Selective Monitoring

Radu Grigore[1]
University of Kent, UK
https://orcid.org/0000-0003-1128-0311

Stefan Kiefer[2]
University of Oxford, UK

Abstract

We study selective monitors for labelled Markov chains. Monitors observe the outputs that are generated by a Markov chain during its run, with the goal of identifying runs as correct or faulty. A monitor is selective if it skips observations in order to reduce monitoring overhead. We are interested in monitors that minimize the expected number of observations. We establish an undecidability result for selectively monitoring general Markov chains. On the other hand, we show for non-hidden Markov chains (where any output identifies the state the Markov chain is in) that simple optimal monitors exist and can be computed efficiently, based on DFA language equivalence. These monitors do not depend on the precise transition probabilities in the Markov chain. We report on experiments where we compute these monitors for several open-source Java projects.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases runtime monitoring, probabilistic systems, Markov chains, automata, language equivalence

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.20

Related Version https://arxiv.org/abs/1806.06143

1 Introduction

Consider an MC (Markov chain) whose transitions are labelled with letters, and a finite automaton that accepts languages of infinite words. Computing the probability that the random word emitted by the MC is accepted by the automaton is a classical problem at the heart of probabilistic verification. A finite prefix may already determine whether the random infinite word is accepted, and computing the probability that such a *deciding* finite prefix is produced is a nontrivial *diagnosability* problem. The theoretical problem we study in this paper is how to catch deciding prefixes without observing the whole prefix; i.e., we want to minimize the expected number of observations and still catch all deciding prefixes.

Motivation. In runtime verification a program sends messages to a monitor, which decides if the program run is faulty. Usually, runtime verification is turned off in production code because monitoring overhead is prohibitive. QVM (quality virtual machine) and ARV (adaptive runtime verification) are existing pragmatic solutions to the overhead problem, which perform best-effort monitoring within a specified overhead budget [1, 3]. ARV relies

[1] Work supported by EPSRC grant EP/R012261/1.
[2] Work supported by a Royal Society University Research Fellowship.

on RVSE (runtime verification with state estimation) to also compute a probability that the program run is faulty [21, 15]. We take the opposite approach: we ask for the smallest overhead achievable without compromising precision at all.

Previous Work. Before worrying about the performance of a monitor, one might want to check if faults in a given system can be diagnosed at all. This problem has been studied under the term *diagnosability*, first for non-stochastic finite discrete event systems [19], which are labelled transition systems. It was shown in [14] that diagnosability can be checked in polynomial time, although the associated monitors may have exponential size. Later the notion of diagnosability was extended to stochastic discrete-event systems, which are labelled Markov chains [22]. Several notions of diagnosability in stochastic systems exist, and some of them have several names, see, e.g., [20, 4] and the references therein. Bertrand et al. [4] also compare the notions. For instance, they show that for one variant of the problem (referred to as A-diagnosability or SS-diagnosability or IF-diagnosability) a previously proposed polynomial-time algorithm is incorrect, and prove that this notion of diagnosability is PSPACE-complete. Indeed, most variants of diagnosability for stochastic systems are PSPACE-complete [4], with the notable exception of AA-diagnosability (where the monitor is allowed to diagnose wrongly with arbitrarily small probability), which can be solved in polynomial time [5].

Selective Monitoring. In this paper, we seem to make the problem harder: since observations by a monitor come with a performance overhead, we allow the monitor to skip observations. In order to decide how many observations to skip, the monitor employs an *observation policy*. Skipping observations might decrease the probability of deciding (whether the current run of the system is faulty or correct). We do not study this tradeoff: we require policies to be *feasible*, i.e., the probability of deciding must be as high as under the policy that observes everything. We do not require the system to be diagnosable; i.e., the probability of deciding may be less than 1. Checking whether the system is diagnosable is PSPACE-complete ([4], Theorem 8).

The Cost of Decision in General Markov Chains. The *cost* (of decision) is the number of observations that the policy makes during a run of the system. We are interested in minimizing the expected cost among all feasible policies. We show that if the system is diagnosable then there exists a policy with finite expected cost, i.e., the policy may stop observing after finite expected time. (The converse is not true.) Whether the infimum cost (among feasible policies) is finite is also PSPACE-complete (Theorem 14). Whether there is a feasible policy whose expected cost is smaller than a given threshold is undecidable (Theorem 15), even for diagnosable systems.

Non-Hidden Markov Chains. We identify a class of MCs, namely non-hidden MCs, where the picture is much brighter. An MC is called *non-hidden* when each label identifies the state. Non-hidden MCs are always diagnosable. Moreover, we show that *maximally procrastinating* policies are (almost) optimal (Theorem 27). A policy is called maximally procrastinating when it skips observations up to the point where one further skip would put a decision on the current run in question. We also show that one can construct an (almost) optimal maximally procrastinating policy in polynomial time. This policy *does not depend* on the exact probabilities in the MC, although the expected cost under that policy does. That is, we efficiently construct a policy that is (almost) optimal regardless of the transition probabilities

on the MC transitions. We also show that the infimum cost (among all feasible policies) can be computed in polynomial time (Theorem 28). Underlying these results is a theory based on automata, in particular, checking language equivalence of DFAs.

Experiments. We evaluated the algorithms presented in this paper by implementing them in Facebook Infer, and trying them on 11 of the most forked Java projects on GitHub. We found that, on average, selective monitoring can reduce the number of observations to a half.

2 Preliminaries

Let S be a finite set. We view elements of $\mathbb{R}^S$ as *vectors*, more specifically as row vectors. We write $\mathbf{1}$ for the all-1 vector, i.e., the element of $\{1\}^S$. For a vector $\mu \in \mathbb{R}^S$, we denote by μ^T its transpose, a column vector. A vector $\mu \in [0,1]^S$ is a *distribution over* S if $\mu\mathbf{1}^\mathsf{T} = 1$. For $s \in S$ we write e_s for the (*Dirac*) distribution over S with $e_s(s) = 1$ and $e_s(t) = 0$ for $t \in S \setminus \{s\}$. We view elements of $\mathbb{R}^{S\times S}$ as *matrices*. A matrix $M \in [0,1]^{S\times S}$ is called *stochastic* if each row sums up to one, i.e., $M\mathbf{1}^\mathsf{T} = \mathbf{1}^\mathsf{T}$.

For a finite alphabet Σ, we write Σ^* and Σ^ω for the finite and infinite words over Σ, respectively. We write ε for the empty word. We represent languages $L \subseteq \Sigma^\omega$ using deterministic finite automata, and we represent probability measures Pr over Σ^ω using Markov chains.

A (discrete-time, finite-state, labelled) *Markov chain (MC)* is a quadruple (S, Σ, M, s_0) where S is a finite set of states, Σ a finite alphabet, s_0 an initial state, and $M : \Sigma \to [0,1]^{S\times S}$ specifies the transitions, such that $\sum_{a\in\Sigma} M(a)$ is a stochastic matrix. Intuitively, if the MC is in state s, then with probability $M(a)(s,s')$ it emits a and moves to state s'. For the complexity results in this paper, we assume that all numbers in the matrices $M(a)$ for $a \in \Sigma$ are rationals given as fractions of integers represented in binary. We extend M to the mapping $M : \Sigma^* \to [0,1]^{S\times S}$ with $M(a_1 \cdots a_k) = M(a_1) \cdots M(a_k)$ for $a_1, \ldots, a_k \in \Sigma$. Intuitively, if the MC is in state s then with probability $M(u)(s,s')$ it emits the word $u \in \Sigma^*$ and moves (in $|u|$ steps) to state s'. An MC is called *non-hidden* if for each $a \in \Sigma$ all non-zero entries of $M(a)$ are in the same column. Intuitively, in a non-hidden MC, the emitted letter identifies the next state. An MC (S, Σ, M, s_0) defines the standard probability measure Pr over Σ^ω, uniquely defined by assigning probabilities to cylinder sets $\{u\}\Sigma^\omega$, with $u \in \Sigma^*$, as follows:

$$\Pr(\{u\}\Sigma^\omega) \;\coloneqq\; e_{s_0} M(u) \mathbf{1}^\mathsf{T}$$

A *deterministic finite automaton (DFA)* is a quintuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ a finite alphabet, $\delta : Q \times \Sigma \to Q$ a transition function, q_0 an initial state, and $F \subseteq Q$ a set of accepting states. We extend δ to $\delta : Q \times \Sigma^* \to Q$ as usual. A DFA defines a language $L \subseteq \Sigma^\omega$ as follows:

$$L \;\coloneqq\; \{\, w \in \Sigma^\omega \mid \delta(q_0, u) \in F \text{ for some prefix } u \text{ of } w \,\}$$

Note that we do not require accepting states to be visited infinitely often: just once suffices. Therefore we can and will assume without loss of generality that there is f with $F = \{f\}$ and $\delta(f, a) = f$ for all $a \in \Sigma$.

For the rest of the paper we fix an MC $\mathcal{M} = (S, \Sigma, M, s_0)$ and a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$. We define their composition as the MC $\mathcal{M} \times \mathcal{A} \;\coloneqq\; (S \times Q, \Sigma, M', (s_0, q_0))$ where $M'(a)((s,q),(s',q'))$ equals $M(a)(s,s')$ if $q' = \delta(q,a)$ and 0 otherwise. Thus, $\mathcal{M}$ and $\mathcal{M} \times \mathcal{A}$ induce the same probability measure Pr.

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang; Article No. 20; pp. 20:3–20:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An *observation* $o \in \Sigma_\perp$ is either a letter or the special symbol $\perp \notin \Sigma$, which stands for 'not seen'. An *observation policy* $\rho : \Sigma_\perp^* \to \{0,1\}$ is a (not necessarily computable) function that, given the observations made so far, says whether we should observe the next letter. An observation policy ρ determines a projection $\pi_\rho : \Sigma^\omega \to \Sigma_\perp^\omega$: we have $\pi_\rho(a_1 a_2 \dots) = o_1 o_2 \dots$ when

$$o_{n+1} \;=\; \begin{cases} a_{n+1} & \text{if } \rho(o_1 \dots o_n) = 1 \\ \perp & \text{if } \rho(o_1 \dots o_n) = 0 \end{cases} \qquad \text{for all } n \geq 0$$

We denote the *see-all policy* by $\bullet$; thus, $\pi_\bullet(w) = w$.

In the rest of the paper we reserve a for letters, o for observations, u for finite words, w for infinite words, v for finite observation prefixes, s for states from an MC, and q for states from a DFA. We write $o_1 \sim o_2$ when o_1 and o_2 are the same or at least one of them is $\perp$. We lift this relation to (finite and infinite) sequences of observations (of the same length). We write $w \gtrsim v$ when $u \sim v$ holds for the length-$|v|$ prefix u of w.

We say that v is *negatively deciding* when $\Pr(\{w \gtrsim v \mid w \in L\}) = 0$. Intuitively, v is negatively deciding when v is incompatible (up to a null set) with L. Similarly, we say that v is *positively deciding* when $\Pr(\{w \gtrsim v \mid w \notin L\}) = 0$. An observation prefix v is *deciding* when it is positively or negatively deciding. An observation policy ρ *decides* w when $\pi_\rho(w)$ has a deciding prefix. A *monitor* is an interactive algorithm that implements an observation policy: it processes a stream of letters and, after each letter, it replies with one of 'yes', 'no', or 'skip n letters', where $n \in \mathbb{N} \cup \{\infty\}$.

▶ **Lemma 1.** *For any w, if some policy decides w then $\bullet$ decides w.*

Proof. Let ρ decide w. Then there is a deciding prefix v of $\pi_\rho(w)$. Suppose v is positively deciding, i.e., $\Pr(\{w' \gtrsim v \mid w' \notin L\}) = 0$. Let u be the length-$|v|$ prefix of w. Then $\Pr(\{w' \gtrsim u \mid w' \notin L\}) = 0$, since v can be obtained from u by possibly replacing some letters with $\perp$. Hence u is also positively deciding. Since u is a prefix of $w = \pi_\bullet(w)$, we have that $\bullet$ decides w. The case where v is negatively deciding is similar. ◀

It follows that $\max_\rho \Pr(\{w \mid \rho \text{ decides } w\}) = \Pr(\{w \mid \bullet \text{ decides } w\})$. We say that a policy ρ is *feasible* when it also attains the maximum, i.e., when

$$\Pr(\{w \mid \rho \text{ decides } w\}) \;=\; \Pr(\{w \mid \bullet \text{ decides } w\})\,.$$

Equivalently, ρ is feasible when $\Pr(\{w \mid \bullet \text{ decides } w \text{ implies } \rho \text{ decides } w\}) = 1$, i.e., almost all words that are decided by the see-all policy are also decided by ρ. If $v = o_1 o_2 \dots$ is the shortest prefix of $\pi_\rho(w)$ that is deciding, then the *cost of decision* $C_\rho(w)$ is $\sum_{k=0}^{|v|-1} \rho(o_1 \dots o_k)$. This paper is about finding feasible observation policies ρ that minimize $\mathrm{Ex}(C_\rho)$, the expectation of the cost of decision with respect to Pr.

3 Qualitative Analysis of Observation Policies

In this section we study properties of observation policies that are qualitative, i.e., not directly related to the cost of decision. We focus on properties of observation prefixes that a policy may produce.

Observation Prefixes.

We have already defined deciding observation prefixes. We now define several other types of prefixes: enabled, confused, very confused, and finitary. A prefix v is *enabled* if it occurs with

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang; Article No. 20; pp. 20:4–20:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

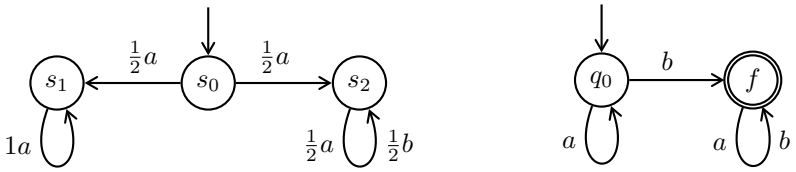
positive probability, $\Pr(\{w \gtrsim v\}) > 0$. Intuitively, the other types of prefixes v are defined in terms of what would happen if we were to observe all from now on: if it is not almost sure that eventually a deciding prefix is reached, then we say v is confused; if it is almost sure that a deciding prefix will not be reached, then we say v is very confused; if it is almost sure that eventually a deciding or very confused prefix is reached, then we say v is finitary. To say this formally, let us make a few notational conventions: for an observation prefix v, we write $\Pr(v)$ as a shorthand for $\Pr(\{\, uw \mid u \sim v \,\})$; for a set Υ of observation prefixes, we write $\Pr(\Upsilon)$ as a shorthand for $\Pr\big(\bigcup_{v\in\Upsilon}\{\, uw \mid u \sim v \,\}\big)$. With these conventions, we define:

1. v is *confused* when $\Pr(\{\, vu \mid vu \text{ deciding} \,\}) < \Pr(v)$
2. v is *very confused* when $\Pr(\{\, vu \mid vu \text{ deciding} \,\}) = 0$
3. v is *finitary* when $\Pr(\{\, vu \mid vu \text{ deciding or very confused} \,\}) = \Pr(v)$

Observe that (a) confused implies enabled, (b) deciding implies not confused, and (c) enabled and very confused implies confused. The following are alternative equivalent definitions:

1. v is *confused* when $\Pr(\{\, uw \mid u \sim v,\ \text{no prefix of } vw \text{ is deciding} \,\}) > 0$
2. v is *very confused* when vu' is non-deciding for all enabled vu'
3. v is *finitary* when $\Pr(\{uw \mid u \sim v,\ \text{no prefix of } vw \text{ is deciding or very confused}\}) = 0$

▶ **Example 2.** Consider the MC and the DFA depicted here:



All observation prefixes that do not start with b are enabled. The observation prefixes ab and $\bot b$ and, in fact, all observation prefixes that contain b, are positively deciding. For all $n \in \mathbb{N}$ we have $\Pr(\{w \gtrsim a^n \mid w \in L\}) > 0$ and $\Pr(\{w \gtrsim a^n \mid w \notin L\}) > 0$, so a^n is not deciding. If the MC takes the right transition first then almost surely it emits b at some point. Thus $\Pr(\{aaa\cdots\}) = \frac{1}{2}$. Hence ε is confused. In this example only non-enabled observation prefixes are very confused. It follows that ε is not finitary. ◀

Beliefs.

For any s we write $\Pr_s$ for the probability measure of the MC $\mathcal{M}_s$ obtained from $\mathcal{M}$ by making s the initial state. For any q we write $L_q \subseteq \Sigma^\omega$ for the language of the DFA $\mathcal{A}_q$ obtained from $\mathcal{A}$ by making q the initial state. We call a pair (s,q) *negatively deciding* when $\Pr_s(L_q) = 0$; similarly, we call (s,q) *positively deciding* when $\Pr_s(L_q) = 1$. A subset of $S \times Q$ is called *belief*. We call a belief *negatively* (*positively*, respectively) *deciding* when all its elements are. We fix the notation $B_0 := \{(s_0, q_0)\}$ (for the *initial belief*) for the remainder of the paper. Define the *belief NFA* as the NFA $\mathcal{B} = (S \times Q, \Sigma_\bot, \Delta, B_0, \emptyset)$ with:

$$\begin{aligned} \Delta((s,q),a) &= \{(s',q') \mid\ M(a)(s,s') > 0,\ \delta(q,a) = q'\} \quad \text{for } a \in \Sigma \\ \Delta((s,q),\bot) &= \bigcup_{a\in\Sigma} \Delta((s,q),a) \end{aligned}$$

We extend the transition function $\Delta : (S \times Q) \times \Sigma_\bot \to 2^{S\times Q}$ to $\Delta : 2^{S\times Q} \times \Sigma_\bot^* \to 2^{S\times Q}$ in the way that is usual for NFAs. Intuitively, if belief B is the set of states where the product $\mathcal{M} \times \mathcal{A}$ could be now, then $\Delta(B, v)$ is the belief adjusted by additionally observing v. To

reason about observation prefixes v algorithmically, it will be convenient to reason about the belief $\Delta(B_0, v)$.

We define confused, very confused, and finitary beliefs as follows:

1. B is *confused* when $\Pr_s(\{\, uw \mid \Delta(B,u) \text{ deciding} \,\}) < 1$ for some $(s,q) \in B$
2. B is *very confused* when $\Delta(B,u)$ is empty or not deciding for all u
3. B is *finitary* when $\Pr_s(\{\, uw \mid \Delta(B,u) \text{ deciding or very confused} \,\}) = 1$ for all $(s,q) \in B$

▶ **Example 3.** In Example 2 we have $B_0 = \{(s_0,q_0)\}$, and $\Delta(B_0, a^n) = \{(s_1,q_0),(s_2,q_0)\}$ for all $n \geq 1$, and $\Delta(B_0, b) = \emptyset$, and $\Delta(B_0, a\bot) = \{(s_1,q_0),(s_2,q_0),(s_2,f)\}$, and $\Delta(B_0, \bot v) = \{(s_2,f)\}$ for all v that contain b. The latter belief $\{(s_2,f)\}$ is positively deciding. We have $\Pr_{s_1}(\{uw \mid \Delta(\{(s_1,q_0)\},u) \text{ is deciding}\}) = 0$, so any belief that contains (s_1,q_0) is confused. Also, B_0 is confused as $\Pr_{s_0}(\{uw \mid \Delta(\{(s_0,q_0)\},u) \text{ is deciding}\}) = \frac{1}{2}$. ◀

Relation Between Observation Prefixes and Beliefs.

By the following lemma, the corresponding properties of observation prefixes and beliefs are closely related.

▶ **Lemma 4.** *Let v be an observation prefix.*

1. *v is enabled if and only if $\Delta(B_0, v) \neq \emptyset$.*
2. *v is negatively deciding if and only if $\Delta(B_0, v)$ is negatively deciding.*
3. *v is positively deciding if and only if $\Delta(B_0, v)$ is positively deciding.*
4. *v is confused if and only if $\Delta(B_0, v)$ is confused.*
5. *v is very confused if and only if $\Delta(B_0, v)$ is very confused.*
6. *v is finitary if and only if $\Delta(B_0, v)$ is finitary.*

The following lemma gives complexity bounds for computing these properties.

▶ **Lemma 5.** *Let v be an observation prefix, and B a belief.*

1. *Whether v is enabled can be decided in P.*
2. *Whether v (or B) is negatively deciding can be decided in P.*
3. *Whether v (or B) is positively deciding can be decided in P.*
4. *Whether v (or B) is confused can be decided in PSPACE.*
5. *Whether v (or B) is very confused can be decided in PSPACE.*
6. *Whether v (or B) is finitary can be decided in PSPACE.*

Proof sketch. The belief NFA $\mathcal{B}$ and the MC $\mathcal{M} \times \mathcal{A}$ can be computed in polynomial time (even in deterministic logspace). For items 1–3, there are efficient graph algorithms that search these product structures. For instance, to show that a given pair (s_1,q_1) is not negatively deciding, it suffices to show that $\mathcal{B}$ has a path from (s_1,q_1) to a state (s_2,f) for some s_2. This can be checked in polynomial time (even in NL).

For items 4–6, one searches the (exponential-sized) product of $\mathcal{M}$ and the *determinization* of $\mathcal{B}$. This can be done in PSPACE. For instance, to show that a given belief B is confused, it suffices to show that there are $(s_1,q_1) \in B$ and u_1 and s_2 such that $\mathcal{M}$ has a u_1-labelled path from s_1 to s_2 such that there do *not* exist u_2 and s_3 such that $\mathcal{M}$ has a u_2-labelled path from s_2 to s_3 such that $\Delta(B, u_1u_2)$ is deciding. This can be checked in NPSPACE = PSPACE by nondeterministically guessing paths in the product of $\mathcal{M}$ and the determinization of $\mathcal{B}$. ◀

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang; Article No. 20; pp. 20:6–20:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Diagnosability.

We call a policy a *diagnoser* when it decides almost surely.

▶ **Example 6.** In Example 2 a diagnoser does not exist. Indeed, the policy $\bullet$ does not decide when the MC takes the left transition, and decides (positively) almost surely when the MC takes the right transition in the first step. Hence $\Pr(\{w \mid \bullet \text{ decides } w\}) = \Pr(\Sigma^*\{b\}\Sigma^\omega) = \frac{1}{2}$. So $\bullet$ is not a diagnoser. By Lemma 1, it follows that there is no diagnoser. ◀

Diagnosability can be characterized by the notion of confusion:

▶ **Proposition 7.** *There exists a diagnoser if and only if ε is not confused.*

The following proposition shows that diagnosability is hard to check.

▶ **Theorem 8** (cf. [4, Theorem 6])**.** *Given an MC $\mathcal{M}$ and a DFA $\mathcal{A}$, it is PSPACE-complete to check if there exists a diagnoser.*

Theorem 8 essentially follows from a result by Bertrand et al. [4]. They study several different notions of diagnosability; one of them (*FA-diagnosability*) is very similar to our notion of diagnosability. There are several small differences; e.g., their systems are not necessarily products of an MC and a DFA. Therefore we give a self-contained proof of Theorem 8.

Proof sketch. By Proposition 7 it suffices to show PSPACE-completeness of checking whether ε is confused. Membership in PSPACE follows from Lemma 5.4. For hardness we reduce from the following problem: given an NFA $\mathcal{U}$ over $\Sigma = \{a, b\}$ where all states are initial and accepting, does $\mathcal{U}$ accept all (finite) words? This problem is PSPACE-complete [16, Lemma 6]. ◀

Allowing Confusion.

We say an observation policy *allows confusion* when, with positive probability, it produces an observation prefix $v\bot$ such that $v\bot$ is confused but v is not.

▶ **Proposition 9.** *A feasible observation policy does not allow confusion.*

Hence, in order to be feasible, a policy must observe when it would get confused otherwise. In § 5 we show that in the non-hidden case there is almost a converse of Proposition 9; i.e., in order to be feasible, a policy need not do much more than not allow confusion.

4 Analyzing the Cost of Decision

In this section we study the computational complexity of finding feasible policies that minimize the expected cost of decision. We focus on the decision version of the problem: *Is there a feasible policy whose expected cost is smaller than a given threshold?* Define:

$$c_{inf} \coloneqq \inf_{\text{feasible } \rho} \mathrm{Ex}(C_\rho)$$

Since the see-all policy $\bullet$ never stops observing, we have $\Pr(C_\bullet = \infty) = 1$, so $\mathrm{Ex}(C_\bullet) = \infty$. However, once an observation prefix v is deciding or very confused, there is no point in continuing observation. Hence, we define a *light see-all* policy $\circ$, which observes until the observation prefix u is deciding or very confused; formally, $\circ(v) = 0$ if and only if v is deciding

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang; Article No. 20; pp. 20:7–20:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

or very confused. It follows from the definition of very confused that the policy $\circ$ is feasible. Concerning the cost $C_\circ$ we have for all w

$$C_\circ(w) \;=\; \sum_{n=0}^{\infty} \big(1 - D_n(w)\big)\,, \tag{1}$$

where $D_n(w) = 1$ if the length-n prefix of w is deciding or very confused, and $D_n(w) = 0$ otherwise. The following results are proved in the full version of the paper, on arXiv:

▶ **Lemma 10.** *If ε is finitary then $\mathrm{Ex}(C_\circ)$ is finite.*

▶ **Lemma 11.** *Let ρ be a feasible observation policy. If $\Pr(C_\rho < \infty) = 1$ then ε is finitary.*

▶ **Proposition 12.** *c_{inf} is finite if and only if ε is finitary.*

▶ **Proposition 13.** *If a diagnoser exists then c_{inf} is finite.*

▶ **Theorem 14.** *It is PSPACE-complete to check if $c_{inf} < \infty$.*

Lemma 10 holds because, in $\mathcal{M} \times \mathcal{A}$, a bottom strongly connected component is reached in expected finite time. Lemma 11 says that a kind of converse holds for feasible policies. Proposition 12 follows from Lemmas 10 and 11. Proposition 13 follows from Propositions 7 and 12. To show Theorem 14, we use Proposition 12 and adapt the proof of Theorem 8.

The main negative result of the paper is that one cannot compute c_{inf}:

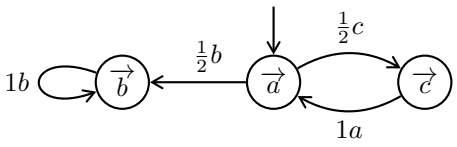
▶ **Theorem 15.** *It is undecidable to check if $c_{inf} < 3$, even when a diagnoser exists.*

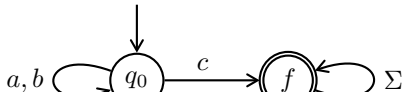
Proof sketch. By a reduction from the undecidable problem whether a given probabilistic automaton accepts some word with probability $> \frac{1}{2}$. The proof is somewhat complicated. In fact, in the full version of the paper (arXiv) we give two versions of the proof: a short incorrect one (with the correct main idea) and a long correct one. ◀

5 The Non-Hidden Case

Now we turn to positive results. In the rest of the paper we assume that the MC $\mathcal{M}$ is non-hidden, i.e., there exists a function $\overrightarrow{\cdot} : \Sigma \to S$ such that $M(a)(s, s') > 0$ implies $s' = \overrightarrow{a}$. We extend $\overrightarrow{\cdot}$ to finite words so that $\overrightarrow{ua} = \overrightarrow{a}$. We write $s \xrightarrow{u}$ to indicate that there is s' with $M(u)(s, s') > 0$.

▶ **Example 16.** Consider the following non-hidden MC and DFA:





$B_0 := \{(\overrightarrow{a}, q_0)\}$
$B_1 := \Delta(B_0, \bot) = \{(\overrightarrow{b}, q_0), (\overrightarrow{c}, f)\}$
$B_2 := \Delta(B_0, \bot^2) = \{(\overrightarrow{b}, q_0), (\overrightarrow{a}, f)\}$
$B_3 := \Delta(B_0, \bot^2 b) = \{(\overrightarrow{b}, q_0), (\overrightarrow{b}, f)\}$

B_0 is the initial belief. The beliefs B_0 and B_1 are not confused: indeed, $\Delta(B_1, b) = \{(\overrightarrow{b}, q_0)\}$ is negatively deciding, and $\Delta(B_1, a) = \{(\overrightarrow{a}, f)\}$ is positively deciding. The belief B_2 is confused, as there is no $i \in \mathbb{N}$ for which $\Delta(B_2, b^i)$ is deciding. Finally, B_3 is very confused. ◀

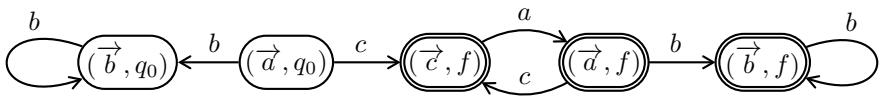
We will show that in the non-hidden case there always exists a diagnoser (Lemma 23). It follows that feasible policies need to decide almost surely and, by Proposition 13, that c_{inf} is finite. We have seen in Proposition 9 that feasible policies do not allow confusion. In this section we construct policies that procrastinate so much that they avoid confusion just barely. We will see that such policies have an expected cost that comes arbitrarily close to c_{inf}.

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang; Article No. 20; pp. 20:8–20:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Language Equivalence.

We characterize confusion by language equivalence in a certain DFA. Consider the belief NFA $\mathcal{B}$. In the non-hidden case, if we disallow $\bot$-transitions then $\mathcal{B}$ becomes a DFA $\mathcal{B}'$. For $\mathcal{B}'$ we define a set of accepting states by $F_{\mathcal{B}'} := \{(s,q) \mid \Pr_s(L_q) = 1\}$.

▶ **Example 17.** For the previous example, a part of the DFA $\mathcal{B}'$ looks as follows:



States that are unreachable from $(\overrightarrow{a}, q_0)$ are not drawn here. ◀

We associate with each (s,q) the language $L_{s,q} \subseteq \Sigma^*$ that $\mathcal{B}'$ accepts starting from initial state (s,q). We call $(s,q), (s',q')$ *language equivalent*, denoted by $(s,q) \approx (s',q')$, when $L_{s,q} = L_{s',q'}$.

▶ **Lemma 18.** *One can compute the relation $\approx$ in polynomial time.*

Proof. For any (s,q) one can use standard MC algorithms to check in polynomial time if $\Pr_s(L_q) = 1$ (using a graph search in the composition $\mathcal{M} \times \mathcal{A}$, as in the proof of Lemma 5.3). Language equivalence in the DFA $\mathcal{B}'$ can be computed in polynomial time by minimization. ◀

We call a belief $B \subseteq S \times Q$ *settled* when all $(s,q) \in B$ are language equivalent.

▶ **Lemma 19.** *A belief $B \subseteq S \times Q$ is confused if and only if there is $a \in \Sigma$ such that $\Delta(B,a)$ is not settled.*

It follows that one can check in polynomial time whether a given belief is confused. We generalize this fact in Lemma 22 below.

▶ **Example 20.** In Example 16 the belief B_3 is not settled. Indeed, from the DFA in Example 17 we see that $L_{\overrightarrow{b},q_0} = \emptyset \neq \{b\}^* = L_{\overrightarrow{b},f}$. Since $B_3 = \Delta(B_2, b)$, by Lemma 19, the belief B_2 is confused. ◀

Procrastination.

For a belief $B \subseteq S \times Q$ and $k \in \mathbb{N}$, if $\Delta(B, \bot^k)$ is confused then so is $\Delta(B, \bot^{k+1})$. We define:

$$cras(B) \;:=\; \sup\{\, k \in \mathbb{N} \mid \Delta(B, \bot^k) \text{ is not confused} \,\} \;\in\; \mathbb{N} \cup \{-1, \infty\}$$

We set $cras(B) := -1$ if B is confused. We may write $cras(s,q)$ for $cras(\{(s,q)\})$.

▶ **Example 21.** In Example 16 we have $cras(B_0) = cras(\overrightarrow{a}, q_0) = 1$ and $cras(B_1) = 0$ and $cras(B_2) = cras(B_3) = -1$ and $cras(\overrightarrow{b}, q_0) = cras(\overrightarrow{a}, f) = \infty$. ◀

▶ **Lemma 22.** *Given a belief B, one can compute $cras(B)$ in polynomial time. Further, if $cras(B)$ is finite then $cras(B) < |S|^2 \cdot |Q|^2$.*

Proof. Let $k \in \mathbb{N}$. By Lemma 19, $\Delta(B, \bot^k)$ is confused if and only if:

$$\exists\, a.\, \exists\, (s,q), (t,r) \in \Delta(B, \bot^k) : s \xrightarrow{a},\; t \xrightarrow{a},\; (\overrightarrow{a}, \delta(q,a)) \not\approx (\overrightarrow{a}, \delta(r,a))$$

This holds if and only if there is $B_2 \subseteq B$ with $|B_2| \leq 2$ such that:

$$\exists\, a.\, \exists\, (s,q), (t,r) \in \Delta(B_2, \bot^k) : s \xrightarrow{a},\; t \xrightarrow{a},\; (\overrightarrow{a}, \delta(q,a)) \not\approx (\overrightarrow{a}, \delta(r,a))$$

Let G be the directed graph with nodes in $S \times Q \times S \times Q$ and edges

$$((s,q,t,r),(s',q',t',r')) \quad \Longleftrightarrow \quad \Delta(\{(s,q),(t,r)\},\bot) \supseteq \{(s',q'),(t',r')\}\,.$$

Also define the following set of nodes:

$$U \;:=\; \{(s,q,t,r) \mid \exists\, a : s \xrightarrow{a},\ t \xrightarrow{a},\ (\overrightarrow{a},\delta(q,a)) \not\approx (\overrightarrow{a},\delta(r,a))\}$$

By Lemma 18 one can compute U in polynomial time. It follows from the argument above that $\Delta(B,\bot^k)$ is confused if and only if there are $(s,q),(t,r) \in B$ such that there is a length-k path in G from (s,q,t,r) to a node in U. Let $k \le |S \times Q \times S \times Q|$ be the length of the shortest such path, and set $k := \infty$ if no such path exists. Then k can be computed in polynomial time by a search of the graph G, and we have $cras(B) = k - 1$. ◀

The Procrastination Policy.

For any belief B and any observation prefix v, the language equivalence classes represented in $\Delta(B,v)$ depend only on v and the language equivalence classes in B. Therefore, when tracking beliefs along observations, we may restrict B to a single representative of each equivalence class. We denote this operation by $B{\downarrow}$. A belief B is settled if and only if $|B{\downarrow}| \le 1$.

A *procrastination policy* $\rho_{pro}(K)$ is parameterized with (a large) $K \in \mathbb{N}$. Define (and precompute) $k(s,q) := \min\{K, cras(s,q)\}$ for all (s,q). We define $\rho_{pro}(K)$ by the following monitor that implements it:

1. $i := 0$
2. while (s_i,q_i) is not deciding:
   a. skip $k(s_i,q_i)$ observations, then observe a letter a_i
   b. $\{(s_{i+1},q_{i+1})\} := \Delta((s_i,q_i),\bot^{k(s_i,q_i)}a_i){\downarrow}$;
   c. $i := i+1$;
3. output yes/no decision

It follows from the definition of *cras* and Lemma 19 that $\Delta((s_i,q_i),v_i){\downarrow}$ is indeed a singleton for all i. We have:

▶ **Lemma 23.** *For all $K \in \mathbb{N}$ the procrastination policy $\rho_{pro}(K)$ is a diagnoser.*

Proof. For a non-hidden MC $\mathcal{M}$ and a DFA $\mathcal{A}$, there is at most one successor for (s,q) on letter a in the belief NFA $\mathcal{B}$, for all s,q,a. Then, by Lemma 19, singleton beliefs are not confused, and in particular the initial belief B_0 is not confused. By Lemma 4.4, ε is not confused, which means that $\Pr(\{\,u \mid u \text{ deciding}\,\}) = \Pr(\varepsilon) = 1$. Since almost surely a deciding word u is produced and since $\Delta(B_0,u) \subseteq \Delta(B_0,v)$ whenever $u \sim v$, it follows that eventually an observation prefix v is produced such that $\Delta(B_0,v)$ contains a deciding pair (s,q). But, as remarked above, $\Delta(B_0,v)$ is settled, so it is deciding. ◀

The Procrastination MC $\mathcal{M}_{pro}(K)$.

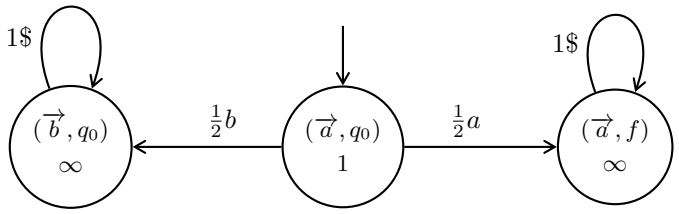
The policy $\rho_{pro}(K)$ produces a (random, almost surely finite) word $a_1a_2\cdots a_n$ with $n = C_{\rho_{pro}(K)}$. Indeed, the observations that $\rho_{pro}(K)$ makes can be described by an MC. Recall that we have previously defined a composition MC $\mathcal{M} \times \mathcal{A} = (S \times Q, \Sigma, M', (s_0,q_0))$. Now define an MC $\mathcal{M}_{pro}(K) := (S \times Q, \Sigma \cup \{\$\}, M_{pro}(K), (s_0,q_0))$ where $\$ \notin \Sigma$ is a fresh letter

and the transitions are as follows: when (s,q) is deciding then $M_{pro}(K)(\$)\big((s,q),(s,q)\big) \coloneqq 1$, and when (s,q) is not deciding then

$$M_{pro}(K)(a)\big((s,q),(\overrightarrow{a},q')\big) \;\coloneqq\; \Big(M'(\bot)^{k(s,q)}M'(a)\Big)\big((s,q),(\overrightarrow{a},q')\big)\,,$$

where the matrix $M'(\bot) \coloneqq \sum_a M'(a)$ is powered by $k(s,q)$. The MC $\mathcal{M}_{pro}(K)$ may not be non-hidden, but could be made non-hidden by (i) collapsing all language equivalent $(s,q_1),(s,q_2)$ in the natural way, and (ii) redirecting all \$-labelled transition to a new state $\overrightarrow{\$}$ that has a self-loop. In the understanding that $\$\$\$\cdots$ indicates 'decision made', the probability distribution defined by the MC $\mathcal{M}_{pro}(K)$ coincides with the probability distribution on sequences of non-$\bot$ observations made by $\rho_{pro}(K)$.

▶ **Example 24.** For Example 16 the MC $\mathcal{M}_{pro}(K)$ for $K \geq 1$ is as follows:



Here the lower number in a state indicate the *cras* number. The left state is negatively deciding, and the right state is positively deciding. The policy $\rho_{pro}(K)$ skips the first observation and then observes either b or a, each with probability $\frac{1}{2}$, each leading to a deciding belief. ◀

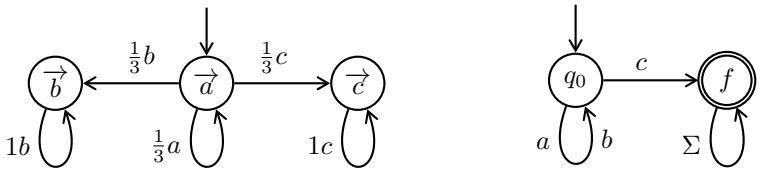
Maximal Procrastination is Optimal.

The following lemma states, loosely speaking, that when a belief $\{(s,q)\}$ with $cras(s,q) = \infty$ is reached and K is large, then a single further observation is expected to suffice for a decision.

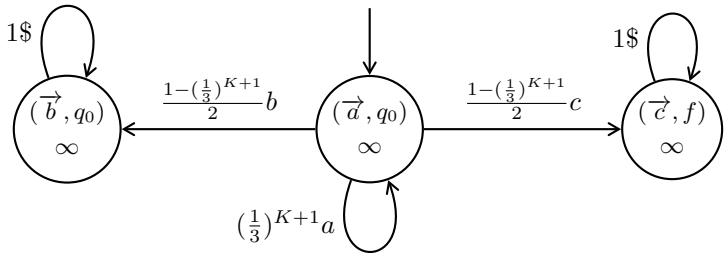
▶ **Lemma 25.** *Let $c(K,s,q)$ denote the expected cost of decision under $\rho_{pro}(K)$ starting in (s,q). For each $\varepsilon > 0$ there exists $K \in \mathbb{N}$ such that for all (s,q) with $cras(s,q) = \infty$ we have $c(K,s,q) \leq 1+\varepsilon$.*

Proof sketch. The proof is a quantitative version of the proof of Lemma 23. The singleton belief $\{(s,q)\}$ is not confused. Thus, if K is large then with high probability the belief $B \coloneqq \Delta(\{(s,q)\}, \bot^K a)$ (for the observed next letter a) contains a deciding pair (s',q'). But if $cras(s,q) = \infty$ then, by Lemma 19, B is settled, so if B contains a deciding pair then B is deciding. ◀

▶ **Example 26.** Consider the following variant of the previous example:



The MC $\mathcal{M}_{pro}(K)$ for $K \geq 0$ is as follows:



The left state is negatively deciding, and the right state is positively deciding. We have $c(K, \overrightarrow{b}, q_0) = c(K, \overrightarrow{c}, f) = 0$ and $c(K, \overrightarrow{a}, q_0) = 1/(1 - (\frac{1}{3})^{K+1})$. ◀

Now we can prove the main positive result of the paper:

▶ **Theorem 27.** *For any feasible policy ρ there is $K \in \mathbb{N}$ such that:*

$$\mathrm{Ex}(C_{\rho_{pro}(K)}) \;\leq\; \mathrm{Ex}(C_\rho)$$

Proof sketch. Let ρ be a feasible policy. We choose $K > |S|^2 \cdot |Q|^2$, so, by Lemma 22, $\rho_{pro}(K)$ coincides with $\rho_{pro}(\infty)$ until time, say, n_∞ when $\rho_{pro}(K)$ encounters a pair (s, q) with $cras(s, q) = \infty$. (The time n_∞ may, with positive probability, never come.) Let us compare $\rho_{pro}(K)$ with ρ up to time n_∞. For $n \in \{0, \ldots, n_\infty\}$, define $\upsilon_{pro}(n)$ and $\upsilon_\rho(n)$ as the observation prefixes obtained by ρ_{pro} and ρ, respectively, after n steps. Write $\ell_{pro}(n)$ and $\ell_\rho(n)$ for the number of non-$\perp$ observations in $\upsilon_{pro}(n)$ and $\upsilon_\rho(n)$, respectively. For beliefs B, B' we write $B \preceq B'$ when for all $(s, q) \in B$ there is $(s', q') \in B'$ with $(s, q) \approx (s', q')$. One can show by induction that we have for all $n \in \{0, \ldots, n_\infty\}$:

$$\ell_{pro}(n) \leq \ell_\rho(n) \quad \text{and} \quad \big(\Delta(B_0, \upsilon_{pro}(n)) \preceq \Delta(B_0, \upsilon_\rho(n)) \quad \text{or} \quad \ell_{pro}(n) < \ell_\rho(n)\big)$$

If time n_∞ does not come then the inequality $\ell_{pro}(n) \leq \ell_\rho(n)$ from above suffices. Similarly, if at time n_∞ the pair (s, q) is deciding, we are also done. If after time n_∞ the procrastination policy $\rho_{pro}(K)$ observes at least one more letter then ρ also observes at least one more letter. By Lemma 25, one can choose K large so that for $\rho_{pro}(K)$ one additional observation probably suffices. If it is the case that ρ almost surely observes only one letter after n_∞, then $\rho_{pro}(K)$ also needs only one more observation, since it has observed at time n_∞. ◀

It follows that, in order to compute c_{inf}, it suffices to analyze $\mathrm{Ex}(C_{\rho_{pro}(K)})$ for large K. This leads to the following theorem:

▶ **Theorem 28.** *Given a non-hidden MC $\mathcal{M}$ and a DFA $\mathcal{A}$, one can compute c_{inf} in polynomial time.*

Proof. For each (s, q) define $c(K, s, q)$ as in Lemma 25, and define $c(s, q) := \lim_{K \to \infty} c(K, s, q)$. By Lemma 25, for each non-deciding (s, q) with $cras(s, q) = \infty$ we have $c(s, q) = 1$. Hence the $c(s, q)$ satisfy the following system of linear equations where some coefficients come from the procrastination MC $\mathcal{M}_{pro}(\infty)$:

$$c(s,q) \;=\; \begin{cases} 0 & \text{if } (s,q) \text{ is deciding} \\ 1 & \text{if } (s,q) \text{ is not deciding and } cras(s,q) = \infty \\ 1 + c'(s,q) & \text{otherwise} \end{cases}$$

$$c'(s,q) \;=\; \sum_a \sum_{q'} M_{pro}(\infty)\big((s,q), (\overrightarrow{a}, q')\big) \cdot c(\overrightarrow{a}, q') \qquad \text{if } cras(s,q) < \infty$$

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang; Article No. 20; pp. 20:12–20:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

By solving the system one can compute $c(s_0, q_0)$ in polynomial time. We have:

$$c_{inf} = \inf_{\text{feasible } \rho} \mathrm{Ex}(C_\rho) \overset{\text{Thm}27}{=} \lim_{K \to \infty} \mathrm{Ex}(C_{\rho_{pro}(K)}) = c(s_0, q_0)$$

Hence one can compute c_{inf} in polynomial time. ◀

6 Empirical Evaluation of the Expected Optimal Cost

We have shown that maximal procrastination is optimal in the non-hidden case (Theorem 27). However, we have not shown *how much* better the optimal policy is than the see-all baseline. It appears difficult to answer this question analytically, so we address it empirically. We implemented our algorithms in a fork of the Facebook Infer static analyzer [8], and applied them to 11 open-source projects, totaling 80 thousand Java methods. We found that in $> 90\%$ of cases the maximally procrastinating monitor is trivial and thus the optimal cost is 0, because Infer decides statically if the property is violated. In the remaining cases, we found that the optimal cost is roughly half of the see-all cost, but the variance is high.

Design. Our setting requires a DFA and an MC representing, respectively, a program property and a program. For this empirical estimation of the expected optimal cost, the DFA is fixed, the MC shape is the symbolic flowgraph of a real program, and the MC probabilities are sampled from Dirichlet distributions.

The DFA represents the following property: 'there are no two calls to *next* without an intervening call to *hasNext*'. To understand how the MC shape is extracted from programs, some background is needed. Infer [8, 9] is a static analyzer that, for each method, infers several preconditions and, attached to each precondition, a symbolic path. For a simple example, consider a method whose body is '$\mathsf{if}\ (b)\ x.next();\ \mathsf{if}\ (!b)\ x.next()$'. Infer would generate two preconditions for it, b and $\neg b$. In each of the two attached symbolic paths, we can see that *next* is not called twice, which we would not notice with a control flowgraph. The symbolic paths are inter-procedural. If a method f calls a method g, then the path of f will link to a path of g and, moreover, it will pick one of the paths of g that corresponds to what is currently known at the call site. For example, if $g(b)$ is called from a state in which $\neg b$ holds, then Infer will select a path of g compatible with the condition $\neg b$.

The symbolic paths are finite because abstraction is applied, including across mutually recursive calls. But, still, multiple vertices of the symbolic path correspond to the same vertex of the control flowgraph. For example, Infer may go around a for-loop five times before noticing the invariant. By coalescing those vertices of the symbolic path that correspond to the same vertex of the control flowgraph we obtain an *SFG (symbolic flowgraph)*. We use such SFGs as the skeleton of MCs. Intuitively, one can think of SFGs as inter-procedural control flowgraphs restricted based on semantic information. Vertices correspond to locations in the program text, and transitions correspond to method calls or returns. Transition probabilities should then be interpreted as a form of static branch prediction. One could learn these probabilities by observing many runs of the program on typical input data, for example by using the Baum–Welch algorithm [17]. Instead, we opt to show that the improvement in expected observation cost is robust over a wide range of possible transition probabilities, which we do by drawing several samples from Dirichlet distributions. Besides, recall that the (optimal) procrastination policy does not depend on transition probabilities.

Once we have a DFA and an MC we compute their product. In some cases, it is clear that the product is empty or universal. These are the cases in which we can give the verdict right away, because no observation is necessary. We then focus on the non-trivial cases.

For non-trivial MC × DFA products, we compute the expected cost of the light see-all policy $\mathrm{Ex}(C_\circ)$, which observes all letters until a decision is made and then stops. We can do so by using standard algorithms [2, Chapter 10.5]. Then, we compute $\mathcal{M}_{pro}$, which we use to compute the expected observation cost c_{inf} of the procrastination policy (Theorem 28). Recall that in order to compute $\mathcal{M}_{pro}$, one needs to compute the *cras* function, and also to find language equivalence classes. Thus, computing $\mathcal{M}_{pro}$ entails computing all the information necessary for implementing a procrastinating monitor.

Methodology. We selected 11 Java projects among those that are most forked on GitHub. We ran Infer on each of these projects. From the inferred specifications, we built SFGs and monitors that employ light see-all policies and maximal procrastination policies. From these monitors, we computed the respective expected costs, solving the linear systems using Gurobi [12]. Our implementation is in a fork of Infer, on GitHub.

Table 1 Reduction in expected observation cost, on real-world data. Each SFG (symbolic flowgraph) corresponds to one inferred precondition of a method. The size of monitors is measured in number of language equivalence classes. (LOC = lines of code; GAvg = geometric average.)

	Project Size			Monitors			$c_{inf}/\mathrm{Ex}(C_\circ)$	
Name	Methods	SFGs	LOC	Count	Avg-Size	Max-Size	Med	GAvg
tomcat	26K	52K	946K	343	69	304	0.53	0.50
okhttp	3K	6K	49K	110	263	842	0.46	0.42
dubbo	8K	16K	176K	91	111	385	0.53	0.51
jadx	4K	9K	48K	204	96	615	0.58	0.50
RxJava	12K	45K	192K	83	41	285	0.52	0.53
guava	22K	43K	1218K	1126	134	926	0.41	0.41
clojure	5K	19K	66K	219	120	767	0.44	0.44
AndroidUtilCode	3K	7K	436K	39	89	288	0.66	0.58
leakcanary	1K	1K	11K	12	79	268	0.66	0.59
deeplearning4j	21K	40K	408K	262	51	341	0.58	0.58
fastjson	2K	7K	47K	204	63	597	0.59	0.53

Results. The results are given in Table 1. We first note that the number of monitors is much smaller than the number of methods, by a factor of 10 or 100. This is because in most cases we are able to determine the answer statically, by analyzing the symbolic paths produced by Infer. The large factor should not be too surprising: we are considering a fixed property about iterators, not all Java methods use iterators, and, when they do, it is usually easy to tell that they do so correctly. Still, each project has a few hundred monitors, which handle the cases that are not so obvious.

We note that $\frac{c_{inf}}{\mathrm{Ex}(C_\circ)} \approx 0.5$. The table supports this by presenting the median and the geometric average, which are close to each-other; the arithmetic average is also close. There is, however, quite a bit of variation from monitor to monitor, as shown in Figure 1. We conclude that selective monitoring has the potential to significantly reduce the overhead of runtime monitoring.

7 Future Work

In this paper we required policies to be feasible, which means that our selective monitors are as precise as non-selective monitors. One may relax this and study the tradeoff between efficiency (skipping even more observations) and precision (probability of making a decision).

29th International Conference on Concurrency Theory (CONCUR 2018).
Editors: Sven Schewe and Lijun Zhang; Article No. 20; pp. 20:14–20:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

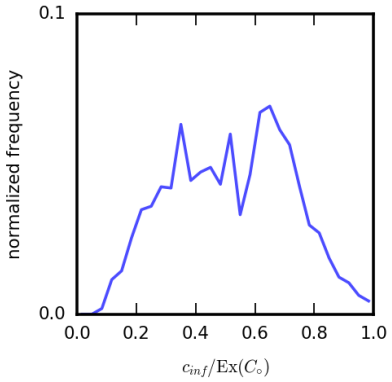


Figure 1 Empirical distribution of $c_{inf}/\mathrm{Ex}(C_\circ)$, across all projects.

Further, one could replace the diagnosability notion of this paper by other notions from the literature; one could investigate how to compute c_{inf} for other classes of MCs, such as acyclic MCs; one could study the sensitivity of c_{inf} to changes in transition probabilities; and one could identify classes of MCs for which selective monitoring helps and classes of MCs for which selective monitoring does not help.

A nontrivial extension to the formal model would be to include some notion of data, which is pervasive in practical specification languages used in runtime verification [13]. This would entail replacing the DFA with a more expressive device, such as a nominal automaton [7], a symbolic automaton [10], or a logic with data (e.g., [11]). Alternatively, one could side-step the problem by using the slicing idea [18], which separates the concern of handling data at the expense of a mild loss of expressive power. Finally, the monitors we computed could be used in a runtime verifier, or even in session type monitoring where the setting is similar [6].

References

1 Matthew Arnold, Martin T. Vechev, and Eran Yahav. QVM: an efficient runtime for detecting defects in deployed systems. In *OOPSLA*, 2008.

2 C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

3 Ezio Bartocci, Radu Grosu, Atul Karmarkar, Scott A. Smolka, Scott D. Stoller, Erez Zadok, and Justin Seyster. Adaptive runtime verification. In *RV*, 2012.

4 N. Bertrand, S. Haddad, and E. Lefaucheux. Foundation of diagnosis and predictability in probabilistic systems. In *Proceedings of FSTTCS*, volume 29 of *LIPIcs*, pages 417–429, 2014.

5 N. Bertrand, S. Haddad, and E. Lefaucheux. Accurate approximate diagnosability of stochastic systems. In *Proceedings of LATA*, pages 549–561. Springer, 2016.

6 Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. *TCS*, 2017.

7 Mikołaj Bojánczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *LMCS*, 2014.

8 C. Calcagno, D. Distefano, J. Dubreil, D. Gabi, P. Hooimeijer, M. Luca, P. O'Hearn, I. Papakonstantinou, J. Purbrick, and D. Rodriguez. Moving fast with software verification. In *NASA Formal Methods Symposium*, 2015.

9 C. Calcagno, D. Distefano, P.W. O'Hearn, and H. Yang. Compositional shape analysis by means of bi-abduction. *JACM*, 2011.

10 Loris D'Antoni and Margus Veanes. The power of symbolic automata and transducers. In *CAV*, 2017.

11 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *TOCL*, 2009.

12 Gurobi Optimization, Inc. Gurobi optimizer reference manual. http://www.gurobi.com, 2017.

13 Klaus Havelund, Martin Leucker, Giles Reger, and Volker Stolz. A Shared Challenge in Behavioural Specification (Dagstuhl Seminar 17462). *Dagstuhl Reports*, 2018. doi:10.4230/DagRep.7.11.59.

14 S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321, 2001.

15 K. Kalajdzic, E. Bartocci, S.A. Smolka, S.D. Stoller, and R. Grosu. Runtime verification with particle filtering. In *RV*, 2013.

16 J.-Y. Kao, N. Rampersad, and J. Shallit. On NFAs where all states are final, initial, or both. *Theoretical Computer Science*, 410(47):5010–5021, 2009.

17 Brian G. Leroux. Maximum-likelihood estimation for hidden markov models. *Stochastic Processes and Their Applications*, 1992.

18 Grigore Roşu and Feng Chen. Semantics and algorithms for parametric monitoring. *LMCS*, 2012.

19 M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

20 A. Prasad Sistla, Miloš Žefran, and Yao Feng. Monitorability of stochastic dynamical systems. In *Proceedings of CAV*, pages 720–736. Springer, 2011.

21 S.D. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S.A. Smolka, and E. Zadok. Runtime verification with state estimation. In *RV*, 2011.

22 D. Thorsley and D. Teneketzis. Diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 50(4):476–492, 2005.