Prompt Runtime Enforcement

No Author Given

Abstract. This paper deals with the problem of runtime enforcement (RE) in the context of reactive systems, which consists in modifying the outputs of a system minimally to ensure its correctness. In contrast to enforcers that can postpone events via buffering, enforcers for reactive systems must operate within the same reactive cycle, always yielding a (possibly modified) output. For safety properties, an enforcer makes sure to satisfy the property at each step. However, for general regular properties, one can only expect to satisfy the property eventually. There is then a risk that even under enforcement the satisfaction is indefinitely delayed, and the property is never actually satisfied. Forcing the satisfaction of regular or ω-regular properties has been considered using bounded fairness and prompt eventuality. In this paper, we propose a new runtime enforcement framework for regular properties with prompt eventualities. Given an automaton specifying a property φ and a bound k, the enforcer should never falsify φ more than k consecutive steps. We formally define this RE problem, characterize k-enforceability of automata, and exhibit the construction of the enforcer. Rather than fixing k, we also study whether k can be computed to ensure k-enforceability or some maximal coverage of φ. We implement the k-prompt enforcement framework, and demonstrate its behaviour with varying k.

1 Introduction

Runtime Enforcement (RE) techniques complement approaches such as model checking, with the goal to monitor the system and enforce a small set of crucial properties of the system at runtime. Such runtime techniques are essential as some properties may be difficult to prove due to scalability issues, and in cases when all the properties are proved statically, the system may still malfunction due to environmental changes or due to security concerns (attacks at runtime). Formal RE techniques focus on construction of enforcers automatically from formal specification of the properties to be enforced. The constructed enforcer attached to the system monitors inputs/outputs of the system and corrects erroneous input/output values instantaneously. In addition to ensuring correctness, an enforcer should also guarantee that correct behaviour is unmodified.

Runtime enforcement has been an active area of research since its inception by F.B. Schneider in 2000 [13] in the context of security policies. Different actions like *suppress, store, delay, and reorder* [7,4,12,6,9] may be used to modify the system behaviour. Enforcement in the domain of cyber-physical systems (CPSs) is considerably different. CPSs, in general, are reactive systems which incessantly interact with their environments. This prohibits us to use actions like suppress

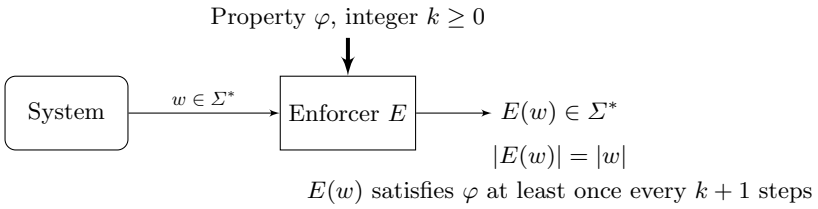


Fig. 1: General scheme of prompt enforcement.

or delay that interrupt the continuous execution of the system. Thus, approaches suitable for reactive systems such as [11] are proposed where the runtime enforcer continuously monitors the system, and instantaneously edits only those events which will lead to the violation of the property being enforced. The approach in [11] is restricted to enforcing safety (prefix closed) properties and in [10] it is extended to regular discrete-timed properties for reactive systems.

In all the enforcement frameworks such as [10] which considers enforcement of regular properties for reactive systems, the notion of soundness that is defined ensures that whatever is released as output by the enforcer can be further extended so as to satisfy the property. That is, the enforcer ensures that it is always possible to extend the current execution to satisfy the property, but at the same time, it does not guarantee that the property will actually be satisfied eventually. Thus for some properties and observations, the output of the enforcer may continuously be a prefix of an accepting word without ever actually satisfying the property.

However, in some contexts it is important to bound the time before an eventuality is satisfied. To illustrate this, consider an autonomous driving application, where we want to enforce that the vehicle is moving within the speed limit. Let us consider that there are 3 ranges of speed (1) less than 80 (safe range), (2) between 80 to 90 (unsafe but not a violation), and (3) above 90 (violation). Here, we want the vehicle to never go beyond 90 but also never to remain between 80 and 90 indefinitely. Indeed, a good enforcer must eventually bring the vehicle to the safe range, rather than always leaving the possibility to do so without actually enforcing it.

This problem of indefinitely delaying the satisfaction of eventualities has already been considered in the literature. This has led to the study of prompt semantics of logics or Büchi automata, where these undesired infinite behaviours are withdrawn (see *e.g.*, [3,1]). In this paper, inspired by these works on prompt semantics, we consider the problem of satisfying eventualities in the context of enforcement. When enforcing a regular property, the expected behaviour of the runtime enforcer is still to produce prefixes of accepting words, but also to guarantee that the property is accepted eventually with regular intervals.

Contributions: We introduce the *prompt runtime enforcement* problem for regular properties, defined as follows and illustrated in Figure 1. Given a regular property defined by a finite automaton, and an integer k, we build a k-prompt

enforcer for φ, where a *k-prompt enforcer* transforms an input stream of actions of the system $w \in \Sigma^*$, (where Σ denotes the alphabet of actions) into an output stream of actions $E(w) \in \Sigma^*$ such that the $E(w)$ is a prefix of a word satisfying φ, and the property is satisfied at least once every $k+1$ steps. Moreover, the enforcer should be instantaneous (exactly one output action is produced for every input action), and should alter the actual input only when necessary.

After formalizing the problem, we characterize k-enforceability (existence of a k-prompt enforcer) in terms of a property of the automaton representing φ, and give polynomial-time algorithms to compute k-prompt enforcers. By fixing k, we fix an upper bound on the number of steps within which the property must be satisfied at least once, for every word generated by the enforcer. However, we may sometimes desire that this bound varies for different words generated by the enforcer. We show how *prompt enforcers* solve this problem, by modelling the problem as two-player game with Büchi objective. We show how prompt enforcers can be assumed to be k-prompt enforcer by choosing k to be the size of the state space of the game. We further show how to compute automatically the minimum k such that a given regular property φ is k-enforceable, thus ensuring that any input stream can satisfy φ at least once every $k+1$ steps. There is however a compromise to make between the promptness, that is, the value of k, and the permissiveness of the enforcer: while a small k ensures that the property is satisfied more often, an enforcer with a larger value of k can be less restrictive and changes the input less often. Unfortunately, for a given property, there is in general no upper bound (permissiveness can always be increased in the presence of loops with no accepting state). Nevertheless, when no such loops exist an upper bound can be computed, and we provide a procedure for computing this maximal k in this case. We implemented the k-prompt enforcer using Python. We test our enforcer with a number of policies and show how the promptness and permissiveness change with varying k.

2 Preliminaries

Let Σ be a finite alphabet of actions. A *finite word* on Σ is an element $w = \sigma_1 \cdot \sigma_2 \cdot \ldots \cdot \sigma_n$ of Σ^* whose length is $|w| = n$, where "$\cdot$" is the usual concatenation of words. An *infinite word* is an element $w = \sigma_1 \cdot \sigma_2 \cdot \ldots$ of Σ^ω; by convention $|w| = +\infty$. The empty word is denoted ε. A language of finite (respectively infinite) words is a subset of Σ^* (resp. Σ^ω). We write $w \preceq w'$ when w is a prefix of w', *i.e.*, there exists w'' such that $w \cdot w'' = w'$. Let $w_{\leq i}$ denote the prefix of w of size i if $|w| \geq i$. We denote by $\mathsf{pref}(w)$ the language of prefixes of w, and for a language L of finite or infinite words, $\mathsf{pref}(L)$ is the language of finite prefixes of words in L. In the remainder, we consider enforcement of finite words, but it will be convenient to reason about infinite words to talk about their unknown and unbounded continuations.

Definition 1. *A deterministic automaton is a tuple $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \delta, Q_F)$, where Q is a set of states, $q_{\mathsf{init}} \in Q$ the initial state, $Q_F \subseteq Q$ a set of accepting states, Σ is an alphabet of actions, and $\delta : Q \times \Sigma \to Q$ a transition function.*

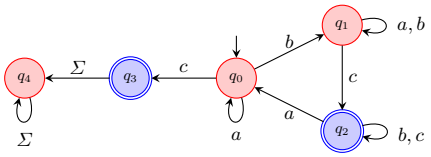


Fig. 2: An automaton $\mathcal{A}$.

A *run* of an automaton $\mathcal{A}$ on a given word $w = \sigma_1 \cdot \sigma_2 \cdot \ldots \cdot \sigma_{n-1}$ is a sequence $q_1 q_2 \ldots q_n$ of states such that $q_1 = q_{\mathsf{init}}$, and for all $1 \leq i \leq n-1$, $q_{i+1} = \delta(q_i, \sigma_i)$. The run is accepting if $q_n \in Q_F$; rejecting otherwise. A word is accepted if one of its runs is accepting. Let $\mathcal{L}(\mathcal{A})$ denote the set of finite words accepted by $\mathcal{A}$ and $\mathcal{L}_\omega(\mathcal{A})$ the set of infinite words accepted by $\mathcal{A}$ by the Büchi acceptance condition (*i.e.*, the set of infinite words w such that $\mathsf{Inf}(w) \cap Q_F \neq \emptyset$, where $\mathsf{Inf}(w)$ is the set of states appearing infinitely often in the run of $\mathcal{A}$ on w).

We will implicitly consider *complete automata*, *i.e.*, for any state q and any action σ, if transition $\delta(q, a)$ is missing, it is directed to a non accepting trap state. This does not change its language.

Example 1. Figure 2 represents an automaton $\mathcal{A}$. The nodes are the states, and the vertices represent transitions from states, labelled with actions. The set of states is $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $q_{\mathsf{init}} = q_0$ is the initial state, $Q_F = \{q_2, q_3\}$ is the set of accepting states, and $\Sigma = \{a, b, c\}$ is the alphabet. $\rho = q_0 q_1 q_2 q_2 q_0 q_3$ is the accepting run of $\mathcal{A}$ on the word $w = bccac$.

Given a state $q \in Q$, we define $\mathsf{Post}_{\mathcal{A}}(q) = \{q' \in Q \mid \exists \sigma \in \Sigma, q' = \delta(q, \sigma)\}$; and $\mathsf{Pre}_{\mathcal{A}}(q) = \{q' \in Q \mid \exists \sigma \in \Sigma, q = \delta(q', \sigma)\}$ respectively the successors and predecessors of q in one step. We extend Pre and Post to sets of states. Moreover, for $X \subseteq Q$ and $0 \leq k \leq l$, we write $\mathsf{Pre}^{[k,l]}_{\mathcal{A}}(X)$ for $\bigcup_{i \in [k,l]} \mathsf{Pre}^i_{\mathcal{A}}(X)$, *i.e.* the set of states from which a state in X is reachable in a number of steps comprised between k and l (where $\mathsf{Pre}^i_{\mathcal{A}}(X)$ is the composition of i-many $\mathsf{Pre}_{\mathcal{A}}$ on X, with the convention that $\mathsf{Pre}^0_{\mathcal{A}}(X) = X$).

We define the *surplus* of a finite word w w.r.t. $\mathcal{A}$, written $\mathsf{surplus}_{\mathcal{A}}(w)$, as the size of its minimal suffix w'' such that when writing $w = w' \cdot w''$, $w' \in \mathcal{L}(\mathcal{A})$; and we define $\mathsf{surplus}_{\mathcal{A}}(w) = |w| + 1$ when no such suffix exists. Intuitively, the surplus of a word is the number of actions after the last accepting prefix if there is one, and $|w| + 1$ otherwise. In other words, it is the number of non-accepting states traversed since the last accepting one (initially including q_{init} if q_{init} is not accepting). We have $\mathsf{surplus}_{\mathcal{A}}(w) = 0$ if and only if $w \in \mathcal{L}(\mathcal{A})$. In particular, $\mathsf{surplus}_{\mathcal{A}}(\varepsilon) = 0$ if $\varepsilon \in \mathcal{L}(\mathcal{A})$, and 1 otherwise.

We then come to the central notion of promptness. The *promptness* of a finite or infinite word w w.r.t an automaton $\mathcal{A}$ specifying φ, is the maximal surplus of its prefixes: $\mathsf{promptness}_{\mathcal{A}}(w) = \sup_{w' \in \mathsf{pref}(w)} \mathsf{surplus}_{\mathcal{A}}(w')$. This means, w falsifies φ at no more than $\mathsf{promptness}_{\mathcal{A}}(w)$-many consecutive steps. Note that, $\mathsf{promptness}_{\mathcal{A}}(w)$ is non-decreasing when w is extended with a suffix. The promptness of a language $L \subseteq \Sigma^\omega$ w.r.t $\mathcal{A}$ is the supremum of the promptness

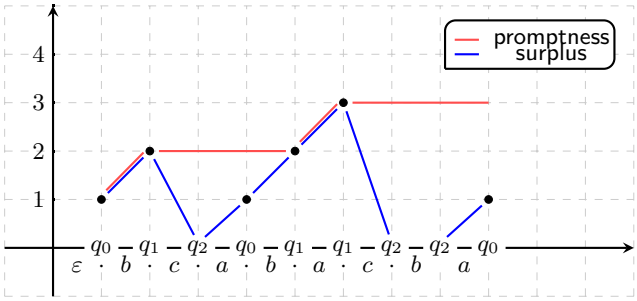


Fig. 3: surplus and promptness of the prefixes of the word *bcabacba* with respect to the automaton $\mathcal{A}$ of Figure 2.

of its words: $\mathsf{promptness}_{\mathcal{A}}(L) = \sup\{\mathsf{promptness}_{\mathcal{A}}(w) \mid w \in L\}$. We will not use the subscript $\mathcal{A}$ from the notations, when it is clear from the context.

By extension, the promptness of a (finite or infinite) run is the minimal number k such that accepting states are seen at least every $k+1$ steps in this run. Formally, for a run $\rho = q_1 q_2 \dots q_n$ of $\mathcal{A}$ for all $1 \leq i \leq n-k$, $\{q_i, \dots, q_{i+k}\} \cap Q_F \neq \emptyset$. The promptness of a word is then the promptness of its run in $\mathcal{A}$.

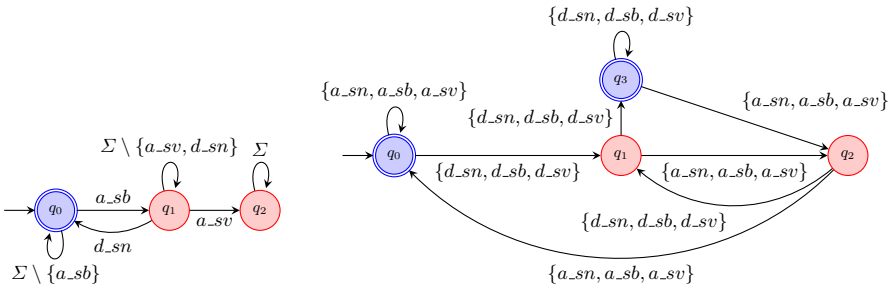
Example 2. Figure 3 illustrates graphically these concepts for the word *bcabacba* using automaton $\mathcal{A}$ of Figure 2. The x-axis represents the consecutive actions and current state of the automaton, and the y-axis depicts the evolution of surplus and promptness. Note that $\mathsf{surplus}(\varepsilon) = 1$ since q_0 is not accepting, it then increases upon b, *i.e.* $\mathsf{surplus}(b) = 2$; then $\mathsf{surplus}(bc) = 0$ since bc is accepting. Later upon the next 3 actions aba it again increments by 1 upon each action, $\mathsf{surplus}(bcaba) = 3$, and it again becomes 0 upon the next action c as $bcabac$ is accepting $\mathsf{surplus}(bcabac) = 0$. Initially $\mathsf{promptness}(w)$ is 1, increments to 2 upon the first action b, then remains 2 for the next three actions and later increases to 3 upon the next action (when the word is $bcaba$, since $\mathsf{surplus}(bcaba) = 3$) and then stays equal to 3. Note that, there are words (*e.g.* a^*) for which both surplus and promptness are unbounded here.

We define the *k-prompt sub-languages* $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}_\omega(\mathcal{A})$, respectively, over finite and infinite words, as follows:

$$\mathcal{L}^k(\mathcal{A}) = \{w \in \mathcal{L}(\mathcal{A}) \mid \mathsf{promptness}_{\mathcal{A}}(w) \leq k\},$$
$$\mathcal{L}^k_\omega(\mathcal{A}) = \{w \in \mathcal{L}_\omega(\mathcal{A}) \mid \mathsf{promptness}_{\mathcal{A}}(w) \leq k\},$$

and let $\mathcal{L}^{<\infty}(\mathcal{A}) = \cup_{k \geq 0} \mathcal{L}^k(\mathcal{A})$ and $\mathcal{L}^{<\infty}_\omega(\mathcal{A}) = \cup_{k \geq 0} \mathcal{L}^k_\omega(\mathcal{A})$ denote the *unbounded prompt languages* respectively.

Here using non-strict inequality means that surpluses of prefixes are of size bounded by k, thus an accepting prefix will be seen at least once every subword of length $k+1$. In other terms, for any run of a word in these languages, an accepting state is seen at least once every $k+1$ steps.



(a) φ_1 expressed as $\mathcal{A}_{\varphi_1}$ (b) φ_2 expressed as $\mathcal{A}_{\varphi_2}$

Fig. 4: Properties of autonomous vehicle system.

Notice that, $\mathcal{L}^k(\mathcal{A})$ is not necessarily included in $\mathsf{pref}(\mathcal{L}^k_\omega(\mathcal{A}))$. Indeed a word in $\mathcal{L}^k_\omega(\mathcal{A})$ requires infinite repetition of both accepting states and promptness constraint; being in $\mathsf{pref}(\mathcal{L}^k_\omega(\mathcal{A}))$ means that a continuation in $\mathcal{L}^k_\omega(\mathcal{A})$ exists; while a word in $\mathcal{L}^k(\mathcal{A})$ may have no infinite continuation with promptness bounded by k, and even no infinite continuation at all. Notice that $\mathcal{L}^k_\omega(\mathcal{A})$ is the subset of infinite words in $\mathcal{L}_\omega(\mathcal{A})$ whose set of prefixes in $\mathcal{L}^k(\mathcal{A})$ is infinite.

Example 3. To illustrate this, consider the language of the automaton $\mathcal{A}$ described by Figure 2 and let $k = 2$. The empty word ε is not accepted, thus has surplus and promptness 1. Then the word c is accepted, has surplus 0 and promptness 1, thus we have $c \in \mathcal{L}^1(\mathcal{A}) \subseteq \mathcal{L}^2(\mathcal{A})$. However, no extension of c is accepted, thus $c \notin \mathsf{pref}(\mathcal{L}^2_\omega(\mathcal{A}))$. On the other hand, bc has surplus 0, promptness 2 and is both in $\mathcal{L}^2(\mathcal{A})$ and $\mathsf{pref}(\mathcal{L}^2_\omega(\mathcal{A}))$ since $bca^\omega \in \mathcal{L}^2_\omega(\mathcal{A})$.

Promptness of safety properties. Safety properties (*i.e.*, prefix-closed languages) have promptness equal to 0: indeed prefixes of safe words have surplus 0 (otherwise said, safe runs visit accepting states at every 1 step). This entails that for any safety deterministic automaton $\mathcal{A}$, both $\mathcal{L}^0(\mathcal{A}) = \mathcal{L}(\mathcal{A})$ and $\mathcal{L}^0_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A})$.

3 Motivation

Enforcement of properties in a system for general regular properties is based on the idea that absolute violation of the property must be averted. When considering the existing enforcement frameworks such as [10], enforcers do not allow the system to deviate from the possibility of satisfying the property that is being enforced. In other words, they allow the system to exist in a state from which property satisfaction can be reached. The enforcers assure property satisfiability, but are incapable of exactly determining when the said property will be satisfied.

Consider an autonomous vehicle system that has an output channel, and let $\Sigma = \{a\_sn, d\_sn, a\_sb, d\_sb, a\_sv, d\_sv\}$ be the alphabet of actions. Action prefix 'a' in $\{a\_sn, a\_sb, a\_sv\}$ denote that the vehicle is accelerating, while prefix 'd'

denotes that the vehicle is decelerating. The suffix of the actions denotes whether the current speed of the vehicle is in *normal range (sn), buffer range (sb), or violation range (sv)*.

Let x be some speed-limit on a motorway, and v be the speed of the vehicle. The system outputs: a_sn or d_sn when $v \leq x$, a_sb or d_sb when $x < v \leq x + \Delta$ and a_sv or d_sv when $v > x + \Delta$, where Δ is some marginal buffer speed. Let φ_1 denote the property *"the vehicle should not accelerate to speed violation, and it should often remain within the speed-limit"*. Property φ_1 can be specified as an automaton $\mathcal{A}_{\varphi_1}$ shown in Figure 4a. However a typical enforcer for regular properties obtained using frameworks such as [10], would still allow the system to forever remain in non-accepting state q_1, because there exists a path from q_1 to an accepting state q_0, thereby providing uncertainty over returning to or below the speed limit ($\leq x$).

In this work, we thus propose a prompt runtime enforcement framework for synthesis of a *k-prompt enforcer*, which for a given value k, will guarantee that state q_0 is reached at least once every $k+1$ steps, thereby guaranteeing that the system will return to or is below the speed limit.

Consider another property φ_2 expressed as $\mathcal{A}_{\varphi_2}$ illustrated in Figure 4b. Property φ_2 differentiates the 'non-jerk' $\{q_0, q_3\}$ states of the system from 'jerk' $\{q_1, q_2\}$ state. A 'jerk' state is identified by the sequence $(a\_..) \cdot (d\_..) \cdot (a\_..)$ or $(d\_..) \cdot (a\_..) \cdot (d\_..)$[1]. Using the parameter k in *k-prompt enforcer*, we can decide how many consecutive 'jerks' are allowed in the system. For example $k = 3$ will not allow more than 3 consecutive jerks, and for $k = 1$ the enforcer will not allow two or more consecutive jerks. Thus, the *k-prompt enforcer* allows to control the permissible number of consecutive jerks in the system.

4 Problem Definition

We want to define an enforcer that can change actions, but cannot remove them, and which ensures that a word belonging to the language $\mathcal{L}(\mathcal{A})$ will be produced regularly. We will consider two variants of this problem: either the designer chooses a parameter k and we look for an enforcer that ensures that the produced word satisfies $\mathcal{L}(\mathcal{A})$ every $k+1$ steps, or it is required that for each infinite word produced by the enforcer (assuming it is run over an infinite duration) there exists some k such that a prefix belonging to $\mathcal{L}(\mathcal{A})$ is seen every $k+1$ steps. Note the subtlety here: in the latter case, for each infinite word, a different bound k may apply (similar to the prompt-Büchi acceptance condition in [1]).

We call *word transducer* a function $E : \Sigma^* \to \Sigma^*$ that is monotonic for the prefix relation, that is, $w \preceq w' \Rightarrow E(w) \preceq E(w')$. A word transducer lifted to infinite words, is defined as: for $w \in \Sigma^\omega$, $E(w) = \lim_{i \geq 1} E(w_{\leq i})$. This limit is well defined due to the monotonicity of E. We say that a word transducer is

[1] $a\_..$ denotes any action where the vehicle is accelerating, *i.e.*, $a\_.. = a\_sn/a\_sb/a\_sv$. Similarly, $d\_..$ denotes any action where the vehicle is decelerating, $d\_.. = d\_sn, d\_sb, d\_sv$.

synchronous when $\forall w \in \Sigma^*$, $|E(w)| = |w|$. All together, it entails that a synchronous enforcer is *instantaneous*: at each step, a given action σ is transformed into exactly one action σ'.

Definition 2. *Given an automaton $\mathcal{A}$ and an integer bound k, a k-*prompt enforcer *E for $\mathcal{A}$ is a synchronous word transducer $E : \Sigma^* \to \Sigma^*$ satisfying the following properties:*

Soundness: $\forall w \in \Sigma^\omega,\ E(w) \in \mathcal{L}_\omega^k(\mathcal{A})$.
Minimal Intervention: $\forall w \in \Sigma^*,\ \forall \sigma \in \Sigma,\ E(w)\cdot\sigma \in \mathsf{pref}(\mathcal{L}_\omega^k(\mathcal{A}))$ *implies* $E(w \cdot \sigma) = E(w) \cdot \sigma$.

We say that a language represented by $\mathcal{A}$ is *k-enforceable* if there exists a k-prompt enforcer for $\mathcal{A}$.

Soundness is defined for infinite words, even though the enforcer works on finite words. Intuitively, soundness means that the enforcer should always keep the possibility of an infinite continuation with promptness bounded by k, and can equivalently be defined as $\forall w \in \Sigma^*, E(w) \in \mathsf{pref}(\mathcal{L}_\omega^k(\mathcal{A}))$. Minimal intervention means that if some infinite extension of $E(w) \cdot \sigma$ with promptness k exists, σ should not be modified by the enforcer.

In [10], the minimal intervention property is called *transparency*. The *monotonicity* criterion of [10] is given here by the fact that E is a word transducer; and *instantaneity* of [10] follows from synchronicity of the word transducer.

Notice that by definition of a k-prompt enforcer for $\mathcal{A}$, the finite language it produces is exactly $\mathsf{pref}(\mathcal{L}_\omega^k(\mathcal{A}))$. Indeed by soundness any enforced word is in $\mathsf{pref}(\mathcal{L}_\omega^k(\mathcal{A}))$. Conversely, by minimal intervention, any input word in $\mathsf{pref}(\mathcal{L}_\omega^k(\mathcal{A}))$ is unmodified, thus can be produced. Therefore, in the limit, a k-prompt enforcer produces exactly $\mathcal{L}_\omega^k(\mathcal{A})$.

The second type of enforcers we are interested in are enforcers that guarantee promptness without a fixed bound k:

Definition 3. *Given an automaton $\mathcal{A}$, a* prompt enforcer *E for $\mathcal{A}$ is a synchronous word transducer $E : \Sigma^* \to \Sigma^*$ satisfying*

Soundness: $\forall w \in \Sigma^\omega,\ E(w) \in \mathcal{L}_\omega^{<\infty}(\mathcal{A})$.

In this case, we do not require any minimal intervention property. In fact, because the promptness of each infinite word can be different, and cannot be deduced from a finite prefix, one cannot impose a minimal intervention property as in Definition 2.

We are interested in determining when such k-prompt enforcers or prompt enforcers exist, and how to build them. In the next section, we will study algorithms for this problem.

5 Algorithms for Prompt Runtime Enforcement

5.1 k-Prompt Enforcers

By Section 4, the language of a k-prompt enforcer is exactly $\mathcal{L}_\omega^k(\mathcal{A})$. In this section, we show that enforceability requires the non-emptiness of $\mathcal{L}_\omega^k(\mathcal{A})$; in-

terestingly, it turns out that this condition is also sufficient. Moreover, we also characterised enforceability by a property on automata and the proof is constructive: it shows how to build a k-prompt enforcer.

Before going into this result, we first define a few notations that will be used to formulate and prove this. Let $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \delta, Q_F)$ be a deterministic automaton, we use Z^k to denote the greatest fixpoint: $Z^k = \nu X.\, Q_F \cap \mathsf{Pre}_{\mathcal{A}}^{[1,k+1]}(X)$. Intuitively, Z^k is the largest set consisting of all the states in Q_F from which another state in Z^k is reachable in at most $k+1$ steps. We now define a distance function $d_{Z^k} : q \mapsto \mathbb{N}$ as $d_X(q) = \min\{\{+\infty\} \cup \{i \in \mathbb{N} \mid q \in \mathsf{Pre}_{\mathcal{A}}^{i}(X)\}\}$. In words, $d_{Z^k}(q)$ denotes the shortest distance of q from Z^k, when Z^k is non-empty and is reachable from q, and the distance is $+\infty$ otherwise. Note that, $d_{Z^k}(q) = 0, \forall q \in Z^k$.

Example 4. We illustrate these notions on the automaton $\mathcal{A}$ depicted in Figure 2. Suppose $k = 2$, the fixpoint computation of Z^2 starts with $X^{(0)} = Q_F = \{q_2, q_3\}$; then since $\mathsf{Pre}_{\mathcal{A}}^{[1,2]}(\{q_2, q_3\}) = \{q_2\}$, we get $X^{(1)} = \{q_2\}$ and the fixpoint is reached, thus $Z^2 = \{q_2\}$. Note that, even though q_3 is accepting, it does not belong to Z^k for any k, since from q_3 no other accepting state (hence no states in Z^k) is reachable. On the other hand, the distances of the states from Z^2 are: $d_{Z^2}(q_0) = 2$, $d_{Z^2}(q_1) = 1$, $d_{Z^2}(q_2) = 0$, and $d_{Z^2}(q_3) = d_{Z^2}(q_4) = +\infty$.

Note also that $Z^0 = Z^1 = \emptyset$, and $Z^k = \{q_2\}$ for $k \geq 2$.

For a finite word w, we define $sum_{Z^k}(w) = \mathsf{surplus}(w) + d_{Z^k}(q_w)$ where $q_w = \delta(q_{\mathsf{init}}, w)$. In words, $sum_{Z^k}(w)$ is the sum of the surplus of w and the shortest distance to the set Z^k after reading w. The intention is to define a k-prompt enforcer E for $\mathcal{A}$ that will maintain the following invariant:

$$\forall w \in \Sigma^*,\ sum_{Z^k}(E(w)) \leq k+1 \tag{1}$$

We then define, for $w \in \Sigma^*$, $\mathsf{filter}(w) = \{\sigma \in \Sigma \mid sum_{Z^k}(w \cdot \sigma) \leq k+1\}$, *i.e.*, the set of all actions σ, that when concatenated with w still satisfy the invariant (1).

We now characterize k-enforceability by the following theorem:

Theorem 4. *Given a deterministic automaton $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \delta, Q_F)$, and a bound $k \geq 0$, the following statements are equivalent:* (1) *A is k-enforceable,* (2) *$\mathcal{L}_\omega^k(\mathcal{A})$ is non-empty, and* (3) *$d_{Z^k}(q_{\mathsf{init}}) \leq k$.*

Proof.
$\boxed{1 \implies 2}$ We assume $\mathcal{A}$ is k-enforceable. Then there exists a k-prompt enforcer E for $\mathcal{A}$. For any infinite word $w \in \Sigma^\omega$, by soundness of E we get that $E(w) \in \mathcal{L}_\omega^k(\mathcal{A})$, thus $\mathcal{L}_\omega^k(\mathcal{A}) \neq \emptyset$.

$\boxed{2 \implies 3}$ Let $w \in \mathcal{L}_\omega^k(\mathcal{A})$ by non-vacuity of this set. The run of $\mathcal{A}$ on w has promptness bounded by k, thus it visits Q_F at least once every $k+1$ steps. Since Q_F is finite, this run must visit some states in Q_F more than once. Let $q_f \in Q_F$ be the first state appearing twice in the run, this means that there exists a loop in $\mathcal{A}$ from q_f to q_f, moreover, this loop crosses Q_F at least once every $k+1$

steps (due to promptness). By definition of Z^k, those states belonging to Q_F in the loop belong to Z^k. Again, due to the promptness being bounded by k, the portion of the run from q_{init} to the first occurrence of q_f visits Q_F at least once every $k+1$ steps. This implies that all the accepting states in this part of the run are also in Z^k, and therefore $d_{Z^k}(q_{\mathsf{init}}) \leq k$.

$\boxed{3 \implies 1}$ We assume that $d_{Z^k}(q_{\mathsf{init}}) \leq k$. We define below a k-prompt enforcer of $\mathcal{A}$ as a word transducer $E : \Sigma^* \to \Sigma^*$. that will maintain the invariant (1). We construct E in an inductive manner. Initially, we define $E(\varepsilon) = \varepsilon$. The invariant (1) holds initially for ε because either $q_{\mathsf{init}} \in Z^k = Pre^0(Z^k)$ and then $\mathsf{surplus}(\varepsilon) = 0$ and $d_{Z^k}(q_{\mathsf{init}}) = 0$, or $q_{\mathsf{init}} \notin Z^k$, and then $\mathsf{surplus}(\varepsilon) = 1$ and $d_{Z^k}(q_{\mathsf{init}}) \leq k$ by hypothesis. In both of these cases, $sum_{Z^k}(\varepsilon) \leq k+1$ and the invariant is maintained.

Now, consider a finite word w, and assume that $E(w)$ has already been defined, satisfying (1). We claim that $\mathsf{filter}(E(w)) \neq \emptyset$. Define $q_{E(w)} = \delta(q_{\mathsf{init}}, E(w))$. Then, since $E(w)$ satisfies (1), $d_{Z^k}(q_{E(w)}) \leq k+1$. One can then observe that there must exist an action σ such that $d_{Z^k}(\delta(q_{E(w)}, \sigma)) = d_{Z^k}(q_{E(w)}) - 1$ when $\delta(q_{E(w)}, \sigma) \notin Z^k$, and is 0 when $\delta(q_{E(w)}, \sigma) \in Z^k$. Therefore, $sum_{Z^k}(E(w).\sigma) \leq k+1$, since the sum is either $\mathsf{surplus}_{\mathcal{A}}(E(w)) + 1 + d_{Z^k}(q_{E(w)}) - 1$ or it is reset to 0 when $\delta(q_{E(w)}, \sigma) \in Z^k$. Hence, $\sigma \in \mathsf{filter}(E(w))$ and thus $\mathsf{filter}(E(w)) \neq \emptyset$.

When receiving the next action $\sigma \in \Sigma$, the enforcer acts as follows: if $\sigma \in \mathsf{filter}(E(w))$ then $E(w.\sigma) = E(w).\sigma$; and otherwise $E(w.\sigma) = E(w).\sigma'$ for some $\sigma' \in \mathsf{filter}(E(w))$. From the argument presented in the previous paragraph, note that, the invariant (1) is maintained by this construction.

Satisfying the invariant for every word w means the following: $E(w)$ can be extended with a word w' that would reach Z^k within $k+1-\mathsf{surplus}(E(w))$ many steps. The surplus of a prefix of $E(w).w'$ then never exceeds k (surplus is 0 in Z^k). Moreover, the word $E(w).w'$ can be continued infinitely, hitting Z^k at least once every $k+1$ steps. Thus $E(w) \in \mathsf{pref}(\mathcal{L}^k_\omega(\mathcal{A}))$, hence soundness follows.

Furthermore, the construction of the enforcer ensures instantaneity and minimal intervention. Therefore, E is indeed a k-prompt enforcer for $\mathcal{A}$. □

Example 5. Let us consider the automaton $\mathcal{A}$ of Figure 2 and suppose $k = 2$. Figure 5 represents the behaviour of the 2-enforcer on the input word *bcabacba*. Starting with ε at q_0, the $\mathsf{surplus}$ is 1 and distance d_{Z^k} is 2, thus their sum does not exceed $k+1 = 3$. When reading b, q_1 is reached, the surplus increases by 1 while the distance decreases by 1, their sum is thus constant and the enforcer keeps b as output. Then reading c, q_2 is reached which is accepting; both surplus and distance are then set to 0. Then, receiving a, the surplus is 1 and the distance to q_2 is 2 and a is kept unmodified. Receiving b sets the surplus to 2 and distance to 1, then b is kept and we reach q_1. Now, receiving a would loop in q_1, leaving the distance unchanged but increasing the surplus to 3, the sum then is $4 > 3$, and the action a is changed to c which leads to q_2 with both surplus and distance reset to 0. The following actions c and b also lead to q_2 thus are kept as output, and finally a is kept, leading to q_0 again.

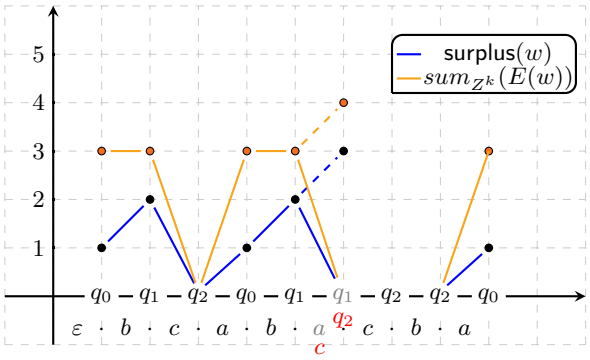


Fig. 5: Behaviour of a 2-enforcer for the automaton $\mathcal{A}$ of Figure 2 on *bcabacba*. The dashed lines represent the values of the surplus and sum_{Z^k} if the enforcer would not have been in play in the last step. The modified action and the resulting states are written in red.

5.2 Prompt Enforcers

Rather than fixing a uniform bound k for all words generated under the enforcer, prompt enforcers (Definition 3) require from the enforcer that each generated infinite word must have bounded promptness for a possibly different bound k. In this section, we show that prompt enforcers can be assumed to be k-prompt enforcers for a well chosen k.

In order to study this case, we will model the prompt enforcement problem using two-player turn-based games [5]. Given $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \delta, Q_F)$, consider the two-player game $\mathcal{G}$ with a Büchi objective, defined as follows. The state space is $Q \cup (Q \times \Sigma)$; Player *Environment* selects $\sigma \in \Sigma$ at a state $q \in Q$, and at any state (q, σ), Player *Enforcer* freely selects $\sigma' \in \Sigma$ and the game moves to state $q' = \delta(q, \sigma')$. Let us put promptness aside for a moment and assume that the objective of Enforcer is to visit Q_F infinitely often, then this is a Büchi game.

In any turn-based two-player game with a Büchi objective, if a player has a winning strategy, then they have a winning strategy under which all outcomes have promptness at most the size of the state space. This is because a winning player has a *positional*[2] winning strategy [5], which, against any adversary, ensures reaching an accepting state at least once every k steps, where k is the size of the state space.

Coming back to our game $\mathcal{G}$, if Enforcer has a winning strategy, then it has one that induces words with promptness at most $k = |Q| + |Q||\Sigma|$. Let $\gamma : Q \times \Sigma \to \Sigma$ denote such a positional winning strategy for Enforcer. Strategy γ corresponds to a prompt enforcer but not necessarily to a k-prompt enforcer since the minimal intervention property might not be satisfied. Nevertheless, a k-prompt enforcer can be derived as follows.

[2] A positional strategy for Enforcer only depends on the current state.

In the game $\mathcal{G}$, the set W of states from which a winning strategy exists can be partitioned into $W_0, W_1, \ldots, W_k$ where $W_0 \subseteq Q_F$, with the following properties:

- for all $1 \leq i \leq k$ and $q \in W_i$, $\forall \sigma \in \Sigma, \delta(q, \gamma(q, \sigma)) \in \cup_{0 \leq j \leq i-1} W_j$,
- for all $q \in W_0$, $\forall \sigma \in \Sigma, \delta(q, \gamma(q, \sigma)) \in W$.

Note that the sequence $(W_i)_i$ corresponds to the attractor computation when solving Büchi games; see [5]. In other terms, the strategy γ ensures that at each step, whatever the environment's action, we make progress towards accepting states, while from W_0, we remain within W.

In our case, given $q \in W$, we have $q \in W_i$ with $i \geq 1$ iff there is $w \in \Sigma^{\leq i}$ such that $\delta(q, w) \in W_0$.

Let us define γ' as follows: For each $(q, \sigma) \in W_i$,

$$\gamma'(q) = \{\sigma' \in \Sigma \mid q \in W_0 \wedge \delta(q, \sigma') \in W \\ \vee \quad \exists 1 \leq i \leq k, q \in W_i \wedge \delta(q, \sigma') \in \cup_{0 \leq j \leq i-1} W_j\}.$$

Thus, $\gamma'(q)$ denotes the set of all σ' which ensure making progress towards W_0. Note that for all $q \in W$, $\gamma(q, \sigma) \in \gamma'(q)$. Then any arbitrary strategy which, at all states (q, σ), selects some action from $\gamma'(q)$ is a winning strategy in $\mathcal{G}$ which induces words with promptness at most k. This is true regardless whether the strategy is positional or not.

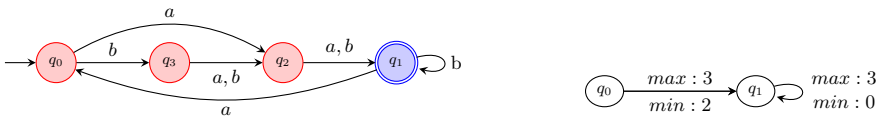
Now, from γ' we can derive a k-prompt enforcer E, built recursively, as follows. Initially $E(\varepsilon) = \varepsilon$, and $q_{\mathsf{init}} \in W$ (since we assumed that there exists a winning strategy). Consider any word w for which $E(w)$ is already defined and consider an input $\sigma \in \Sigma$. Let $q \in W$ be the state $\delta(q_{\mathsf{init}}, E(w))$. Now if $\sigma \in \gamma'(q)$, then $E(w.\sigma) = E(w).\sigma$; and otherwise $E(w.\sigma) = E(w).\sigma'$ for an arbitrary $\sigma' \in \gamma'(q)$. By the previous discussion, E is a winning strategy for Büchi with promptness at most k.

Moreover, E has the minimal intervention property. In fact, consider any word w and $\sigma \in \Sigma$, and let $q \in W_i$ denote the state $\delta(q_{\mathsf{init}}, E(w))$. Assume $E(w.\sigma) \in \mathsf{pref}(\mathcal{L}^k_\omega(\mathcal{A}))$. If $i > 0$, this means that there exists $w' \in \Sigma^{i-1}$ such that $\delta(q, \sigma \cdot w') \in W_0$, and so $E(w.\sigma) = E(w).\sigma$. If $i = 0$, then we necessarily have $\delta(q, \sigma) \in W$ since $E(w.\sigma) \in \mathsf{pref}(\mathcal{L}^k_\omega(\mathcal{A}))$ means that there is a winning strategy from $\delta(q, \sigma)$. In this case we also have $E(w.\sigma) = E(w).\sigma$.

Thus any prompt enforcer corresponds to a winning strategy for the Büchi objective in our game (it actually satisfies the prompt Büchi objective, which is stronger). Conversely, we just showed that a winning strategy for the Büchi objective can be assumed to be a k-prompt enforcer. We get the following theorem.

Theorem 5. *Consider an automaton $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \delta, Q_F)$ and the Büchi game $\mathcal{G}$ described above. The following statements are equivalent:* (1) *Enforcer has a winning strategy in $\mathcal{G}$;* (2) *there exists a prompt enforcer for $\mathcal{A}$;* (3) *there exists a k-prompt enforcer for $\mathcal{A}$ with $k = |Q| + |Q||\Sigma|$.*

Theorem 5 can be used to decide the existence of prompt enforcers, and compute prompt enforcers using the bound $|Q| + |Q||\Sigma|$. However, this bound might in general be too large. We next discuss how to compute a minimal bound k.



(a) Automaton $\mathcal{A}$.

(b) $G_{\mathcal{A}}^{\min}$ and $G_{\mathcal{A}}^{\max}$.

Fig. 6: An automaton and its $G_{\mathcal{A}}^{\min}$ and $G_{\mathcal{A}}^{\max}$ graph.

5.3 Computing the Optimal k for Enforceability

Given an automaton $\mathcal{A}$, we will consider an abstraction in the form of a weighted graph $G_{\mathcal{A}}$ whose vertices are $Q_F \cup \{q_{\mathsf{init}}\}$ and there is an edge from $q \in Q_F \cup \{q_{\mathsf{init}}\}$ to $q' \in Q_F$ if there is a path in $\mathcal{A}$ from q to q' visiting no accepting states (other than q and q'), called a *non-accepting path*. $G_{\mathcal{A}}$ is parameterised by a function weight: we will define $\mathsf{weight}(q, q')$ labelling the edge $q \to q'$ if it exists in $G_{\mathcal{A}}$.

Minimal Promptness. If an automaton $\mathcal{A}$ is k-prompt enforceable, trivially it is also k'-prompt enforceable for every $k' > k$. Thus, for a promptly enforceable property, we are interested in computing the minimal value $k_{\min}$ of k such that the property is k-prompt enforceable.

Given $\mathcal{A}$, we can compute $k_{\min}$ in polynomial (in $|\mathcal{A}|$) time, as follows. We create a weighted graph $G_{\mathcal{A}}^{\min}$, where for every q, q', weight assigns to (q, q') the number of non-accepting states appearing in the shortest non-accepting path from q to q' in $\mathcal{A}$ (including q if it is q_{init} and is not in Q_F).

Let us write the set of all weights appearing in $G_{\mathcal{A}}^{\min}$, in ascending order, as $\zeta_1 \leq \zeta_2 \leq \cdots \leq \zeta_m$. Now, for each $1 \leq i \leq m$, we consider the graph $G_{\mathcal{A}}^{\leq i}$ obtained from $G_{\mathcal{A}}^{\min}$ by removing all edges having weight larger than ζ_i, and check the existence of a lasso that starts from q_{init}: such a lasso corresponds to an accepting lasso in $\mathcal{A}$. The procedure stops whenever such a lasso is found, or when we exhaust the search in all the graphs $G_{\mathcal{A}}^{\leq i}$. If we stop at step i, then $k_{\min} = \zeta_i$ is the minimal value for which there is a $k_{\min}$-prompt enforcer for $\mathcal{A}$. Indeed, the existence of a lasso means that $\mathcal{L}_\omega^{k_{\min}}(\mathcal{A})$ is non-empty, and by Theorem 4 this is equivalent to $\mathcal{A}$ being ζ_i-prompt enforceable. Conversely, note that, if there does not exist such a lasso in $G_{\mathcal{A}}^{\leq i}$ then $\mathcal{L}_\omega^{\zeta_i}(\mathcal{A})$ is empty and hence again from Theorem 4, there is no ζ_i-prompt enforcer for that ζ_i.

Theorem 6. *Given an automaton $\mathcal{A}$, one can compute in polynomial time if $\mathcal{A}$ is k-prompt enforceable for some $k \in \mathbb{N}$, and if it is, one can also compute the minimal k such that $\mathcal{A}$ is k-prompt enforceable.*

For example, consider the automaton $\mathcal{A}$ in Figure 6. The graph $G_{\mathcal{A}}^{\min}$ is given by Figure 6b using "min" weights attached to edges. Clearly $G_{\mathcal{A}}^{\leq 0}$ and $G_{\mathcal{A}}^{\leq 1}$ have no lasso while $G_{\mathcal{A}}^{\leq 2}$ does. We then get $k_{\min} = 2$.

Maximal Permissiveness. There is, in general, a trade-off between promptness of an enforcer and its *permissiveness*. In fact, while a small k is preferable because

the produced words will satisfy $\mathcal{A}$ more often, it also means that the incoming words might be modified too often. An enforcer with a larger k is more permissive since it will let more words pass through and only modify them when necessary (according to the minimal intervention property). Here, we discuss whether there exists *most permissive enforcers* obtained by choosing some large enough k.

We already saw that if $\mathcal{A}$ is k-prompt enforceable it is k'-prompt enforceable for $k' \geq k$. However, in general, there is no maximal value $k_{\max}$ such that the prompt language is 'stable' after this value, that is, $\mathcal{L}_\omega^k(\mathcal{A}) = \mathcal{L}_\omega^{k_{\max}}(\mathcal{A})$, $\forall k \geq k_{\max}$. For instance, consider the automaton $\mathcal{A}$ in Figure 2, we have already shown that this automaton is 2-prompt enforceable (see Figure 5). Now, note that $bcaa^ibcb^\omega$ is in $\mathcal{L}_\omega^{2+i}(\mathcal{A})$ but not in $\mathcal{L}_\omega^{2+i-1}(\mathcal{A})$, for every integer $i \geq 1$. Therefore, in this example, $\mathcal{L}_\omega^k(\mathcal{A}) \subsetneq \mathcal{L}_\omega^{k'}(\mathcal{A})$, for every $k' > k \geq 2$. This happens essentially due to the presence of a loop that does not contain an accepting state (*e.g.* the self-loop at q_0) from which another loop (*e.g.* the loop $q_0 \to q_1 \to q_2 \to q_0$) that contains an accepting state is reachable.

Nevertheless, when no such loops exist, we argue that there indeed exists a maximal bound $k_{\max}$, computable in polynomial time. The algorithm starts by decomposing $\mathcal{A}$ into strongly connected components (SCC) in linear time using Tarjan's algorithm [14]. Let us call an SCC *winning* if it contains an accepting state, and *losing* otherwise. We construct the pruned automaton $\mathcal{A}' = (Q', q_{\mathsf{init}}, \Sigma, \delta', Q'_F)$ from $\mathcal{A}$, by removing all the components from which no winning component is reachable. Note that, this procedure does not alter the language of $\mathcal{A}$, *i.e.* $\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}')$. By hypothesis, $\mathcal{A}'$ contains no non-trivial losing component from which a winning component can be reached. If q_{init} was removed, then $\mathcal{A}$ is not enforceable. Otherwise we build a weighted graph $G_{\mathcal{A}}^{\max}$ from $\mathcal{A}'$ with the weight taken in $\mathcal{A}'$ as follows: for each pair of states $q \in Q'_F \cup \{q_{\mathsf{init}}\}$, $q' \in Q'_F$, $\mathsf{weight}(q, q')$ is the maximal number of non-accepting states in non-accepting paths from q to q'. We then define $k_{\max}$ as follows: $k_{\max} = \max_{q \in Q'_F \cup \{q_{\mathsf{init}}\}, q' \in Q'_F} \mathsf{weight}(q, q')$. Therefore, no prefix of an accepted word in $\mathcal{A}$ has promptness greater than $k_{\max}$. Otherwise said, in every accepting run of $\mathcal{A}'$ (hence, of $\mathcal{A}$ as well) an accepting state is seen every $k_{\max}+1$ steps. This immediately implies the following theorem.

Theorem 7. *Consider an automaton $\mathcal{A}$, and assume that from all reachable[3] and losing SCCs, no winning SCC is reachable. Then there exists $k_{\max} \geq 0$ such that $\forall k \geq k_{\max}, \mathcal{L}_\omega^k(\mathcal{A}) = \mathcal{L}_\omega^{k_{\max}}(\mathcal{A})$.*

Consider again $\mathcal{A}$ in Figure 6. Since there is only one SCC in $\mathcal{A}$ which is accepting, we get $\mathcal{A}' = \mathcal{A}$ here. The graph $G_{\mathcal{A}}^{\max}$ is then the graph of Figure 6b using "max" weights attached to edges. We then get $k_{\max} = 3$.

6 Implementation and Evaluation

Given a regular property specified as an automaton $\mathcal{A}$, in Algorithm 1, we present the implementation of constructing a k-prompt enforcer E (discussed in 5.1) as

[3] An SCC is *reachable* when its states are reachable from the initial state.

Algorithm 1 Pseudocode for implementation of k-enforcer E. The algorithm assumes that the property defined by $\mathcal{A}$ is k-enforceable, that $Z = \nu X.Q_F \cap \mathsf{Pre}^{[1,k]}_{\mathcal{A}_\varphi}(X)$ is known, as well as the distance function $d_{Z^k} : Q \to \mathbb{N}$.

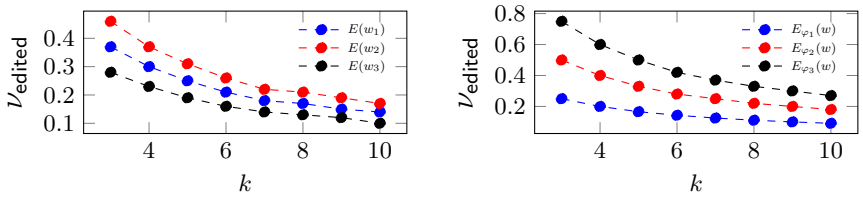
1: Input k, $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \delta, Q_F)$
2: $\mathsf{cs} \leftarrow 0$ **if** $q_{init} \in Q_F$ **else** 1
3: $q \leftarrow q_{init}$
4: $w_E \leftarrow \varepsilon$
5: **while** true:
6: $\quad\sigma \leftarrow \mathsf{next\_action}()$
7: $\quad\mathcal{F} \leftarrow \emptyset$
8: $\quad$**for all** $\sigma' \in \Sigma$:
9: $\quad\quad\mathsf{cs}_{\mathsf{temp}} \leftarrow 0$ **if** $\delta(q, \sigma') = q' \in Q_F$ **else** $\mathsf{cs} + 1$
10: $\quad\quad$**if** $\mathsf{cs}_{\mathsf{temp}} + d_{Z^k}(q') \leq k + 1$: $\mathcal{F} \leftarrow \mathcal{F} \cup \{\sigma'\}$
11: $\quad\sigma' \leftarrow \sigma$ **if** $\sigma \in \mathcal{F}$ **else** pick some σ' in $\mathcal{F}$
12: $\quad q \leftarrow \delta(q, \sigma')$
13: $\quad\mathsf{cs} \leftarrow 0$ **if** $q \in Q_F$ **else** $\mathsf{cs} + 1$
14: $\quad w_E \leftarrow w_E \cdot \sigma'$

an online algorithm. We assume that $\mathcal{A}$ indeed has a k-prompt enforcer. We compare the results produced by our algorithm with that of [10].

Algorithm 1 takes as input an automaton $\mathcal{A}$ and a positive integer k. The actions emitted by the system are fed to the algorithm one at a time and the algorithm modifies this action whenever necessary. The word enforced so far is stored in the variable w_E, initially $w_E = \varepsilon$; the current surplus is maintained in the variable cs: initially 0 if $q_{\mathsf{init}} \in Q_F$, otherwise 1. Further, the algorithm maintains the current state q of the property automaton, initially set to q_{init}. The online enforcement starts with the while loop in step 5. The enforcer receives the next action σ from the system (step 6). Steps 7 through 10 implement the filter function as defined in Page 9. We initialize the filter set $\mathcal{F}$ with $\emptyset$. Then, for each possible action σ', we update the value of the current surplus (in step 9) and compute the sum_{Z^k} (as defined also in Page 9) in step 10. Note here that we assume that Algorithm 1 is also provided with the distance function d_{Z^k}, we implement this function as a simple fixpoint computation. We add σ' to $\mathcal{F}$ if the computed value of sum_{Z^k} does not exceed $k + 1$. Step 11 chooses the enforced letter $\sigma' \in \mathcal{F}$, respecting minimal intervention, *i.e.* keeping $\sigma' = \sigma$ if $\sigma \in \mathcal{F}$. The current word is then updated, and the while loop continues with the next action.

An execution of the algorithm on the automaton $\mathcal{A}$ in Example 1 is provided in the Appendix A.

Algorithm 1 is implemented in Python. The implementation consists of a class called DFA that provides the required functionalities related to defining an automaton and its traversal on a word. We then encode Algorithm 1 as a function $\mathsf{enforce\_event}()$. This function is defined as a member function of the DFA class in the automata module. It takes the current state q, current action σ, and k as arguments, and updates the action and the word according to Algorithm 1 described above. Using a driver program in Python, we import the automata



(a) Varying w and fixed φ

(b) Fixed w and varying φ

Fig. 7: Plot showing change in frequency of 'edited' actions with increasing k.

module, and create an object of the DFA class to specify the property as an automaton. We import the trace of the system dynamically, action by action, and call enforce_event() to enforce the specified property.

6.1 Results and Observations

We tested the *k-prompt* enforcer by enforcing 50 different properties. We used a random automata generator [16] to generate DFA specifications of 50 random properties. We enforced each property on 10 words of varying length (100 to 1000) using our *k-prompt* enforcer. For consistent results, all the words were such that no prefix of any word satisfied the property being enforced. We computed the number of times the property was satisfied, and the number of times the actions in the words were edited. We calculated three normalized frequencies:

- frequency of property satisfaction $\nu_{\text{sat}} = \dfrac{\text{number of accepting prefixes}}{\text{word length}}$,
- frequency of edited events (by the enforcer) $\nu_{\text{edited}} = \dfrac{\text{number of edited events}}{\text{word length}}$,
- and frequency of unedited events $\nu_{\text{unedited}} = 1 - \nu_{\text{edited}}$.

We first demonstrate the behaviours of the enforcer with varying k.

Figure 7a shows that an enforcer designed for a particular property modifies different words differently. On the other hand, in Figure 7b, we keep the input word w fixed, and use enforcers for three different properties. Here, w is chosen such that no prefixes of w satisfy any of φ_1, φ_2, or φ_3. We now make the following observations about ν_{edited} with respect to k from both of the figures: (i) frequency of edited actions (ν_{edited}) depends on both the input word as well as the enforced property; (ii) ν_{edited} monotonically decreases with increasing k, more precisely, $\nu_{\text{edited}} \propto \frac{1}{k}$; (iii) consequently, the frequency of unedited actions $\nu_{\text{unedited}} = 1 - \nu_{\text{edited}}$ monotonically increases with increasing k (cf. Fig 8 in Appendix B).

Higher ν_{unedited} implies lesser deviation from the original output of the system, while higher ν_{sat} implies more frequent property satisfaction. Through above experiments, we observed that ν_{edited} (thus ν_{unedited}) depends on the property being enforced, and the input word. However, for words with no prefix satisfying the property, $\nu_{\text{sat}} \propto \frac{1}{k}$. Therefore, choosing appropriate k is significant to maintain optimum trade-off between ν_{unedited} and ν_{sat} (cf. Figure 9 in Appendix B).

Comparison with [10]. We ported the implementation of [10] (available online) and compared the behaviours of our enforcer with this on the same machine. For the words considered in the previous part, the chosen words had no prefix satisfying the property but always had an extension to property satisfaction. We observed that indeed the enforcer of [10] emitted the words unedited, thereby never satisfying the property (over the length of the word). However, our enforcer ensured the property is satisfied periodically. To illustrate the difference further, we compared the two enforcers (ours with $k = 3$) computed for a fixed property on 10 randomly generated words, the comparison showed that our enforcer ensures the property is satisfied more often by editing more letters. Due to lack of space, we move the comparison (Figures 10, 11) to Appendix.

7 Related Work

Runtime enforcement is a formal monitoring method that guarantees a system's behaviour consistently conforms to specified properties. An enforcement monitor acts as a guard on top of an executing system that observes its execution and transforms the events if necessary to avoid violation of the property being monitored. In order to correct/edit the execution of the system and avoid violation of the policy being monitored, different enforcement actions such as *halt, suppress, store, delay, and reorder* [13,7,4,12,6,9] are applied.

When we consider enforcement techniques for reactive systems which continuously interact with its environment, enforcement actions such as halting, delaying and suppression are not suitable.

In another approach for runtime enforcement in reactive systems [2], Bloem *et al.* propose synthesis of online safety enforcers called *shields*. Shields are capable of efficiently enforcing a smaller number of safety critical properties, by allowing deviation from non-critical properties of the system. They have a notion of *k-stabilization* where minimum deviation from original output of the system (and consequently deviation from non-critical properties) is allowed for at most k consecutive steps after the initial violation of a safety critical property. However if a second violation (of a critical property) occurs within those k steps, the shield enters *fail-safe* mode where minimum deviation is permanently ignored in order to satisfy the critical safety property. In a similar work [15], Wu *et al.* propose an approach that provides instantaneous recovery from property violation, thereby allowing persistent minimum deviation from system's outputs. The notion of k-promptness in our work is different from that of k-stabilization in shields. The k-stabilization refers to the recovery phase (of k-steps) after a violation of the property has occurred, while k-promptness refers to property satisfaction at least once every $k + 1$ steps.

There are other streams of work related to enforcement for reactive systems such as [11]. The framework in [11] focuses only on prefix-closed (safety) properties that instantaneously edits actions (only when necessary) to enforce the property being monitored. The approach is later extended to general regular

properties in the discrete-time context [10,8] where properties to be enforced are modelled using an extension of finite state automata with discrete clocks.

Similar to approaches in [11,10,8] we focus on enforcement for reactive systems where the enforcer has to instantaneously correct the events. To simplify the presentation and formalization, in this work we do not explicitly take into account input-output parts in formalizing events for bi-directional enforcement considered in approaches such as [11,10,8]. Our work can be easily adapted to the bi-directional setup. In [11,10,8] when enforcing a prefix-closed property (*i.e.*, a safety property), the synthesized enforcer ensures that the output emitted satisfies the property being monitored at every step. However in [10,8] when considering general regular properties it is not possible to ensure that the output after every step satisfies the property. For regular properties, the enforcer thus ensures that the output can always be extended to eventually to satisfy the property (the output emitted is a prefix of a word that satisfies the property). However, there is also a possibility that eventually it may never end up in a word that actually satisfies the property being monitored.

As far as we know, none of the existing enforcement frameworks of regular properties for reactive systems can ensure periodic satisfaction of the property by the output emitted by the enforcer. In this work, we thus propose a new enforcement framework for regular properties with prompt eventualities. Given any regular property defined as an automaton and a bound k, the synthesized enforcer ensures that the output emitted by it satisfies the property being monitored at least once every $k+1$ steps. The enforcer obtained using the proposed framework also fulfils other constraints such as monotonicity and transparency.

As already mentioned, bounding the number of steps before an event in temporal logics was considered in [3], and in other subsequent works. In particular, prompt Büchi acceptance is defined in [1] by existentially quantifying the bound k: a word is in the language if there exists k such that a k-prompt run can be built in $\mathcal{A}$. Our work is clearly based on these.

8 Conclusion and Future Work

We formalize for the first time the prompt runtime enforcement synthesis problem for reactive systems. We developed a new RE framework with prompt eventualities for regular properties. For any given property φ and a bound k, the enforcer synthesized using the framework ensures that φ is never violated for more that k consecutive steps. We formally define this RE problem, delineate the k-enforceability of automata, and provide algorithms for the design of the enforcer. We consider the prompt-enforcement problem where the bound k is not fixed, and show how the problem reduces to k-enforcement problem for sufficiently large k. We show the computation of minimal k for the property to be k-enforceable. We also implement the k-prompt enforcer in Python and show how promptness and permissiveness of the enforcer change with varying k.

In the near future, we will consider extending the prompt enforcement framework in multiple directions including tackling regular timed properties.

References

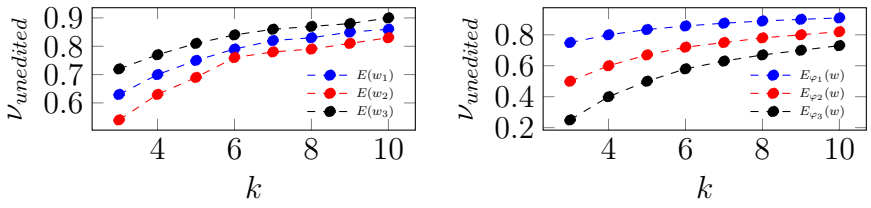
1. Almagor, S., Hirshfeld, Y., Kupferman, O.: Promptness in ω-regular automata. In: Automated Technology for Verification and Analysis: 8th International Symposium, ATVA 2010, Singapore, September 21-24, 2010. Proceedings 8. pp. 22–36. Springer (2010)
2. Bloem, R., Könighofer, B., Könighofer, R., Wang, C.: Shield synthesis: Runtime enforcement for reactive systems. In: International conference on tools and algorithms for the construction and analysis of systems (TACAS). pp. 533–548. Springer (2015)
3. Dershowitz, N., Jayasimha, D., Park, S.: Bounded fairness. Verification: Theory and Practice: Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday pp. 304–317 (2003)
4. Falcone, Y., Mounier, L., Fernandez, J.C., Richier, J.L.: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Formal Methods in System Design 38(3), 223–262 (2011)
5. Fijalkow, N., Bertrand, N., Bouyer-Decitre, P., Brenguier, R., Carayol, A., Fearnley, J., Gimbert, H., Horn, F., Ibsen-Jensen, R., Markey, N., et al.: Games on graphs. arXiv preprint arXiv:2305.10546 (2023)
6. Ligatti, J., Bauer, L., Walker, D.: Edit automata: Enforcement mechanisms for run-time security policies. International Journal of Information Security 4, 2–16 (2005)
7. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Trans. Inf. Syst. Secur. 12(3) (Jan 2009)
8. Pearce, H.A., Pinisetty, S., Roop, P.S., Kuo, M.M.Y., Ukil, A.: Smart I/O modules for mitigating cyber-physical attacks on industrial control systems. IEEE Trans. Ind. Informatics 16(7), 4659–4669 (2020)
9. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H., Rollet, A., Nguena Timo, O.: Runtime enforcement of timed properties revisited. Formal Methods in System Design 45, 381–422 (2014)
10. Pinisetty, S., Roop, P.S., Smyth, S., Allen, N., Tripakis, S., von Hanxleden, R.: Runtime enforcement of cyber-physical systems. ACM Trans. Embed. Comput. Syst. 16(5s), 178:1–178:25 (2017), https://doi.org/10.1145/3126500
11. Pinisetty, S., Roop, P.S., Smyth, S., Tripakis, S., von Hanxleden, R.: Runtime enforcement of reactive systems using synchronous enforcers. In: Erdogmus, H., Havelund, K. (eds.) Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017. pp. 80–89. ACM (2017)
12. Pradhan, A., Akil, C.G.M., Pinisetty, S.: Runtime enforcement with event reordering. In: Anutariya, C., Bonsangue, M.M. (eds.) Theoretical Aspects of Computing - ICTAC 2024 - 21st International Colloquium, Bangkok, Thailand, November 25-29, 2024, Proceedings. Lecture Notes in Computer Science, vol. 15373, pp. 386–407. Springer (2024), https://doi.org/10.1007/978-3-031-77019-7_22
13. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3(1), 30–50 (2000)
14. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM journal on computing 1(2), 146–160 (1972)
15. Wu, M., Zeng, H., Wang, C.: Synthesizing runtime enforcer of safety properties under burst error. In: Rayadurgam, S., Tkachuk, O. (eds.) NASA Formal Methods - 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016,

Proceedings. Lecture Notes in Computer Science, vol. 9690, pp. 65–81. Springer (2016), https://doi.org/10.1007/978-3-319-40648-0_6
16. Zuzak, I., Jankovic, V.: FSM simulator. https://ivanzuzak.info/noam/webapps/fsm_simulator/, accessed: 2025-4-21

A Appendix (A run of Algorithm 1)

We illustrate the behavior of Algorithm 1 for the property specified by $\mathcal{A}$ in Example 1, for promptness $k = 2$, and for a given input word $w = bcaba$ produced by the system. This example illustrates how the algorithm edits w in order to enforce $\mathcal{L}^2_\omega(\mathcal{A})$. Initially, $w_E = \varepsilon$, current state $q = q_0$, and $\mathsf{cs} = 1$, since $q_0 \notin Q_F$. $Z^2 = \{q_2\}$, and $d_{Z^2}(q_0)$, $d_{Z^2}(q_1)$, $d_{Z^2}(q_2)$, $d_{Z^2}(q_3)$ and $d_{Z^2}(q_4)$ are 2, 1, 0, ∞ and ∞ respectively. In the first iteration of the while loop, $\sigma = b$, $q = q_0$, and $(\delta(q,a), \delta(q,b), \delta(q,c)) = (q_0, q_1, q_3)$, and $\mathsf{surplus}(w \cdot a) = \mathsf{surplus}(w \cdot b) = \mathsf{surplus}(w \cdot c) = 2$, and $d_{Z^2}(q_1) + 2 < d_{Z^2}(q_0) + 2 \leq 3$. The set $\mathcal{F}$ is computed as $\{a, b\}$, and since $\sigma = b \in \mathcal{F}$, the action remains unchanged, and transition $\delta(q_0, b) = q_1$ is taken to update the current state, and $\sigma' = b$ is appended to the word w_E. In the second iteration, $\sigma = c$, $q = q_1$, and $(\delta(q,a), \delta(q,b), \delta(q,c))$ $= (q_1, q_1, q_2)$, and $\mathsf{surplus}(w \cdot a) = \mathsf{surplus}(w \cdot b) = \mathsf{surplus}(w \cdot c) = 3$, and $d_{Z^2}(q_1) + 3 \not\leq 3$ while $d_{Z^2}(q_2) + 3 \leq 3$. Now $\mathcal{F} = \{c\}$. Since $\sigma = c \in \mathcal{F}$, $\sigma = c$ remains unchanged. The current state is updated to q_2, and $w_E = bc$. The iteration continues with upcoming events in sequence a and b which also remain unchanged. In the fifth iteration, $\sigma = a$, $q = q_1$, and $(\delta(q,a), \delta(q,b), \delta(q,c))$ $= (q_1, q_1, q_2)$, and $\mathsf{surplus}(w \cdot a) = \mathsf{surplus}(w \cdot b) = \mathsf{surplus}(w \cdot c) = 3$, and $d_{Z^2}(q_1) + 3 \not\leq 3$ while $d_{Z^2}(q_2) + 3 \leq 3$. Now $\mathcal{F} = \{c\}$. $\sigma = a \notin \mathcal{F}$, and therefore $\sigma' = c$ is chosen from $\mathcal{F}$, and the current state transitions to q_2. We can see that the word $bcaba \notin \mathcal{L}^2_\omega(\mathcal{A})$ is edited to $bcabc \in \mathcal{L}^2_\omega(\mathcal{A})$.

B Appendix (Observations- Additional Plots)



(a) Varying w and fixed φ

(b) Fixed w and varying φ

Fig. 8: Plot showing change in frequency of 'unedited' actions with increasing k.

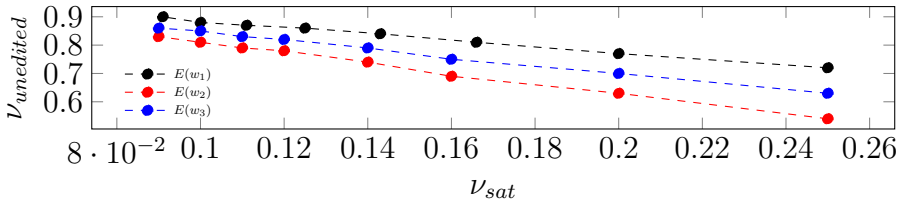


Fig. 9: Plot showing relationship between ν_{edited} and ν_{sat}.

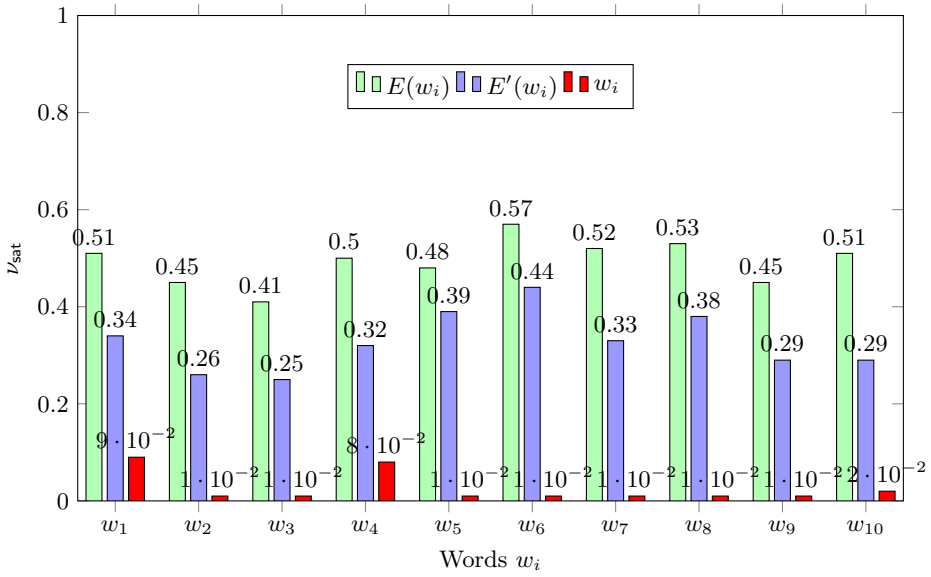


Fig. 10: Plot showing frequency of policy satisfaction ν_{sat} by output of 3-prompt enforcer $E(w)$, output of enforcer in [10] $E'(w)$, and original (unedited) input word w.

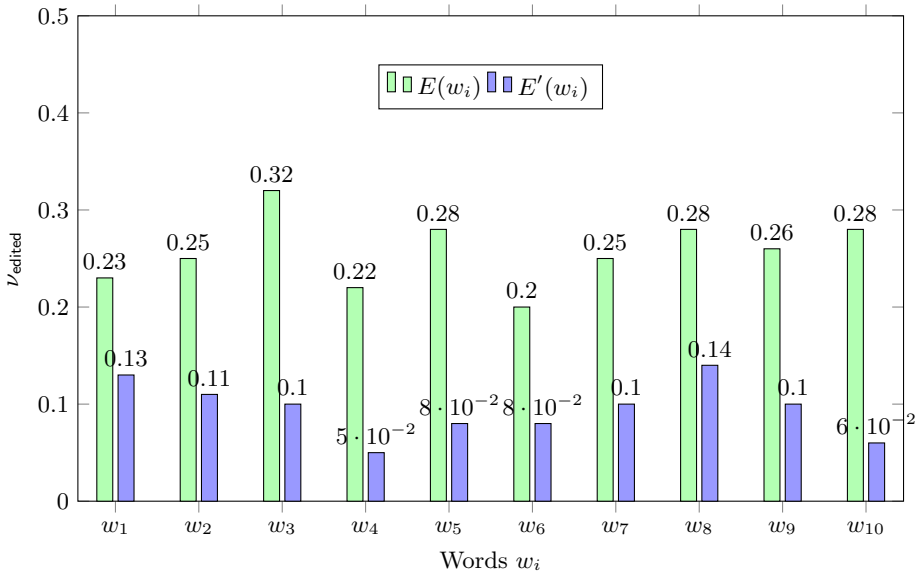


Fig. 11: Plot showing frequency of edited events ν_{edited} by output of 3-prompt enforcer $E(w)$, and output of enforcer in [10] $E'(w)$.