Matching linear and non-linear trace patterns with regular policies

Franz Baader[1⋆], Andreas Bauer[2], and Alwen Tiu[2]

[1] Theoretical Computer Science, TU Dresden, Germany
baader@inf.tu-dresden.de
[2] Computer Sciences Laboratory, ANU, Canberra, Australia
{baueran,tiu}@rsise.anu.edu.au

Abstract. In this paper, we consider policies that are described by regular languages. Such regular policies L are assumed to describe situations that are problematic, and thus should be avoided. Given a trace pattern u, i.e., a sequence of action symbols and variables, were the variables stand for unknown (i.e., not observed) sequences of actions, we ask whether u potentially violates a given policy L, i.e., whether the variables in u can be replaced by sequences of actions such that the resulting trace belongs to L. We determine the complexity of this violation problem, depending on whether trace patterns are linear or not, and on whether the policy is assumed to be fixed or not.

1 Introduction

In an online transaction system, policies that define which sequences of actions (called traces in the following) are viewed as being problematic can be specified using regular languages over the alphabet of action symbols. A trace w violates the policy L if $w \in L$. Sometimes, it is not possible to observe all the actions that take place. For example, assume that an online auctioning firm such as ebay.com is trying to monitor the behaviour of its buyers and sellers w.r.t. certain security policies. Then some of the actions (like making a bid or giving a positive evaluation of the seller/buyer) can be observed by ebay, whereas other actions (like actually sending the goods or paying for received goods) are not observable. We model this with the help of trace patterns, i.e., sequences of actions and variables, where the variables stand for unknown sequences of actions. For example, $abXaY$ is a trace pattern where a, b are actions (more precisely, symbols for actions) and X, Y are variables. This trace pattern says: all we know about the actual trace is that it starts with ab, is followed by some trace w, which is followed by a, which is in turn followed by some trace u. Given such a trace pattern, all traces that can be obtained from it by replacing its variables with traces (i.e., finite sequences of actions) are possibly the actual traces. In our example, these are all the traces of the form $abwau$ where w and u are arbitrary traces. The policy L is potentially violated if one of the traces obtained by such

⋆ Supported by NICTA, Canberra Research Lab.

a substitution of the variables by traces belongs to L. In our example, $abXaY$ potentially violates $L = (ab)^*$ since replacing X by ab and Y by b yields the trace $ababab \in L$.

The trace pattern in our examples is linear since every variable occurs at most once in it. We can also consider non-linear trace patterns such as $abXaX$, where different occurrences of the same variable must be replaced by the same trace. The underlying idea is that, though we do not know which actions took place in the unobserved part of the trace, we know (from some source) that the same sequence of actions took place. It is easy to see that the policy $L = (ab)^*$ is not potentially violated by the non-linear trace pattern $abXaX$ since it is not possible to replace X by a trace w such that $abwaw \in L$.

In this paper, we will show that the complexity of the problem of deciding whether a given trace pattern potentially violates a regular policy depends on whether the trace pattern is linear or not. For linear trace patterns, the problem is decidable in polynomial time whereas for non-linear trace patterns the problem is PSpace-complete. If we assume that the size of the security policy (more precisely, of a non-deterministic finite automaton or regular expression representing it) is constant, then the problem can be solved in linear time for linear trace patterns and is NP-complete for non-linear trace patterns.

2 Preliminaries

In the following, we consider finite alphabets Σ, whose elements are called *action symbols*. A *trace* is a (finite) word over Σ, i.e., an element of Σ^*. A *trace pattern* is an element of $(\Sigma \cup \mathcal{V})^*$, i.e., a finite word over the extended alphabet $\Sigma \cup \mathcal{V}$, where $\mathcal{V}$ is a finite set of *trace variables*. The trace pattern u is called *linear* if every trace variable occurs at most once in u. A *substitution* is a mapping $\sigma : \mathcal{V} \to \Sigma^*$. This mapping is extended to a mapping $\widehat{\sigma} : (\Sigma \cup \mathcal{V})^* \to \Sigma^*$ in the obvious way, by defining $\widehat{\sigma}(\varepsilon) = \varepsilon$ for the empty word ε, $\widehat{\sigma}(a) = a$ for every action symbol $a \in \Sigma$, $\widehat{\sigma}(X) = \sigma(X)$ for every trace variables $X \in \mathcal{V}$, and $\widehat{\sigma}(uv) = \widehat{\sigma}(u)\widehat{\sigma}(v)$ for every pair of non-empty trace patterns u, v.

A *policy* is a regular language over Σ. We assume that such a policy is given either by a regular expression or by a (non-deterministic) finite automaton. For our complexity results, it is irrelevant which of these representations we actually use.

Definition 1. *Given a trace pattern u and a policy L, we say that u* potentially violates *L (written $u \lesssim L$) if there is a substitution σ such that $\widehat{\sigma}(u) \in L$. The* violation problem *is the following decision problem:*

Given: *A policy L and a trace pattern u.*
Question: *Does $u \lesssim L$ hold or not?*

If the trace pattern u in this decision problem is restricted to being linear, then we call this the linear violation problem.

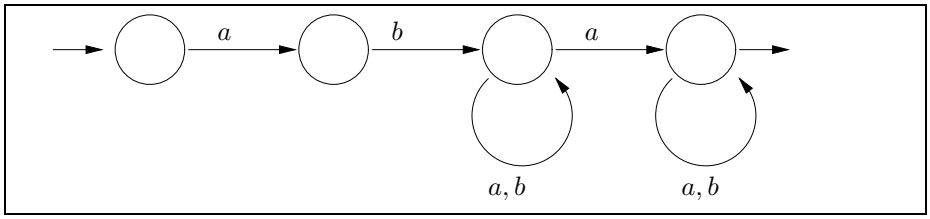

Fig. 1. A non-deterministic finite automaton accepting $ab\Sigma^*a\Sigma^*$.

We assume that the reader is familiar with regular expressions and finite automata. Given a (non-deterministic) finite automaton $\mathcal{A}$, states p, q in $\mathcal{A}$, and a word w, we write $p \rightarrow_{\mathcal{A}}^{w} q$ to say that there is a path in $\mathcal{A}$ from p to q with label w. The set of labels of all paths from p to q is denoted by $L_{p,q}$.

The following problem for regular languages turns out to be closely connected to the violation problem. The *intersection emptiness problem* for regular languages is the following decision problem:

Given: Regular languages $L_1, \ldots, L_n$.
Question: Does $L_1 \cap \ldots \cap L_n = \emptyset$ hold or not?

It is well-known that this problem is PSpace-complete [3, 1], independent of whether the regular languages are given as regular expressions, non-deterministic finite automata, or deterministic finite automata.

In the following, we assume that regular languages are given by a regular expression, a non-deterministic finite automaton, or a deterministic finite automaton.

3 The linear violation problem

Assume that u is a linear trace pattern and L is a regular language. Let the trace pattern u be of the form $u = u_0X_1u_1 \ldots X_mu_m$ where $u_i \in \Sigma^*$ $(i = 0, \ldots, m)$ and $X_1, \ldots, X_m$ are distinct variables. Obviously, we have

$$u \lesssim L \text{ iff } u_0\Sigma^*u_1 \ldots \Sigma^*u_m \cap L \neq \emptyset.$$

If n is the length of $u_0u_1 \ldots u_m$, then we can build a non-deterministic finite automaton $\mathcal{A}$ accepting the language $u_0\Sigma^*u_1 \ldots \Sigma^*u_m$ that has $n + 1$ states. For example, given the linear trace pattern $abXaY$ from the introduction, we consider the language $ab\Sigma^*a\Sigma^*$, where $\Sigma = \{a, b\}$. Fig. 1 shows a non-deterministic finite automaton with 4 states accepting this language.[1] In addition, there is a non-deterministic finite automaton $\mathcal{B}$ accepting L such that the number of states ℓ of $\mathcal{B}$ is polynomial in the size of the original representation for L.[2] By

[1] Note that arrows without a source pointing into a state denote initial states, and arrows without a target coming out of a state denote final states.

[2] In fact, it is well-known that, given a regular expression r for L, one can construct a non-deterministic finite automaton accepting L in time polynomial in the size of r.

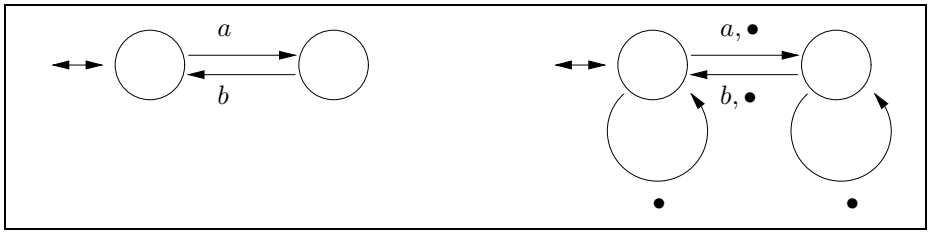

Fig. 2. A non-deterministic finite automaton $\mathcal{A}$ accepting $(ab)^*$ (left) and the corresponding automaton $\widehat{\mathcal{A}}$ (right).

constructing the product automaton of $\mathcal{A}$ and $\mathcal{B}$, we obtain a non-deterministic finite automaton accepting $u_0\Sigma^*u_1 \dots \Sigma^*u_m \cap L$ with $(n+1)\cdot\ell$ states. Thus, emptiness of this language can be tested in time polynomial in $(n+1)\cdot\ell$, and thus in time polynomial in the size of the input u, L of our linear violation problem.

Theorem 1. *The linear violation problem can be solved in polynomial time.*

In the following, we describe an alternative way of showing this polynomiality result, which will turn out to be more convenient for proving that the problem is linear in case the size of the policy L is assumed to be constant.

Let $\bullet$ be an action symbol not contained in Σ. Given a linear trace pattern u, let $\widehat{u}$ denote the trace over the alphabet $\Sigma^\bullet := \Sigma \cup \{\bullet\}$ obtained from u by replacing every variable in u by $\bullet$. Now, assume that $\mathcal{A}$ is a non-deterministic finite automaton accepting the policy L. We transform $\mathcal{A}$ into a non-deterministic finite automaton $\widehat{\mathcal{A}}$ by adding to the transitions of $\mathcal{A}$ all transitions $(p, \bullet, q)$ such that there is $u \in \Sigma^*$ with $p \rightarrow^u_{\mathcal{A}} q$. Fig. 2 illustrates this construction for the policy $L = (ab)^*$.

The following is an easy consequence of the way $\widehat{u}$ and $\widehat{\mathcal{A}}$ have been constructed.

Lemma 1. *Let u be a linear trace pattern, and L a policy that is accepted by the non-deterministic finite automaton $\mathcal{A}$. Then,*

$$u \lesssim L \quad \textit{iff} \quad \widehat{u} \textit{ is accepted by } \widehat{\mathcal{A}}.$$

For example, we have already seen in the introduction that the linear trace pattern $u = abXaY$ violates the policy $L = (ab)^*$. The trace over $\Sigma^\bullet$ corresponding to u is $\widehat{u} = ab\bullet a\bullet$. Obviously, $\widehat{u}$ is accepted by the non-deterministic finite automaton $\widehat{\mathcal{A}}$ in Fig. 2.

The above lemma reduces the violation problem to the word problem for the automaton $\widehat{\mathcal{A}}$. It is well-know that, given a non-deterministic finite automaton $\mathcal{B}$ of size m (where the size of $\mathcal{B}$ is the sum of the number of states and the number of transitions of $\mathcal{B}$) and a word w of length n, the question whether w is accepted by $\mathcal{B}$ can be decided in $O(n \cdot m)$. This yields an alternative proof of Theorem 1. In fact, the size of $\widehat{\mathcal{A}}$ is polynomial in the size of $\mathcal{A}$ (and it can be

computed in polynomial time), and the length of $\widehat{u}$ is the same as the length of u. In addition, if $\mathcal{A}$ is assumed to be constant, then the size of $\widehat{\mathcal{A}}$ is also constant (and it can be computed in constant time).

Theorem 2. *Assume that the policy is fixed. Then, the linear violation problem can be solved in time linear in the length of the input trace pattern.*

4 The general violation problem

Allowing also the use of non-linear patterns increases the complexity of the violation problem.

Theorem 3. *The violation problem is PSpace-complete.*

Proof. PSpace-hardness can be shown by a reduction of the intersection emptiness problem for regular languages. Given regular languages $L_1, \ldots, L_n$, we construct the trace pattern $u_n := \#X\#X \ldots \#X\#$ of length $2n + 1$ and the policy $L(L_1, \ldots, L_n) := \#L_1\#L_2 \ldots \#L_n\#$. Here X is a variable and $\#$ is a new action symbol not occurring in any of the words belonging to one of the languages $L_1, \ldots, L_n$. Obviously, both u_n and (a representation of) $L(L_1, \ldots, L_n)$ can be constructed in time polynomial in the size of (the representation of) $L_1, \ldots, L_n$. To be more precise regarding the representation issue, if we want to show PSpace-hardness for the case where the policy is given by a regular expression (a non-deterministic finite automaton, a deterministic finite automaton), then we assume that the regular languages $L_1, \ldots, L_n$ are given by the same kind of representation. It is easy to see that the following equivalence holds:

$$L_1 \cap \ldots \cap L_n \neq \emptyset \text{ iff } u_n \lesssim L(L_1, \ldots, L_n).$$

Thus, we have shown that the intersection emptiness problem for regular languages can be reduced in polynomial time to the violation problem. Since the intersection emptiness problem is PSpace-complete (independent of whether the regular languages are given as regular expressions, non-deterministic finite automata, or deterministic finite automata), this shows that the violation problem is PSpace-hard (also independent of whether the policy is given as a regular expression, a non-deterministic finite automaton, or a deterministic finite automaton).

To show *membership* of the violation problem *in PSpace*, consider the violation problem for the trace pattern u and the policy L. Let n be the length of u and $\mathcal{A}$ a non-deterministic finite automaton accepting L. For $i \in \{1, \ldots, n\}$, we denote the symbol in $\Sigma \cup \mathcal{V}$ occurring at position i in u with u_i, and for every variable X occurring in u, we denote the set of positions in u at which X occurs with P_X, i.e., $P_X = \{i \mid 1 \leq i \leq n \wedge u_i = X\}$.

It is easy to see that $u \lesssim L$ iff there is a sequence $q_0, \ldots, q_n$ of states of $\mathcal{A}$ such that the following conditions are satisfied:

1. q_0 is an initial state and q_n is a final state;

2. for every $i \in \{1, \ldots, n\}$, if $u_i \in \Sigma$, then $q_{i-1} \to_{\mathcal{A}}^{u_i} q_i$;
3. for every variable X occurring in u, we have

$$\bigcap_{i \in P_X} L_{q_{i-1}, q_i} \neq \emptyset.^3$$

In fact, if $u \lesssim L$, then there is a substitution σ with $\widehat{\sigma}(u) \in L$. Thus, $\widehat{\sigma}(u)$ is accepted by $\mathcal{A}$, which yields a sequence $q_0, \ldots, q_n$ of states of $\mathcal{A}$ such that q_0 is an initial, $q_{i-1} \to_{\mathcal{A}}^{\widehat{\sigma}(u_i)} q_i$, and q_n is a final state. In particular, this shows that the sequence satisfies Condition 1 from above. If $u_i \in \Sigma$, then $\widehat{\sigma}(u_i) = u_i$, which shows that Condition 2 from above is also satisfied. Finally, if $u_i = X \in \mathcal{V}$, then $\widehat{\sigma}(u_i) = \sigma(X)$, and thus $q_{i-1} \to_{\mathcal{A}}^{\widehat{\sigma}(u_i)} q_i$ implies that $\sigma(X) \in L_{q_{i-1}, q_i}$. Since this holds for all $i \in P_X$, this shows that $\sigma(X) \in \bigcap_{i \in P_X} L_{q_{i-1}, q_i}$. Consequently, Condition 3 is satisfied as well.

Conversely, assume that $q_0, \ldots, q_n$ is a sequence of states of $\mathcal{A}$ satisfying the Conditions 1–3 from above. By Condition 3, for every variable X occurring in u, there is a trace $s_X \in \bigcap_{i \in P_X} L_{q_{i-1}, q_i}$. If we define the substitution σ such that $\sigma(X) = s_X$, then it is easy to see that $\widehat{\sigma}(u)$ is accepted by $\mathcal{A}$. Thus, we have $u \lesssim L$.

Based on this characterisation of "$u \lesssim L$" we can obtain a PSpace decision procedure for the violation problem as follows. This procedure is non-deterministic, which is not a problem since NPSpace = PSpace by Savitch's theorem [5]. It guesses a sequence $q_0, \ldots, q_n$ of states of $\mathcal{A}$, and then checks whether this sequence satisfies the Conditions 1–3 from above. Obviously, the first two conditions can be checked in polynomial time, and the third condition can be checked within PSpace since the intersection emptiness problem for regular languages is PSpace-complete. □

Alternatively, we could have shown membership in PSpace by reducing it to *solvability of word equations with regular constraints* [6]. In the terminology of this paper, this problem can be defined as follows:

Given: Trace patterns u, v and for every variable X occurring in u or v a regular language L_X.
Question: Is there a substitution σ such that $\widehat{\sigma}(u) = \widehat{\sigma}(v)$ and $\sigma(X) \in L_X$ for all variables X occurring in u or v?

This problem is known to be PSpace-complete [4]. The violation problem can be reduced to it (in polynomial time) as follows. Let u be a trace pattern and L a regular language. We take a new variable X (i.e., one not occurring in u), and build the the following word equation with regular constraints: the trace patterns to be unified are u and X, and the regular constraints are given as $L_X = L$ and $L_Y = \Sigma^*$ for all other variables Y. It is easy to see that $u \lesssim L$ iff there is a substitution σ such that $\widehat{\sigma}(u) = \widehat{\sigma}(X)$ and $\sigma(X) \in L$, i.e., if the constructed word equation with regular constraints has a solution.

[3] Recall that, for a given non-deterministic finite automaton $\mathcal{A}$ and states p, q in $\mathcal{A}$, we denote the set of words labeling paths from p to q in $\mathcal{A}$ by $L_{p,q}$.

It should be noted, however, the algorithm for testing solvability of word equations with regular constraints described in [4] is rather complicated and it is not clear how it could be transformed into a "practical" algorithm.

Let us now consider the complexity of the violation problem for the case where the policy is assumed to be fixed. In this case, the NPSpace algorithm described in the proof of Theorem 3 actually becomes an NP algorithm. In fact, guessing the sequence of states $q_0, \ldots, q_n$ can be realized using polynomially many binary choices (i.e., with an NP algorithm), testing Conditions 1 and 2 is clearly polynomial, and testing Condition 3 becomes polynomial since the size of $\mathcal{A}$, and thus of non-deterministic finite automata accepting the languages L_{q_{i-1},q_i}, is constant.

Theorem 4. *If the policy is assumed to be fixed, then the violation problem is in NP.*

The matching NP-hardness result of course depends on the fixed policy. For example, if $L = \Sigma^*$, then we have $u \lesssim L$ for all trace patterns u, and thus the violation problem for this fixed policy can be solved in constant time. However, we can show that there are policies for which the problem is NP-hard. Given a fixed policy L, the *violation problem for* L is the following decision problem

Given: A trace pattern u.
Question: Does $u \lesssim L$ hold or not?

Theorem 5. *There exists a fixed policy such that the violation problem for this policy is NP-hard.*

Proof. To show *NP-hardness*, we use a reduction from the well-known NP-complete problem 3SAT [1]. Let $C = c_1 \wedge \ldots \wedge c_m$ be an instance of 3SAT, and $\mathcal{P} = \{p_1, \ldots, p_n\}$ the set of propositional variables occurring in C. Every 3-clause c_i in C is of the form $c_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$, where the $l_{i,j}$ are literals, i.e., propositional variables or negated propositional variables. In the corresponding violation problem, we use the elements of $\mathcal{V} := \{P_i \mid p_i \in \mathcal{P}\}$ as trace variables, and as alphabet we take $\Sigma := \{\#, \neg, \vee, \wedge, \top, \bot\}$. The positive literal p_i is encoded as the trace pattern $\#P_i\#$ and the negative literal $\neg p_i$ as $\neg\#P_i\#$. For a given literal l, we denote its encoding as a trace pattern by $\iota(l)$. 3-Clauses are encoded as "disjunctions" of the encodings of their literals, i.e., $c_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$ is encoded as $\iota(c_i) = \iota(l_{i,1}) \vee \iota(l_{i,2}) \vee \iota(l_{i,3})$, and 3SAT-problems are encoded as "conjunctions" of their 3-clauses, i.e., if $C = c_1 \wedge \ldots \wedge c_m$, then $\iota(C) = \iota(c_1) \wedge \ldots \wedge \iota(c_m)$.

Our fixed policy describes all situations that can make a 3-clause true. To be more precise, consider $\iota(c) = \iota(l_1) \vee \iota(l_2) \vee \iota(l_3)$ for a 3-clause $c = l_1 \vee l_2 \vee l_3$. If we replace the trace variables in c by either $\top$ or $\bot$, then we get a trace of the form $w_1 \vee w_2 \vee w_3$ where each w_i belongs to the set

$$K := \{\#\top\#,\ \#\bot\#,\ \neg\#\top\#,\ \neg\#\bot\#\}.$$

Intuitively, replacing the trace variable P_i by $\top$ ($\bot$) corresponds to replacing the propositional variable p_i by true (false). Thus, a substitution σ that replaces trace variables by $\top$ or $\bot$ corresponds to a propositional valuation v_σ. The valuation v_σ makes the 3-clause c true iff $\widehat{\sigma}(\iota(c)) = w_1 \vee w_2 \vee w_3$ is such that there is an $i, 1 \leq i \leq 3$, with $w_i \in \{\#\top\#, \neg\#\bot\#\}$. For this reason, we define

$$T := \{w_1 \vee w_2 \vee w_3 \mid \{w_1, w_2, w_3\} \subseteq K \text{ and there is an } i, 1 \leq i \leq 3, \text{ with } w_i \in \{\#\top\#, \neg\#\bot\#\}\}.$$

To make a conjunction of 3-clauses true, we must make every conjunct true. Consequently, we define our fixed policy L as

$$L_{3SAT} := (T\wedge)^* T.$$

Since T is a finite language, L_{3SAT} is obviously a regular language. NP-hardness of the violation problem for L_{3SAT} is an immediate consequence of the following claim.

Claim For a given 3SAT problem C the following are equivalent:

1. C is satisfiable.
2. $\iota(C) \lesssim L_{3SAT}$.

To show "1 $\rightarrow$ 2," assume that the valuation v satisfies C. Consider the corresponding substitution σ_v that replaces P_i by $\top$ ($\bot$) if $v(p_i)$ is true (false). Then it is easy to see that $\widehat{\sigma}_v(\iota(C)) \in L_{3SAT}$.

Conversely, to show "2 $\rightarrow$ 1," assume that σ is a substitution such that $\widehat{\sigma}(\iota(C)) \in L_{3SAT}$. It is easy to see that the definitions of $\iota(C)$ and L_{3SAT} then ensure that σ replaces trace variables by $\top$ or $\bot$, and that the valuation v_σ corresponding to σ satisfies C. □

5 Conclusion and future work

In this paper, we have assumed that a regular policy L describes situations that are problematic, and thus should be avoided. This motivated our definition of the violation problem, which asks whether a given trace pattern u potentially violates a given policy L, i.e., whether there is a substitution σ with $\widehat{\sigma}(u) \in L$. We have seen that the complexity of the violation problem depends on whether trace patterns are linear or not, and on whether the policy is assumed to be fixed or not.

Alternatively, one could also assume that a regular policy L describes all the admissible situations. In this case, we want to know whether u always adheres to the policy L, i.e., whether $\sigma(u) \in L$ holds for all substitutions σ. Let us write $u \models L$ to denote that this is the case. Obviously, we have $u \models L$ iff not $u \lesssim \Sigma^* \backslash L$, which shows that the two problems can be reduced to each other. However, this reduction is not polynomial. In fact, there cannot be a polynomial time

reduction between the two problems since the adherence problem is intractable even for linear trace pattern, for which the violation problem is tractable. To see this, consider the (linear) trace pattern X and an arbitrary regular language L over the alphabet Σ. Obviously, we have $X \models L$ iff $L = \Sigma^*$. The problem of deciding whether a regular language (given by a regular expression or a nondeterministic finite automaton) is the universal language Σ^* or not is PSpace-complete [1]. Consequently, the adherence problem is PSpace-hard even for linear trace patterns. However, the exact complexity of the problem (for linear and non-linear trace patterns) is still open.

References

1. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman & Co., New York, USA, 1979.
2. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
3. Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 254–266. IEEE Computer Society, 1977.
4. Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. In *40th Annual Symposium on Foundations of Computer Science (FOCS'99)*, pages 495–500. IEEE Computer Society, 1999.
5. Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.
6. Klaus U. Schulz. Makanin's algorithm for word equations - two improvements and a generalization. In *First International Workshop on Word Equations and Related Topics (IWWERT'90)*, volume 572 of *Lecture Notes in Computer Science*, pages 85–150, 1990.