automata with the full transition monoid (with the previous upper bound 2n^2 − 6n + 5) [10] and synchronizing automata with simple idempotents (with the previous upper bound 2n^2 − 4n + 2) [14].

2 Solving Complete Reachability in Polynomial Time

2.1 Reachability

A *deterministic finite complete semi-automaton* (called simply automaton) is a 3-tuple $(Q, \Sigma, \delta)$, where $Q$ is a finite set of states, $\Sigma$ is an input alphabet, and $\delta: Q \times \Sigma \to Q$ is the *transition function*, which is extended to a function $Q \times \Sigma^* \to Q$ in the usual way. Throughout the paper, by $n$ we always denote the number of states in $Q$.

Given a subset $S \subseteq Q$, the *image* of $S$ under the action of a word $w \in \Sigma^*$ is $\delta(S, w) = \{\delta(q, w) \mid q \in S\}$. The *preimage* of $S$ under the action of $w$ is $\delta^{-1}(S, w) = \{q \in Q \mid \delta(q, w) \in S\}$. For a state $q$, we also write $\delta^{-1}(q, w) = \delta^{-1}(\{q\}, w)$. Note that $q \in S = \delta(Q, w)$ if and only if $\delta^{-1}(q, w) \neq \emptyset$.

For a subset $S \subseteq Q$, by $\overline{S}$ we denote its complement $Q \setminus S$.

For two subsets $S, T \subseteq Q$, if there exists a word $w \in \Sigma^*$ such that $\delta(T, w) = S$ then we say that $S$ is reachable from $T$ with $w$. Then we also say that $T$ is a $w$-predecessor of $S$. It is simply a predecessor of $S$ if it is a $w$-predecessor for some word $w$. There can be many $w$-predecessors of one set, but there is at most one maximal with respect to inclusion (and size).

**Remark 1.** For $S \subseteq Q$ and $w \in \Sigma^*$, the preimage $\delta^{-1}(S, w)$ is a $w$-predecessor of $S$ if and only if $\delta^{-1}(q, w) \neq \emptyset$ for every state $q \in S$. Equivalently, we have $\delta(\delta^{-1}(S, w)) = w$. 

\[23:2\] **Completely Reachable Automata: A Polynomial Solution and Quadratic Bounds for the Subset Reachability Problem**
Let \( S \subseteq Q \) and \( w \in \Sigma^* \), if \( \delta^{-1}(S, w) \) is a \( w \)-predecessor of \( S \), then all \( w \)-predecessors of \( S \) are contained in it, thus \( \delta^{-1}(S, w) \) is maximal in terms of inclusion and size. If \( \delta^{-1}(S, w) \) is not a \( w \)-predecessor of \( S \), then \( S \) does not have any \( w \)-predecessors.

A word \( w \) is called extending a subset \( S \) if \(|\delta^{-1}(S, w)| > |S|\) (cf. a study on the related problems \([2]\)). It is called properly extending if additionally \( \delta^{-1}(S, w) \) is a \( w \)-predecessor of \( S \), i.e., \( \delta^{-1}(q, w) \neq \emptyset \) for all \( q \in S \).

We say simply that a subset \( S \subseteq Q \) is reachable in the automaton if \( S \) is reachable from \( Q \) (with any word). An automaton is completely reachable if all non-empty subsets \( S \subseteq Q \) are reachable. Equivalently, \( Q \) is a predecessor of all its non-empty subsets. This gives an alternative direct characterization of completely reachable automata:

**Remark 3.** An automaton \((Q, \Sigma, \delta)\) is completely reachable if and only if for every non-empty proper subset of \( Q \) there is a properly extending word.

The decision problem Completely Reachable is the following: Given an automaton \((Q, \Sigma, \delta)\), is it completely reachable?

### 2.2 Witnesses

For an automaton \((Q, \Sigma, \delta)\), we consider unreachable sets that have the maximal size of all unreachable subsets of \( Q \). They play the role of witnesses (counterexamples) for the non-complete reachability of an automaton.

**Definition 4 (Witness).** A non-empty set \( S \subseteq Q \) is a witness if it is unreachable and has the maximal size of all unreachable subsets of \( Q \).

Since \( Q \) is trivially reachable, the maximal size of unreachable subsets of \( Q \) is in \( \{0, \ldots, n-1\} \). It equals 0 (there is no witness) if and only if the automaton is completely reachable.

Although any non-empty unreachable set is evidence that the automaton is not completely reachable, it turns out that verifying if a set is a witness is computationally easier – we show later that we can verify this and also find a witness in polynomial time. Verifying whether a given set is reachable (or unreachable) in general is PSPACE-complete\([4]\).

We start with a simpler solution in co-NP, where we just guess a candidate for a witness and verify it. Then we describe suitable modifications to work in deterministic polynomial time, where we also compute and check all potential candidates for a witness.

A witness obviously cannot have a larger predecessor, as this predecessor or some other larger set would be a witness instead. Still, a set can have exponentially many predecessors of the same size. The following observation allows us to infer the existence of a larger predecessor indirectly.

**Lemma 5.** Let \( S, T \subseteq Q \) be distinct. If a word \( u \in \Sigma^* \) is properly extending \( S \cup T \), then \( u \) is also properly extending either \( S \) or \( T \).

**Proof.** Let \( u \) be a properly extending word for \( U = S \cup T \subseteq Q \), thus \( \delta(U') = U \), \( \delta^{-1}(U', u) = U' \), and \(|U'| > |U|\). By Remark 1 we have \(|\delta^{-1}(q, u)| \geq 1\) for all \( q \in U \). As this holds for all the states of \( S \) and \( T \), they also have their \( u \)-predecessors \( \delta^{-1}(S, u) \) and \( \delta^{-1}(T, u) \), respectively. Suppose that \(|\delta^{-1}(S, u)| = |S|\) and \(|\delta^{-1}(T, u)| = |T|\). Then \(|\delta^{-1}(q, u)| = 1\) for all \( q \in U \), which gives a contradiction with \(|U'| > |U|\).

**Corollary 6.** Let \( S, T \subseteq Q \) be two distinct witnesses. Then \( S \cup T = Q \).

**Proof.** Since \( S \neq T \), the union \( S \cup T \) is larger than the witness size \(|S| = |T|\), thus it must be reachable. If \( S \cup T \neq Q \), then \( S \cup T \) has a larger predecessor. By Lemma 5 either \( S \) or \( T \) also has a larger predecessor, which contradicts that they are witnesses.
We can now focus on detecting sets that are potential witnesses. We relax a bit the required property for being a witness and introduce an auxiliary definition:

Definition 7 (Witness candidate). A non-empty set $S \subseteq Q$ is a witness candidate if it does not have a larger predecessor and all its predecessors (which are of the same size) have pairwise disjoint complements.

Every witness is a witness candidate, and clearly, every witness candidate is unreachable (moreover, it is unreachable from any larger set). However, both converses do not necessarily hold.

Example 8. The automaton from Figure 1 is not completely reachable.

- Every set of size 5 is reachable; e.g., $\delta(Q, ab) = Q \setminus \{q_0\}$.
- Witnesses have size 4 and they are: $Q \setminus \{q_i, q_{(i+3) \mod 6}\}$ for $i \in \{0, 1, 2\}$.
- The sets $\{q_i, q_{(i+3) \mod 6}\}$ for $i \in \{0, 1, 2\}$ are witness candidates but not witnesses, as they have size 2.
- The sets $\{q_i, q_{(i+1) \mod 6}, q_{(i+2) \mod 6}\}$ for $i \in \{0, 1, 2\}$ are reachable.
- Singletons are unreachable, but they have a larger predecessor, e.g., $\delta^{-1}(\{q_0\}, ab) = \{q_2, q_3\}$; so they are not witness candidates.
- If we add a letter $c$ such that $\delta(Q, c) = \{q_0\}$, then singletons will be reachable and every proper subset of $Q$ will admit an extending word. But witnesses and witness candidates will remain the same, as these extending words will not be properly extending.

Lemma 9. A witness is a witness candidate.

Proof. Let $S$ be a witness and let $T$ and $T'$ be two distinct predecessors of $S$. As they are of the same size $|T| = |T'| = |S|$ and $S$ is reachable from both of the predecessors, they are also witnesses. Note that $T \cap T' = \emptyset$ is equivalent to $T \cup T' = Q$. Therefore, by Lemma 5 we must have $T \cap T' = \emptyset$, thus $S$ is a witness candidate.

We note that the second condition in Definition 7 is necessary, since then verifying whether a set is a witness candidate can be done efficiently. Although for determining that an automaton is not completely reachable, it would be enough to find a set without a larger predecessor, checking this condition in general is PSPACE-complete.$^1$

$^1$ A proof will be added in a next revision of this paper.
2.3 An Algorithm in co-NP

We can now develop a polynomial procedure that checks whether a given set \( S \) is a witness candidate. The verification function is shown in Algorithm 1. Starting from \( S \), we process all its predecessors in a breath-first search manner; for this, a queue \( \text{Process} \) is used. A next predecessor \( T \) is taken in line 6. Then, we check whether \( T \) is larger than \( S \), which directly implies that \( S \) is not a witness candidate. The second check in line 9 verifies whether \( T \cap T' \neq \emptyset \), for some previously processed set \( T' \). For this, we maintain \( \text{Absent} \) array, which for every state \( q \) indicates whether \( q \) occurred in the complement of some previously processed set, and if so, this set is stored in \( \text{Absent}[q] \). We also use this to check whether the same set \( T \) has not been processed previously (line 11), which is possible through different words.

When all predecessors of \( S \) were considered and neither of the two conditions occurred, the function reports that \( S \) is a witness candidate.

Algorithm 1 Verifying whether a given subset is a witness candidate from Definition 7.

**Input:** An \( n \)-state automaton \( \mathcal{A} = (Q, \Sigma, \delta) \) and a non-empty \( S \subseteq Q \).

**Output:** true if \( S \) is a witness candidate; false otherwise.

1: function \( \text{IsWitnessCandidate}(\mathcal{A}, S) \)
2: \( \text{Process} \leftarrow \text{EMPTYFifoQueue()} \) \( \triangleright \) It contains predecessors of \( S \) to be processed
3: \( \text{Process}.\text{PUSH}(S) \)
4: \( \text{Absent} \leftarrow \text{Array indexed by } q \in Q \text{ initialized with none} \)
5: while not \( \text{Process}.\text{EMPTY()} \) do
6: \( T \leftarrow \text{Process}.\text{POP()} \)
7: if \( |T| > |S| \) then
8: return false \( \triangleright T \) is a larger predecessor of \( S \)
9: if \( \text{Absent}[q] \neq \text{none} \) for some \( q \in T \) then \( \triangleright \) Then \( T \cup T' \neq Q \)
10: \( T' \leftarrow \text{Absent}[q] \)
11: if \( T = T' \) then
12: continue \( \triangleright T \) has been processed previously
13: return false \( \triangleright T \) or \( T' \) has a larger predecessor by Lemma 5
14: for all \( q \in T \) do
15: \( \text{Absent}[q] \leftarrow T \) \( \triangleright \text{We know that } \text{Absent}[q] \text{ was none} \)
16: for all \( a \in \Sigma \) do
17: if \( \delta^{-1}(T, a) \) is \( a \)-predecessor of \( T \) then
18: \( \text{Process}.\text{PUSH}(\delta^{-1}(T, a)) \)
19: return true \( \triangleright \) All predecessors of \( S \) were checked

\( \triangleright \text{Lemma 10. Function } \text{IsWitnessCandidate} \text{ from Algorithm 1 is correct and works in } O(n^2 \cdot |\Sigma|) \text{ time.} \)

**Proof.** The function ends in line 8, 13, or 19. The first case with \( |T| > |S| \) is straightforward, as a witness candidate cannot have a larger predecessor. The second case is that there is some \( q \in T \cap T' \), where \( T' = \text{Absent}[q] \) was processed previously and \( T \neq T' \). Again \( T \cap T' \neq \emptyset \) implies that \( S \) is not a witness candidate.

The function ends positively (in line 19) when there are no more predecessors of \( S \) to consider (\( \text{Process} \) becomes empty). As we checked all predecessors of \( S \), they satisfy the conditions from Algorithm 1.
Finally, we bound the running time. The number of processed predecessors (including \( S \) itself) of the same size \(|S| < n\) with pairwise disjoint complements is at most \( \lceil n/(n - |S|) \rceil \). This bounds the number of iterations of the main loop (line 5); there may be one additional iteration at the end with \( T \) violating a condition for a witness candidate. One iteration takes \( O(n \cdot |\Sigma|) \) time: lines 6–13 trivially take \( O(n) \) time, updating \( \text{Absent} \) in lines 14–15 also takes \( O(n) \) time if we store a pointer/reference to \( T \), and computing direct (i.e., one-letter) predecessors in lines 13–15 takes \( O(n \cdot |\Sigma|) \) time. Therefore, the function works in \( O(n^2 \cdot |\Sigma|/(n - |S|)) \) time.

\[\Box\]

**Theorem 11.** Problem **Completely Reachable** can be solved in \( \text{co-NP} \).

**Proof.** We guess some witness – a non-empty subset \( S \subseteq Q \) and call \( \text{IsWitnessCandidate}(\mathcal{A}, S) \). The algorithm returns the negated answer of the function.

If the automaton is not completely reachable, then there exists some witness. The function returns \( \text{true} \) for it, so the algorithm can guess the witness and answer negatively. Conversely, if the automaton is completely reachable, then there are no unreachable non-empty sets and thus no witness candidates. For every non-empty \( S \subseteq Q \), the function returns \( \text{false} \), so the algorithm must answer positively.

\[\Box\]

### 2.4 A Polynomial-Time Algorithm

The overall idea is as follows. We replace guessing a witness with a constructive procedure. We expand the function from [Algorithm 1] so that it not just returns a Boolean answer but also finds a properly extending word for \( S \) if it exists. This works under a certain assumption for larger sets. Then, the function is used to hunt for a witness, starting from all the sets of size \( n - 1 \) and reducing them iteratively in such a way that a witness will be found if it exists.

#### 2.4.1 Finding Properly Extending Words

The function is shown in [Algorithm 2]. Similarly to \( \text{IsWitnessCandidate} \), it verifies whether \( S \) is a witness candidate, but if not, it additionally finds a properly extending word for \( S \). Now, together with predecessor sets \( T \) of \( S \), we also keep track of the words \( w \) such that \( \delta(T, w) = S \). The main difference with \( \text{IsWitnessCandidate} \) is the case in line 9. For the union \( T \cup T' \), we aim at finding a properly extending word for it, which then turns out to be properly extending either for \( T \) or \( T' \); this is done by a recursive call in line 13. This call could fail in general, when \( T \cup T' \) or some larger superset is a witness candidate. Hence, the function requires \( \text{Assumption 1} \), which excludes this possibility: all subsets larger than \(|S|\) have a larger predecessor, except for \( Q \). Recall that \( \text{Assumption 1} \) is necessary, as without it, the problem of finding a properly extending word is \( \text{PSPACE-hard} \).

**Example 12.** Consider the automaton from [Figure 2]; this is the example of a completely reachable binary automaton from [\text{[6]} Fig. 3].

Let \( S = \{q_0, q_{10}\} \). The execution of \( \text{FindProperlyExtendingWord()} \) is summarized as follows:
**Algorithm 2**  An algorithm finding a properly extending word (a larger predecessor) for a given set.

**Input:** An $n$-state automaton $\mathcal{A} = (Q, \Sigma, \delta)$ and a non-empty $S \subseteq Q$.

**Output:** A properly extending word $w$ of $S$ if it exists, or **none** if $S$ is a witness candidate.

**Require:** [Assumption 1] All $S' \subseteq Q$ such that $|S| < |S'| < n$ have a larger predecessor.

1. **function** `FindProperlyExtendingWord(\mathcal{A}, S)`
2.  
3.  
4.  
5.  
6.  
7.  
8.  
9.  
10.  
11.  
12.  
13.  
14.  
15.  
16.  
17.  
18.  
19.  
20.  
21.  
22.  
23.  
24.  

```
$\delta$ is either two words or one word, depending on which case occurs. $\text{word}$ denotes the number of the recursive call. $\text{Found word}$ is either two words or one word, depending on which case occurs. $\text{Found set}$ is either the union of two sets or a larger predecessor; in any case, it is larger than the input set. $\text{Result}$ is the returned properly extending word, which is either constructed from the recursive call or is known directly. The underlined results are those chosen for constructing the final word.

| Depth | Set   | Found word | Found set | Result |
|-------|-------|------------|-----------|--------|
| 1     | \{q_0, q_1\} | $a \lor b$ | \{q_0, q_1\} $\cup$ \{q_0, q_{11}\} | $ab^2$ |
| 2     | \{q_0, q_9, q_{10}, q_{11}\} | $a \lor b$ | \{q_0, q_9, q_{10}, q_{11}\} $\cup$ \{q_{8}, q_{11}\} | $ab^2$ |
| 3     | \{q_0, q_9, \ldots, q_{11}\} | $a \lor b$ | \{q_0, q_9, \ldots, q_{11}\} $\cup$ \{q_7, q_{11}\} | $ab^2$ |
| 4     | \{q_0, q_7, \ldots, q_{11}\} | $a \lor b$ | \{q_0, q_7, \ldots, q_{11}\} $\cup$ \{q_{6}, q_{11}\} | $ab^2$ |
| 5     | \{q_0, q_6, \ldots, q_{11}\} | $a \lor b$ | \{q_0, q_6, \ldots, q_{11}\} $\cup$ \{q_5, q_{11}\} | $ab^2$ |
| 6     | \{q_0, q_5, \ldots, q_{11}\} | $a \lor b$ | \{q_0, q_5, \ldots, q_{11}\} $\cup$ \{q_4, q_{11}\} | $ab^2$ |
| 7     | \{q_0, q_4, \ldots, q_{11}\} | $a \lor b$ | \{q_0, q_4, \ldots, q_{11}\} $\cup$ \{q_3, q_{11}\} | $ab^2$ |
| 8     | \{q_0, q_3, \ldots, q_{11}\} | $a \lor b$ | \{q_0, q_3, \ldots, q_{11}\} $\cup$ \{q_2, \ldots, q_{11}\} | $ab^2$ |
| 9     | $Q \setminus \{q_1\}$ | $a \lor b$ | $Q$ | $ab$ |
```

$\text{Depth}$ denotes the number of the recursive call. $\text{Set}$ is the input set $S$ for this call. $\text{Found word}$ is either two words or one word, depending on which case occurs. $\text{Found set}$ is either the union of two sets or a larger predecessor; in any case, it is larger than the input set. $\text{Result}$ is the returned properly extending word, which is either constructed from the recursive call or is known directly. The underlined results are those chosen for constructing the final word.
The function works as follows: \( S = \{q_0, q_{10}\} \) does not have a \( a \)-predecessor, because \( \delta^{-1}(q_0, a) \) is empty, but it has the \( b \)-predecessor \( \delta^{-1}(\{q_0, q_{10}\}, b) = \{q_9, 11\} \). The complements of \( \{q_0, q_{10}\} \) and \( \{q_9, q_{11}\} \) are not disjoint, thus there is a recursive call for the union \( \{q_0, q_9, q_{10}, q_{11}\} \). This is repeated 7 times, when the states \( q_8, \ldots, q_2 \) are subsequently added one by one. Finally, we get the set \( Q \{q_1\} \). Its \( b \)-predecessor \( Q \{q_0\} \) is processed and has a disjoint complement with \( Q \{q_1\} \). Then, \( \delta^{-1}(Q \{q_0\}, a) = Q \), thus a larger predecessor is found. Now, the returned word is constructed from the recursive calls. Word \( ab \) found in the last call is properly extending both sets \( \{q_0, q_3, \ldots, q_{11}\} \) and \( Q \{q_0, q_1\} \). Here the function uses the latter, but both choices would work. Thus, the word \( ab^2 \) is properly extending for \( \{q_0, q_3, \ldots, q_{11}\} \). It turns out that in all the previous calls, \( ab^2 \) is properly extending for the input set, thus the word is not prolonged.

**Lemma 13.** Function \textit{FindProperlyExtendingWord} is correct and works in \( O(n^2 \log n \cdot |\Sigma|) \) time. Moreover, the returned word has length \( O(n \log n) \).

**Proof.** The case where \( S \) is a witness candidate follows in the same way as for \textit{IsWitnessCandidate()} (in the proof of Lemma 10). The function checks all its predecessors and returns \textbf{none} in line 24.

Consider a set \( S \) that is not a witness candidate. Then necessarily, the function ends in line 8, line 16, or line 18. In line 8, a larger predecessor is found directly thus the answer is correct. So suppose that the condition in line 9 holds. The recursive call in line 13 returns a word \( u \) that is properly extending \( T \cup T' \subset Q \); this follows by induction on the set size, assuming that \textit{FindProperlyExtendingWord} is correct for larger sets than \( S \). By Lemma 5, \( u \) is properly extending either \( T \) or \( T' \). The function directly checks the first case in line 15, and the second case must hold otherwise. Therefore, the returned word is properly extending \( S \).

Concerning the running time, one call of the function does at most \( \lceil n/(n - |S|) \rceil \) iterations.
An iteration takes $O(n \cdot |\Sigma|)$ time if we do not copy words directly but operate with pointers (for concatenation) and maintain the induced transformations. These give $O(n^2 \cdot |\Sigma|/(n - |S|))$ time. There are also recursive calls for larger subsets. In the worst case, each time the union subset is larger by 1, so by summing the running time from all the calls, we obtain:

$$
\sum_{k=|S|}^{n-1} O(n^2 \cdot |\Sigma|/(n - k)) \leq O(n^2 \cdot |\Sigma| \cdot H_{n-|S|}) \leq O(n^2 \log n \cdot |\Sigma|),
$$

where $H_i$ is the $i$-th harmonic number ($H_i = 1 + \frac{1}{2} + \ldots + \frac{1}{i} \leq \ln(i) + 1$ is a well-known inequality).

To bound the length of the found word, it is enough to sum the upper bounds on the number of iterations (as each iteration increments the processed word by at most one letter) in all recursive calls:

$$
\sum_{k=|S|}^{n-1} \lfloor n/(n - k) \rfloor \leq n H_{n-|S|} + (n - |S|) \leq O(n \log n).
$$

\section{2.4.2 Reduction}

Suppose that given a set $S \subseteq Q$, we search for a witness that is contained within it. Having a properly extending word $w$ for $S$, we can reduce $S$ to its proper subset by excluding some states which surely cannot be in any witness contained in $S$. The next lemma shows the criterion, which is realized by $\text{REDUCE}(\mathcal{A}, S, w)$ in Algorithm 3.

\begin{lemma}
Let $S \subseteq Q$ and $\delta^{-1}(S, w)$ be a $w$-predecessor of $S$ for some $w$. Then for every witness $S' \subseteq S$, every state $q \in S'$ is such that $|\delta^{-1}(q, w)| = 1$.
\end{lemma}

\begin{proof}
Since $\delta^{-1}(S, w)$ is a $w$-predecessor of $S$, by Remark 1, we have $\delta^{-1}(q, w) \neq \emptyset$ for every state $q \in S$. Suppose for a contradiction that there exists $S' \subseteq S$ that is a witness and contains a state $p \in S'$ such that $|\delta^{-1}(p, w)| > 1$. Let $T' = \delta^{-1}(S', w)$. Since $\delta^{-1}(q, w) \neq \emptyset$ for all $q \in S'$, $T'$ is a $w$-predecessor of $S'$. Because $|\delta^{-1}(p, w)| > 1$, we have $|T'| > |S'|$, so $T'$ must be reachable as it is larger than the witness $S'$. But then also $S'$ is reachable through $T'$, contradicting the property of $S'$.
\end{proof}

\begin{algorithm}
\caption{Reducing a set for possible witness containment.}
\begin{algorithmic}
\Function{REDUCE}{$\mathcal{A}, S, w$}
\State \Return $\{q \in S : |\delta^{-1}(q, w)| = 1\}$
\EndFunction
\end{algorithmic}
\end{algorithm}

\section{2.4.3 Finding a Witness}

We have all ingredients to build a polynomial algorithm that solves the decision problem $\text{COMPLETELY} \ \text{REACHABLE}$. It is shown in Algorithm 4. If the automaton is not completely reachable, the algorithm also finds a witness. Starting from all sets of size $n-1$, we process
Algorithm 4 A polynomial-time algorithm verifying the complete reachability of an automaton or finding a witness.

**Input:** An $n$-state automaton $\mathcal{A} = (Q, \Sigma, \delta)$.

**Output:** none if $\mathcal{A}$ is completely reachable; a witness otherwise.

1: function FindWitness($\mathcal{A}$)
2: Queue ← EmptyPriorityQueue()  $\triangleright$ Ordered by set size; the largest sets go first
3: for all $q \in Q$ do
4:  Queue.Push($Q \setminus \{q\}$)
5: while not Queue.Empty() do
6:  $S \leftarrow$ Queue.Pop()  $\triangleright$ Get a set of the largest size
7:  $w \leftarrow$ FindProperlyExtendingWord($\mathcal{A}, S$)
8:  if $w = \text{none}$ then
9:    return $S$  $\triangleright$ Witness found
10:  $S' \leftarrow$ Reduce($\mathcal{A}, S, w$)  $\triangleright$ $S' \subseteq S$
11:  if $S' \neq \emptyset$ and not Queue.Contains($S'$) then
12:    Queue.Push($S'$)
13: return none  $\triangleright$ No witnesses

sets in the order of decreasing size. Processing a set consists of finding a properly extending word for it and reducing the set by this word.

▶ Lemma 15. Function FindWitness is correct and works in $O(n^4 \log n \cdot |\Sigma|)$ time.

**Proof.** First note that if there is no witness, then Assumption 1 for FindProperlyExtendingWord is always satisfied, thus this function in line 7 always returns a word. So the algorithm will not return a witness. As the inserted sets $S'$ are proper subsets of $S$, all the sets are eventually reduced to the empty set, thus the algorithm terminates and returns none.

Suppose that there is a witness $W \subseteq Q$. We prove by induction that before each iteration of the main loop in line 5, $W$ is contained in one of the sets in Queue. For the first iteration the statement holds: since $W \neq Q$, we have $W \subseteq Q \setminus \{q\}$ for some $q \in Q$. It is enough to ensure that an iteration does not break the statement. An iteration removes only $S$ from Queue. If there is some witness contained in the other sets from Queue, then we are done; so we further assume that all witnesses are contained in $S$. If $S$ is a witness, then FindProperlyExtendingWord($S$) returns none and the function ends. If a witness is contained in $S$, then from Lemma 14 it is also contained in $S'$, which is then added to Queue.

Concerning the running time, there are at most $n(n-1)$ iterations of the main loop: we start with $n$ sets of size $n-1$, and they are reduced by at least one up to the empty set. The call to FindProperlyExtendingWord in an iteration takes at most $O(n^2 \log n \cdot |\Sigma|)$ time (Lemma 13; the other operations take no more than $O(n^2)$ with a native implementation. Altogether, we have $O(n^4 \log n \cdot |\Sigma|)$.

▶ Remark 16. The number of witnesses of size $k$ is at most $\lfloor n/k \rfloor$, as their complements must be pairwise disjoint. Function FindWitness can be easily modified to find all the witnesses: after finding the first witness of size $k$, we can continue the main loop in line 5 until all sets of size $k$ are processed.
3 An Upper Bound on Reset Threshold

3.1 Synchronization

A reset word is a word \( w \) such that \( |\delta(Q, w)| = 1 \). Equivalently, we have \( \delta^{-1}(q, w) = Q \) for some \( q \in Q \). If an automaton admits a reset word, then it is called synchronizing and its reset threshold is the length of the shortest reset words.

The central problem in the theory of synchronizing automata is the famous Černý conjecture, which states that every synchronizing \( n \)-state automaton has its reset threshold at most \( (n - 1)^2 \).

Obviously, completely reachable automata are synchronizing. The previously proposed upper bound on their reset threshold is \( \frac{7}{48}n^3 + O(n^2) \) [3], which is obtained through the technique of avoiding [16], that is, it follows in particular from the fact that every set of size \( n - 1 \) is reachable with a word of length at most \( n \).

3.2 Finding Short Properly Extending Words

For a completely reachable automaton, function \text{FindProperlyExtendingWord}() always finds a properly extending word. Therefore, using the well-known extension method (e.g., [17]), we can construct a synchronizing word starting from some singleton \( \{q\} \) and iteratively increasing the set by at least one, finally obtaining \( Q \). This is an easy way to get the upper bound of order \( O(n^2 \log n) \): the length of a word returned by \text{FindProperlyExtendingWord}(S) is upper bounded by \( O(n \log n) \) (Lemma 13). However, we can do better.

The idea is to keep track of all subsets for an intersection of complements, instead of starting an independent search for a properly extending word recursively. This modification is shown in [Algorithm 5] The correctness follows from our earlier arguments.

3.2.1 Nested Boxes

To bound the length of the found word, we consider an auxiliary combinatorial problem.

Consider an \( n \)-element universe \( Q \). Two subsets \( S, T \subseteq Q \) are colliding if \( S \cap T \notin \{\emptyset, S, T\} \). Thus, colliding sets have a non-trivial intersection. A family of non-empty subsets of \( Q \) is called non-colliding if all the subsets are pairwise non-colliding.

\( \triangleright \) \textbf{Definition 17.} For an \( n \geq 1 \), the number \( \text{MaxNestedBoxes}(n) \) is the maximum size of a non-colliding family for an \( n \)-element universe.

The problem is equivalent to, e.g., the maximum number of boxes for \( n \) items, such that a box must contain either an item or at least two boxes. Another interpretation is the maximum number of balanced non-trivial parentheses for an \( n \)-element string, where a pair is trivial if it is empty inside or contains only one pair.

\( \triangleright \) \textbf{Lemma 18.} \( \text{MaxNestedBoxes}(n) = 2n - 1 \).

\textbf{Proof.} It goes by induction on \( n \). For the base case, \( \text{MaxNestedBoxes}(1) = 1 \).

For \( n > 1 \), we can always take the set with all \( n \) elements, and there should be exactly two its proper subsets that are maximal with respect to inclusion. Otherwise, if there are more, we could add a union of two of them and obtain more sets. Thus, the problem is
**Algorithm 5** An algorithm finding a short properly extending word.

**Input:** An $n$-state completely reachable automaton $\mathcal{A} = (Q, \Sigma, \delta)$ and a non-empty $S \subseteq Q$.

**Output:** A properly extending word $w$ of $S$.

1: function FINDSHORTPROPERLYEXTENDINGWORD($\mathcal{A}, S$)
2:   $Trace \leftarrow$ EMPTYMAP() \> For a processed set, it stores a word or two sets
3:   $S' \leftarrow S$
4:   $Process \leftarrow$ EMPTYFIFOQUEUE() \> It contains pairs $(T, w)$ such that $\delta(T, w) = S'$
5:   $Process$.Push(($S', \varepsilon$))
6:   while not $Process$.EMPTY() do
7:     $(T, w) \leftarrow Process$.Pop()
8:     if $Trace[T]$ is defined then
9:       continue
10:      $Trace[T] \leftarrow w$
11:     if $|T| > |S'|$ then
12:       $S' \leftarrow T$
13:       break
14:   while there is $T'$ such that $Trace[T']$ is defined and $T \subseteq T \cup T' \subseteq Q$ do
15:     $U \leftarrow T \cup T'$ \> $|U| > |S|$
16:     $Trace[U] \leftarrow (T, T')$
17:     $T \leftarrow U$
18:     $S' \leftarrow U$
19:     $w \leftarrow \varepsilon$ \> The empty word
20:   $Process \leftarrow$ EMPTYFIFOQUEUE() \> Continue only from the largest set
21:   for all $a \in \Sigma$ do
22:     if $\delta^{-1}(T, a)$ is a-predecessor of $T$ then
23:       $Process$.Push(($\delta^{-1}(T, a), aw$))
24:     $u \leftarrow \varepsilon$ \> The empty word
25:   while $S' \neq S$ do
26:     if $Trace[S']$ is a word then
27:       $w \leftarrow Trace[S']$
28:       $S' \leftarrow \delta(S', w)$
29:       $u \leftarrow uw$
30:     else
31:       $(T, T') \leftarrow Trace[S']$
32:       if $|\delta^{-1}(T, u)| > |T|$ then
33:         $S' \leftarrow T$
34:       else
35:         $S' \leftarrow T'$
36:   return $w$

recursive:

$MaxNestedBoxes(n) = 1 + \max_{p \in \{1, \ldots, n-1\}} (MaxNestedBoxes(p) + MaxNestedBoxes(n-p))$

$= 1 + \max_{p \in \{1, \ldots, n-1\}} (2p - 1 + 2(n - p) - 1)$

$= 1 + \max_{p \in \{1, \ldots, n-1\}} (2n - 2)$

$= 1 + 2n - 2 = 2n - 1$. 
The generalized version of the problem limits the maximum size of the subsets:

**Definition 19.** For an \( n \geq 1 \), the number \( \text{MaxNestedBoxes}(n, k) \) is the maximum size of a non-colliding family for an \( n \)-element universe, where each subset from the family has size at most \( k \).

**Lemma 20.** \( \text{MaxNestedBoxes}(n, k) = 2n - \lceil n/k \rceil \).

**Proof.** We prove the equality by induction on \( k \) and on \( n \). The cases with \( k \geq n \) are solved by Lemma 18. Let \( k > n \geq 1 \). Consider a family satisfying the conditions for \( n \) and \( k \) that has size \( \text{MaxNestedBoxes}(n, k) \).

If the largest subsets have size \( k \), then we split the problem:

\[
\text{MaxNestedBoxes}(n, k) = \text{MaxNestedBoxes}(k) + \text{MaxNestedBoxes}(n - k, k)
\]

By Lemma 18 and the inductive assumption for \( \text{MaxNestedBoxes}(n - k, k) \):

\[
\text{MaxNestedBoxes}(k) + \text{MaxNestedBoxes}(n - k, k) = 2k - 1 + 2(n - k) - \lceil (n - k)/k \rceil = 2n - \lceil n/k \rceil.
\]

If the largest subsets have size \( k' < k \), then this means from the inductive assumption:

\[
\text{MaxNestedBoxes}(n, k) = \text{MaxNestedBoxes}(n, k') = 2n - \lceil n/k' \rceil \leq 2n - \lceil n/k \rceil.
\]

Thus, the number is not larger than in the previous case, and by taking the maximum from the two possibilities we get the claim.

### 3.2.2 Final Bounding

We apply the above combinatorial problem to upper bound the length of a word found by \texttt{FindShortProperlyExtendingWord}:

**Lemma 21.** For a completely reachable automaton and a non-empty proper subset \( S \subseteq Q \), the word returned by \texttt{FindShortProperlyExtendingWord} from Algorithm 5 has length at most \( \text{MaxNestedBoxes}(n, n - |S|) = 2n - \lceil n/(n - |S|) \rceil \).

**Proof.** Let \( \mathcal{X} \) be the family of subsets that were defined as keys for \( \text{Trace} \) as the last in an iteration of the main loop in line 6 (i.e., defined either in line 16 or line 10 if the while condition was not fulfilled). Let \( \overline{\mathcal{X}} \) denote \( \mathcal{X} \) where each subset is replaced with its complement. Note that \( \overline{\mathcal{X}} \) is a non-colliding family; the loop in line 15 is repeated until a set \( T' \) is found such that its complement does not collide with any complement of a key in \( \text{Trace} \).

The size of \( \overline{\mathcal{X}} \) equals the number of iterations of the main loop. When reconstructing the word, the sum of the lengths of words \( w \) in line 28 is also upper bounded by the number of iterations.

As the family \( \overline{\mathcal{X}} \) is non-colliding and the subsets have size \( \leq n - |S| \), the upper bound follows.

To provide a formula for our bounds, we will need the following auxiliary inequality:

**Lemma 22.** For \( 1 \leq m \leq n - 1 \), we have:

\[
\sum_{k=m}^{n-1} \text{MaxNestedBoxes}(n, n - k) \leq (n - m)2n - n\ln(n - m) - n/(n - m).
\]
Proof.

\[
\sum_{k=m}^{n-1} \text{MaxNestedBoxes}(n, n - k) = (n - m)2n - \sum_{k=m}^{n-1} \left\lfloor n/(n-k) \right\rfloor \\
\leq (n - m)2n - \sum_{k=m}^{n-1} n/(n-k) \\
= (n - m)2n - nH_{n-m},
\]

where \( H_i \) is the \( i \)-th harmonic number. Using the well-known inequality \( H_i \geq \ln i + 1/i \), we get the upper bound:

\[
(n - m)2n - nH_{n-m} \leq (n - m)2n - n(ln(n - m) + 1/(n - m)) \\
= (n - m)2n - n\ln(n - m) - n/(n - m).
\]

\[\Box\]

Using the standard extension method, we get:

\[\Box\textbf{Theorem 23.} \text{ For a completely reachable } n\text{-state automaton } (Q, \Sigma, \delta), \text{ every non-empty proper subset } S \subseteq Q \text{ is reachable with a word of length at most } (n - |S|)2n - n\ln(n - m) - n/(n - |S|) < 2n\ln(n - |S|).
\]

\textbf{Proof.} Starting from } S \text{, we apply at most } n - |S| \text{ times a properly extending word found by } \text{FindShortProperlyExtendingWord}, \text{ which, by Lemma } \text{11} \text{ is bounded by } \text{MaxNestedBoxes}(n, n - k), \text{ where } k = |S|, \ldots, n - 1. \text{ The bound on the sum is provided by Lemma } \text{22} \text{.}

\[\Box\]

\[\Box\textbf{Theorem 24.} \text{ The reset threshold of a completely reachable automaton with } n \geq 3 \text{ states is at most } (n - 2)2n - n\ln(n - 2) - n/(n - 2) < 2n^2 - n\ln n - 4n + 2.
\]

\textbf{Proof.} We find a state } q \text{ such that } |\delta^{-1}(\{q\}, a)| \geq 2 \text{ for some letter } a. \text{ Then we extend } \delta^{-1}(\{q\}, a) \text{ by the word from Theorem } \text{23} \text{. Since } \ln n - \ln(n - 2) \leq 2 \text{ for } n \geq 3, \text{ we also get the inequality.}

\[\Box\]

\textbf{References}

1. D. S. Ananichev, M. V. Volkov, and V. V. Gusev. Primitive digraphs with large exponents and slowly synchronizing automata. \textit{Journal of Mathematical Sciences}, 192(3):263–278, 2013.
2. M. V. Berlinkov, R. Ferens, and M. Szykuła. Preimage problems for deterministic finite automata. \textit{Journal of Computer and System Sciences}, 115:214–234, 2021.
3. E. A. Bondar, D. Casas, and M. V. Volkov. Completely reachable automata: an interplay between automata, graphs, and trees, 2022. URL: \texttt{https://arxiv.org/abs/2205.09404}.
4. E. A. Bondar and M. V. Volkov. Completely Reachable Automata. In Cezar Câmpeanu, Florin Manea, and Jeffrey Shallit, editors, \textit{DCFS}, pages 1–17. Springer, 2016.
5. E. A. Bondar and M. V. Volkov. A Characterization of Completely Reachable Automata. In Mizuho Hoshi and Shinmasu Seki, editors, \textit{DLT}, pages 145–155. Springer, 2018.
6. D. Casas and M. V. Volkov. Binary completely reachable automata. In \textit{LATIN}. Springer, 2022. To appear; full version at \texttt{https://arxiv.org/abs/2205.09404}.
7. J. Černý. Poznámka k homogénnym experimentom s konečnými automatami. \textit{Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied}, 14(3):208–216, 1964. In Slovak.
8. H. Don. The Černý Conjecture and 1-Contracting Automata. \textit{Electronic Journal of Combinatorics}, 23(3):P3.12, 2016.
9 R. Ferens, M. Szykuła, and V. Vorel. Lower Bounds on Avoiding Thresholds. In MFCS, volume 202 of LIPIcs, pages 46:1–46:14. Schloss Dagstuhl, 2021.

10 F. Gonze, V. V. Gusev, B. Gerencser, R. M. Jungers, and M. V. Volkov. On the interplay between Babai and Černý’s conjectures. In DLT, volume 10396 of LNCS, pages 185–197. Springer, 2017.

11 F. Gonze and R. M. Jungers. Hardly Reachable Subsets and Completely Reachable Automata with 1-Deficient Words. Journal of Automata, Languages and Combinatorics, 24(2–4):321–342, 2019.

12 S. Hoffmann. Completely Reachable Automata, Primitive Groups and the State Complexity of the Set of Synchronizing Words. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, LATA, LNCS, pages 305–317. Springer, 2021.

13 J.-E. Pin. On two combinatorial problems arising from automata theory. In Proceedings of the International Colloquium on Graph Theory and Combinatorics, volume 75 of North-Holland Mathematics Studies, pages 535–548, 1983.

14 I.K. Rystsov. Estimation of the length of reset words for automata with simple idempotents. Cybern. Syst. Anal. 36, pages 339–344, 2000.

15 Y. Shitov. An Improvement to a Recent Upper Bound for Synchronizing Words of Finite Automata. Journal of Automata, Languages and Combinatorics, 24(2–4):367–373, 2019.

16 M. Szykuła. Improving the Upper Bound on the Length of the Shortest Reset Word. In STACS 2018, LIPIcs, pages 56:1–56:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

17 M. Volkov. Synchronizing automata and the Černý conjecture. In Language and Automata Theory and Applications, volume 5196 of LNCS, pages 11–27. Springer, 2008.