Matching Trace Patterns With Regular Policies

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

[1] TU Dresden, Germany, baader@inf.tu-dresden.de
[2] The Australian National University, {baueran, alwen.tiu}@rsise.anu.edu.au

Abstract. We consider policies that are described by regular expressions, finite automata, or formulae of linear temporal logic (LTL). Such policies 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 also consider the dual case where the regular policy L is supposed to describe all the admissible situations. Here, we want to know whether u always adheres to the given policy L, i.e., whether all instances of u belong to L. We determine the complexity of the violation and the adherence problem, depending on whether trace patterns are linear or not, and on whether the policy is assumed to be fixed or not.

1 Introduction

Regular languages (defined by regular expressions, finite automata, or temporal logics such as LTL) are frequently used in Computer Science to specify the wanted or unwanted behaviour of a system [1, 8, 3, 4]. Such specifications are not only employed in the design phase of the system, where one may try to verify that every execution trace of the system respects the specification [5]. They can also be used in the deployment phase to monitor the actual system trace and raise an alarm if the specification is violated by the behaviour of the system [7, 2]. In many of these applications, one actually employs ω-regular languages to describe the infinite behaviour of a reactive system, but in our intended application domain, considering finite sequences of actions (called traces in the following) appears to be more appropriate.

In online trading systems, like eBay, formal policies can be used to describe sequences of actions that indicate potentially malicious, dishonest, or fraudulent behaviour (see, e.g., [14]). Of course, it is not always easy to define potentially problematic behaviour without creating too many false positives. But even if such a definition is available, detecting that the actual trace indeed violates the given policy is non-trivial due to the fact that the administrator of an online trading platform may not be able to observe all the relevant user actions. For example, payments made through a third-party institution, shipments of goods,

---

etc., cannot be directly observed. Our approach for modelling this situation is that we use regular languages to describe policies, but instead of traces we consider trace patterns, i.e., traces with variables, where the variables stand for unknown sequences of actions. More formally, assume that L is a policy (formally defined as a regular language) describing undesirable sequences of actions.[1] We say that a given trace w violates the policy L if $w \in L$. Checking for a violation is thus just an instance of the word problem for regular languages. If, instead of a trace, we only have a trace pattern, then detecting violations of the policy becomes more complicated. For example, consider the trace pattern $abXaY$, where a, b are action symbols 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 u, which is followed by a, which is in turn followed by some trace v. 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 trace. The policy L is potentially violated if one of the traces obtained by such 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 $abXaY$ 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 $abXaX$.

In this paper, we will show that the complexity 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 policy is fixed, i.e., its size (more precisely, the size of a 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. We also consider the dual case where the regular policy L is supposed to describe all the admissible situations. Here, we want to know whether u always adheres to the given policy L, i.e., whether all instances of u belong to L. For the case of a fixed policy, the adherence problem is coNP-complete for arbitrary trace patterns and linear for linear trace patterns. If the policy is not assumed to be fixed, however, then the adherence problem is PSpace-complete both for linear and non-linear trace patterns. Finally, we consider the case where the policy is given by an LTL formula. If the policy is not assumed to be fixed, then the violation and the adherence problem are PSpace-complete both for linear and non-linear trace patterns. For the case of a fixed policy, the violation (adherence) problem is NP-complete (coNP-complete) for non-linear patterns and it can be solved in linear time for linear patterns.

---

[1] Using regular languages of *finite* words to express policies means that we only monitor safety properties [11]. For more general notions of policies, see [20].

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 $\sigma : \mathcal{V} \rightarrow \Sigma^*$. This mapping is extended to a mapping $\widehat{\sigma} : (\Sigma \cup \mathcal{V})^* \rightarrow \Sigma^*$ 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 variable $X \in \mathcal{V}$, 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 $u \precsim L$) 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 $u \precsim L$ 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.

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 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?

This problem is PSpace-complete [13, 10], independent of whether the languages are given as regular expressions, non-deterministic finite automata, or deterministic finite automata.

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 $u = u_0 X_1 u_1 \ldots X_m u_m$ where $u_i \in \Sigma^*$ $(i = 0, \ldots, m)$ and $X_1, \ldots, X_m$ are distinct variables. Obviously, we have

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

If n is the length of $u_0 u_1 \ldots u_m$, 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
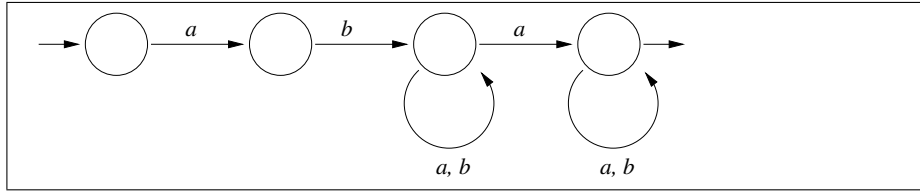
Fig. 1. A non-deterministic finite automaton accepting $ab\Sigma^*a\Sigma^*$.

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

Theorem 1. *The linear violation problem can be solved in polynomial time. If the policy is assumed to be fixed, it can even 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.

Theorem 2. *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),

---

[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.

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 [13] (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 (again independent of the chosen representation).

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 $q_{i-1} \rightarrow^{u_i}_{\mathcal{A}} q_i$;
3. for every variable X occurring in u, we have $\bigcap_{i \in P_X} L_{q_{i-1}, q_i} \neq \emptyset$.[3]

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 [19]. 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 [13].   □

Alternatively, we could have shown membership in PSpace by reducing it to the known PSpace-complete problem of *solvability of word equations with regular constraints* [21, 17]. Due to limited space and the fact that the algorithm for testing solvability of word equations with regular constraints described in [17] is rather complicated and "impractical," we do not describe this reduction here.

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 2 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.

---

[3] Recall that $L_{p,q}$ denotes the set of words labeling paths in $\mathcal{A}$ from p to q.

Theorem 3. *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 4. *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 [10]. 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 $\hat{\sigma}(\iota(c)) = w_1 \vee w_2 \vee w_3$ 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 (easy to prove) fact that C is satisfiable iff $\iota(C) \lesssim L_{3SAT}$. $\square$

5   The adherence problem

Instead of using regular languages to describe traces that are viewed as being problematic, one could assume that a regular policy L describes all the admissible situations. In this case, we want to know whether u always adheres L.

Definition 2. *Given a trace pattern u and a policy L, we say that u* always *adheres to L (written $u \models L$) if $\widehat{\sigma}(u) \in L$ holds for all substitutions σ. The* adherence problem *is the following decision problem:*

Given: *A policy L and a trace pattern u.*
Question: *Does $u \models L$ hold or not?*
If the trace pattern u in this decision problem is restricted to being linear, then we call this the linear adherence problem.

Obviously, we have $u \models L$ iff not $u \lesssim \Sigma^* \setminus L$, which shows that the adherence problem and the complement of the violation problem can be reduced to each other. If the policy L is assumed to be fixed, then these reductions are linear in the length of u. Thus, we obtain the following corollary to our Theorems 1, 3, and 4.

Corollary 1. *Assume that the policy is fixed. Then, the linear adherence problem can be solved in time linear in the length of the input trace pattern. In addition, the general adherence problem is in coNP, and there exists a policy such that the general adherence problem for this policy is coNP-hard.*

Another case for which the above reductions are linear is if the policy is given by a *deterministic* finite automaton. Thus, our Theorems 1 and 2 yield the following corollary.

Corollary 2. *Assume that the policy is given by a deterministic finite automaton. Then, the linear adherence problem can be solved in polynomial time, and the general adherence problem is PSpace-complete.*

If the policy is neither fixed nor given by a deterministic finite automaton, then the reductions between the adherence problem and the complement of the violation problem are not polynomial since they involve the (potentially exponential) construction of the complement automaton for a non-deterministic finite automaton. In fact, in this case there cannot be a polynomial time reduction between the two problems since the adherence problem is intractable even for linear trace patterns, for which the violation problem is tractable.

Lemma 1. *The linear adherence problem is PSpace-hard if the policy is given by a non-deterministic finite automaton or a regular expression.*

Proof. 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 non-deterministic finite automaton) is the universal language Σ^* or not is PSpace-complete [10]. Consequently, the adherence problem is PSpace-hard even for linear trace patterns.   □

Obviously, this PSpace lower bound then also holds for the general adherence problem. Next, we show that a matching PSpace upper bound holds for the general adherence problem, and thus for the linear one as well.

Lemma 2. *The adherence problem is in PSpace if the policy is given by a non-deterministic finite automaton or a regular expression.*

Proof. Since PSpace is a deterministic complexity class, it is sufficient to show that the complement of the adherence problem is in PSpace. Thus, given the trace pattern u of length n and the policy L, we want to decide whether $u \models L$ does not hold. As noted above, this is the same as deciding whether $u \lesssim \Sigma^* \setminus L$. Basically, we will use the PSpace decision procedure for the violation problem described in the proof of Theorem 2 to decide this problem. However, we cannot explicitly construct the automaton for $\Sigma^* \setminus L$ from the one for L since the size of this automaton might be exponential in the size of the automaton (or regular expression) for L. Instead, we construct the relevant parts of this automaton on-the-fly.

Let $\mathcal{A}$ be a non-deterministic finite automaton accepting L that has k states, and $\mathcal{B}$ the deterministic automaton for $\Sigma^* \setminus L$ constructed in the usual way from $\mathcal{A}$, i.e., the states of $\mathcal{B}$ are all the subsets of the set of states of $\mathcal{A}$, the initial state of $\mathcal{B}$ is the set of initial states of $\mathcal{A}$, the final states of $\mathcal{B}$ are the sets not containing any final state of $\mathcal{A}$, and $P \rightarrow_{\mathcal{B}}^a Q$ iff $Q = \{q \mid \exists p \in P \colon p \rightarrow_{\mathcal{A}}^a q\}$. Although the size of $\mathcal{B}$ is exponential in the size of $\mathcal{A}$, every single state of $\mathcal{B}$ can be represented using linear space. Also, deciding whether a given state of $\mathcal{B}$ is the initial state or a final state requires only polynomial space, and the same is true for constructing, for a given state P of $\mathcal{B}$ and $a \in \Sigma$, the unique state Q such that $P \rightarrow_{\mathcal{B}}^a Q$.

For $i \in \{1, \ldots, n\}$, we again 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., $P_X = \{i \mid 1 \le i \le n \wedge u_i = X\}$. Then $u \lesssim \Sigma^* \setminus L$ iff there is a sequence $Q_0, \ldots, Q_n$ of states of $\mathcal{B}$ such that the following conditions are satisfied:

1. Q_0 is the initial state and Q_n is a final state of $\mathcal{B}$;
2. for every $i \in \{1, \ldots, n\}$, if $u_i \in \Sigma$, then $Q_{i-1} \rightarrow_{\mathcal{B}}^{u_i} Q_i$;
3. for every variable X occurring in u, we have $\bigcap_{i \in P_X} L_{Q_{i-1}, Q_i} \ne \emptyset$.

Obviously, this characterisation yields the desired PSpace decision procedure for $u \lesssim \Sigma^* \setminus L$ if we can show that, for each variable X, the non-emptiness of $\bigcap_{i \in P_X} L_{Q_{i-1}, Q_i}$ can be decided by an NPSpace procedure.

Let $P_X = \{i_1, \ldots, i_m\}$, and $I_j := Q_{i_j-1}, F_j := Q_{i_j}$ for $j = 1, \ldots, m$. Note that $m \le n$ where n is the length of the pattern u. To check $\bigcap_{1 \le j \le m} L_{I_j, F_j} \ne \emptyset$, we proceed as follows.

1. Start with the m-tuple $(T_1, \ldots, T_m)$ where $T_j := I_j$ for $j = 1, \ldots, m$.
2. Check whether $(T_1, \ldots, T_m) = (F_1, \ldots, F_m)$. If this is the case, then terminate successfully, i.e., with the result that the intersection $\bigcap_{1 \le j \le m} L_{I_j, F_j}$ is non-empty. Otherwise, continue with 3.

3. Guess a letter $a \in \Sigma$, and replace $(T_1, \ldots, T_m)$ by the corresponding tuple of successor states in $\mathcal{B}$, i.e., make the assignment $T_j := \{q \mid \exists p \in T_j \colon p \to_{\mathcal{A}}^a q\}$. Continue with 2.

Obviously, if this procedure terminates successfully, then $\bigcap_{1 \leq j \leq m} L_{I_j, F_j}$ is indeed non-empty. In addition, if the intersection is non-empty, then there is a way of guessing letters such that the procedure terminates successfully. However, as described until now, the procedure does not terminate if the intersection is empty. It is, however, easy to see that it is enough to guess a sequence of letters of length at most $2^{k \cdot m}$, which is the number of different tuples $(T_1, \ldots, T_m)$ to be encountered during a run of the procedure.[4] In fact, if a tuple is reached more than once in a run of our procedure, then we can cut out this cycle to get a shorter run that achieves the same. Consequently, one can stop each run after $2^{k \cdot m}$ iterations. This can be realized using a binary counter that requires $k \cdot m$ bits, i.e., space polynomial in the size of the input.   □

The following theorem is an immediate consequence of the two lemmas that we have just shown.

Theorem 5. *Both the general and the linear adherence problem are PSpace-complete if the policy is given by a non-deterministic finite automaton or a regular expression.*

6   Policies defined by LTL formulae

In many applications, linear temporal logic (LTL) [18] is used to specify the (wanted or unwanted) behaviour of a system [15]. LTL is usually interpreted in temporal structures with infinitely many time points, but variants where temporal structures are finite sequences of time points have also been considered in the literature [6, 9]. Here, we consider the setting employed in [6], where an LTL formula φ defines a regular language of finite words, which we denote by L_φ in the following. It is well-known that not all regular languages can be defined by LTL formulae: the class of languages definable by LTL formulae is precisely the class of star-free languages, which is a strict subclass of the class of all regular languages [6]. Given an LTL formula φ, one can construct a finite non-deterministic automaton $\mathcal{A}_\varphi$ that accepts L_φ (by adapting the Vardi-Wolper construction [23] to the finite case). Although the size of this automaton is exponential in the size of φ, it satisfies properties similar to the ones mentioned for the automaton $\mathcal{B}$ in the proof of Lemma 2: every single state of $\mathcal{A}_\varphi$ can be represented using linear space, deciding whether a given state of $\mathcal{A}_\varphi$ is an initial state or a final state requires only polynomial space, and the same is true for guessing, for a given state p of $\mathcal{A}_\varphi$ and $a \in \Sigma$, a state q such that $p \to_{\mathcal{A}_\varphi}^a q$. We will also use that, just as in the infinite case, the satisfiability problem for LTL over finite temporal

---

[4] Recall that k is the number of states of $\mathcal{A}$, and $m \leq n$ where n is the length of the input pattern u.

structures (i.e., for a given LTL formula φ, decide whether L_φ is empty or not) is PSpace-complete (this can be shown by a proof identical to the one in [22] for the infinite case).

In this section, we consider the complexity of the violation (adherence) problem for the case where the policy L_φ is given by an LTL formula φ. First, note that the adherence and the violation problem can be reduced to each other in linear time since LTL allows for negation:

$$u \lesssim L_\varphi \ \text{ iff } \ u \not\models L_{\neg\varphi} \quad \text{ and } \quad u \models L_\varphi \ \text{ iff } \ u \not\lesssim L_{\neg\varphi}.$$

Theorem 6. *Both the general and the linear violation (adherence) problem are PSpace-complete if the policy is given by an LTL formula.*

Proof. It is sufficient to show PSpace-completeness for the violation problem.

PSpace-hardness of the violation problem can be shown by a reduction of the satisfiability problem: the LTL formula φ is satisfiable iff $X \lesssim L_\varphi$.

Membership in PSpace can be shown just as in the proof of Lemma 2. We apply the PSpace decision procedure described in the proof of Theorem 2 to the automaton $\mathcal{A}_\varphi$, but without constructing this automaton explicitly. Instead, its relevant parts are constructed on the fly. Again, the main fact to be shown is that the induced intersection emptiness problems for the variable can be decided within PSpace. This can be done just as in the proof of Lemma 2. The only difference is that the automaton $\mathcal{A}_\varphi$ is non-deterministic whereas the automaton $\mathcal{B}$ considered in the proof of Lemma 2 was deterministic. However, this just means that, instead of constructing the unique tuple of successors states in Step 3, we guess such a successor tuple.   □

Let us now consider the case where the policy (i.e., the LTL formula) is assumed to be fixed. This means that the size of the automaton $\mathcal{A}_\varphi$ is a constant. For the case of linear patterns, we can then decide the violation problem (and thus also the adherence problem) in linear time (see the proof of the second part of Theorem 1).

Theorem 7. *Assume that the policy is fixed and given by an LTL formula. Then, the linear violation problem and the linear adherence problem can be solved in time linear in the length of the input trace pattern.*

For non-linear trace patterns, the violation problem for fixed LTL policies has the same complexity as in the case of fixed regular policies. The results for the adherence problem then follow from the above reductions between the violation problem and the (complement of the) adherence problem. The proof of the following theorem is identical to the one of Theorem 3.

Theorem 8. *Assume that the policy is fixed and given by an LTL formula. Then the violation problem is in NP and the adherence problem is in coNP.*

In order to show NP-hardness of the violation problem for a fixed policy φ (which then implies coNP-hardness of the adherence problem for the fixed

policy $\neg\varphi$), it is enough to show that the fixed policy $(T\wedge)^*T$ used in the proof of Theorem 4 is star-free [6]. Since the star-free languages are closed under complement, it is enough to show that the complement of this language is star-free. By definition, finite languages (and thus also T) are star-free. In addition, star-free languages are closed under all Boolean operations and concatenation. It is also well-known (and easy to see) that Σ^* as well as $(\Sigma \setminus \{a\})^*$ for any $a \in \Sigma$ are star-free [16]. The language $\Sigma^* \setminus (T\wedge)^*T$ is the union of the following star-free languages:

- all words not containing $\wedge$ and not belonging to T: $(\Sigma^* \setminus T) \cap (\Sigma \setminus \{\wedge\})^*$.
- all words not starting with an element of T before the first $\wedge$: $((\Sigma^* \setminus T) \cap (\Sigma \setminus \{\wedge\})^*)\wedge\Sigma^*$.
- all words not having a word of T between two consecutive occurrences of $\wedge$: $\Sigma^*\wedge((\Sigma^* \setminus T) \cap (\Sigma \setminus \{\wedge\})^*)\wedge\Sigma^*$.
- all words not ending with an element of T after the last $\wedge$: $\Sigma^*\wedge((\Sigma^* \setminus T) \cap (\Sigma \setminus \{\wedge\})^*)$.

This shows that $(T\wedge)^*T$ is star-free and thus can be expressed by an LTL formula φ. Thus, the proof of Theorem 4 yields the following results.

Theorem 9. *There exists a fixed policy given by an LTL formula such that the violation problem (adherence problem) for this policy is NP-hard (coNP-hard).*

7   Future work

One of the main tasks to be addressed in our future work is to investigate which kinds of unwanted behaviour in online trading systems one can formally describe using regular languages and LTL formulae. We also intend to consider the problem of debugging policies as another application for our approach of matching trace patterns with regular policies. The violation and the adherence problem can be viewed as instances of the *policy querying* problem, which has been introduced in [12] as a tool for the analysis of access control policies. The main idea is that, while it may be quite hard for inexperienced users to correctly define a policy using a regular expression or an LTL formula, it should be easier to describe, using trace patterns, types of traces they want to allow or disallow. Checking for violation/adherence can then be used to find potential errors in the definition of the policy.

References

1. R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification language. In *Proc. TACAS'02*, Springer LNCS 2280, 2002.
2. A. Bauer, M. Leucker, and Ch. Schallhart. Monitoring of real-time properties. In *Proc. FSTTCS'06*, Springer LNCS 4337, 2006.

3. A. Bauer, M. Leucker, and J. Streit. SALT—Structured Assertion Language for Temporal logic. In *Proc. ICFEM'06*, Springer LNCS 4260, 2006.
4. S. Ben-David, D. Fisman, and S. Ruah. Embedding finite automata within regular expressions. *Theoretical Computer Science*, 404:202–218, 2008.
5. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
6. J. Cohen, D. Perrin, and J.-E. Pin. On the expressive power of temporal logic. *J. Comput. System Sci*, 46:271–294, 1993.
7. S. Colin and L. Mariani. Run-time verification. In *Model-Based Testing of Reactive Systems*, Springer LNCS 3472, 2004.
8. C. Eisner and D. Fisman. *A Practical Introduction to PSL (Series on Integrated Circuits and Systems)*. Springer, 2006.
9. C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. van Campenhout. Reasoning with temporal logic on truncated paths, In *Proc. CAV'03*, Springer LNCS 2725, 2003.
10. M. R. Garey and D. S. Johnson. *Computers and Intractability — A guide to NP-completeness*. W. H. Freeman and Company, San Francisco (CA, USA), 1979.
11. K. Havelund and G. Rosu. Synthesizing monitors for safety properties, In *Proc. TACAS'02*. Springer LNCS 2280, 2002.
12. C. Kirchner, H. Kirchner, and A. Santana de Oliveira. Analysis of rewrite-based access control policies. In *Proc. 3rd International Workshop on Security and Rewriting Techniques*, 2008.
13. D. Kozen. Lower bounds for natural proof systems. In *Proc. FOCS'77*. IEEE Computer Society, 1977.
14. K. Krukow, M. Nielsen, and V. Sassone. A framework for concrete reputation-systems with applications to history-based access control. In *Proc. ACM Conference on Computer and Communications Security*. ACM, 2005.
15. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.
16. D. Perrin. *Finite Automata*. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*. Elsevier, Amsterdam, 1990.
17. W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *Proc. FOCS'99*. IEEE Computer Society, 1999.
18. A. Pnueli. The temporal logic of programs. In *Proc. FOCS'77*. IEEE Computer Society, 1977.
19. W. J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. of Computer and System Sciences*, 4:177–192, 1970.
20. F. B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.
21. K. U. Schulz. Makanin's algorithm for word equations - two improvements and a generalization. In *Proc. IWWERT'90*, Springer LNCS 572, 1990.
22. A. Prasad Sistla and E. C. Clarke. The complexity of propositional linear temporal logic. *J. of the ACM*, 32(3):733–749, 1985.
23. M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. In *Proc. STOC'84*, 1984.