Lazy Regular Sensing*

Orna Kupferman[1] and Asaf Petruschka[2]

[1] School of Engineering and Computer Science, The Hebrew University, Israel.
[2] Dept. of Math. and Computer Science, The Weizmann Institute of Science, Israel.
E-mails: orna@cs.huji.ac.il; asaf.petruschka@weizmann.ac.il

Abstract. A complexity measure for regular languages based on the *sensing* required to recognize them was recently introduced by Almagor, Kuperberg, and Kupferman. Intuitively, the sensing cost quantifies the detail in which a random input word has to be read in order to decide its membership in the language, when the input letters composing the word are truth assignments to a finite set of *signals*. We introduce the notion of *lazy sensing*, where the signals are not sensed simultaneously. Rather, the signals are ordered, and a signal is sensed only if the values of the signals sensed so far have not determined the successor state. We study four classes of lazy sensing, induced by distinguishing between the cases where the order of the signals is static or dynamic (that is, fixed in advance or depends on the values of the signals sensed so far), and the cases where the order is global or local (that is, the same for all states of the automaton, or not). We examine the different classes of lazy sensing and the saving they enable, with respect to each other and with respect to (non-lazy) sensing. We also examine the trade offs between sensing cost and size. Our results show that the good properties of sensing are preserved in the lazy setting. In particular, saving sensing does not conflict with saving size: in all four classes, the lazy-sensing cost of a regular language can be attained in the minimal automaton recognizing the language.

1 Introduction

The classical complexity measure for regular languages is the size of a minimal deterministic automaton that recognizes the language. In [1], the authors introduced a new complexity measure, namely the *sensing cost* of the language. Intuitively, the sensing cost of a language measures the detail with which a random input word needs to be read in order to decide membership in the language. The study is motivated by the use of finite-state automata in reasoning about on-going behaviors of reactive systems. In particular, when *monitoring* a computation, we seek a monitor that minimizes the activation of sensors used in the monitoring process, and when *synthesizing* a system, we prefer I/O-transducers that satisfy a given specification while minimizing the activation of sensors (of

* Work partially supported by the Israel Science Foundation, ISF grant agreement no 2357/19.

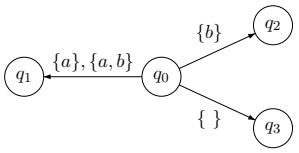
input signals) [1]. Sensing has been studied in several other computer-science contexts. In theoretical computer science, in methodologies such as PCP and property testing, we are allowed to sample or query only part of the input [5]. In more practical applications, mathematical tools in signal processing are used to reconstruct information based on compressed sensing [3], and in the context of data streaming, one cannot store in memory the entire input, and therefore has to approximate its properties according to partial "sketches" [8].

The automata used in formal methods are over alphabets of the form 2^P, for a finite set P of signals. Consider a deterministic automaton (DFA) $\mathcal{A}$ over an alphabet 2^P. For a state q of $\mathcal{A}$, we say that a signal $p \in P$ is *sensed* in q if at least one transition taken from q depends on the truth value of p. The *sensing cost* of q is the number of signals it senses, and the sensing cost of a run is the average sensing cost of states visited along the run. The definition is extended to DFAs by defining the sensing cost of $\mathcal{A}$ as the limit of the expected sensing of runs over words of increasing length, assuming a uniform distribution of the letters in 2^P, thus each signal $p \in P$ holds in each moment in time with probability $\frac{1}{2}$. It is easy to extend the setting to a non-uniform distribution on the letters, given by a Markov chain. The sensing cost of a language $L \subseteq (2^P)^*$ is then the infimum of the sensing costs of DFAs for L.

In this work, we refine the notion of regular sensing from [1], which we call *naive sensing*, to a new notion called *lazy sensing*. Intuitively, in naive sensing, the signals in P are sensed simultaneously. Consequently, if a signal p is defined to be sensed in a state q, then a sensor for p must indeed be activated whenever a run of the DFA is in state q and needs to determine the successor state. In lazy sensing, the signals are not sensed simultaneously. Instead, they can be activated "on demand", one after the other, and we may reach a decision about the successor state before they are all sensed. This is demonstrated in the following simple example.

Example 1. Let $P = \{a, b\}$, and consider a state q_0 with three successor states q_1, q_2, and q_3, and transitions as shown on the right. According to the definition in [1], both a and b are sensed in q_0. Indeed, in naive sensing, when the signals are sensed simultaneously, both a and b must be sensed in order to determine the successor state.

In lazy sensing, we can start by sensing only the signal a. If a is `True`, then we know that the successor state is q_1, and there is no need to sense b. Accordingly, if we assume that a has probability $\frac{1}{2}$ to be `True`, the number of sensors we are expected to activate in state q_0 is only $1\frac{1}{2}$, rather than 2. □



The underlying idea of lazy sensing is simple and is similar to *short-circuit evaluation* in programming languages. There, the second argument of a Boolean operator is executed or evaluated only if the first argument does not suffice to determine the value of the expression [7]. Our study examines such a lazy evaluation in the context of DFAs.

In order to perform lazy sensing, each state of the DFA should be equipped with a data structure that directs it which signal to sense next. We examine four different classes of lazy sensing, induced by the following two parameters: (1) Is ordering of signals sensed *dynamic* or *static*: in the dynamic classes, the order may depend on the truth value of signals sensed earlier. That is, the data structure supports policies like "if a is `True`, then next sense b, and if a is `False`, then next sense c". In the static classes, the data structure is a linear order on the signals – the order in which they are going to be sensed, regardless of the result. (2) Is the sensing policy *local* or *global*: in the local classes, each state may have its own data structure. In the global ones, the same data structure is used for all the states. Note that both parameters are irrelevant in short-circuit evaluation in programming languages. Indeed, lazy evaluation concerns Boolean expressions, each evaluated independently, and the control flow is induced by the structure of the expression.

The difference between the dynamic and static classes can be viewed as follows. Consider a DFA with state space Q, and consider a state $q \in Q$. The data structure maintaining the transitions from q is a *sensing tree*: a decision tree in which each vertex is labeled by a signal in $p \in P$ and has two successors, corresponding to the two truth values that p may have. Each path in the tree corresponds to a set of assignments to the signals in P – assignments that are consistent with the truth values that the path assigns to signals that appear in it. Accordingly, if we label the leaves of the tree by states in Q, then each sensing tree maintains a function $f : 2^P \to Q$. In the static classes, all paths in the sensing tree follow the same fixed order of the signals in P. Thus, the sensing tree is related to a *multiple-valued decision diagram* [2,6]. On the other hand, in the dynamic classes, the order of the signals in each path of the sensing tree may be different.

For all the four classes, the lazy sensing cost of a state $q \in Q$ is the expected number of signals sensed when a transition from q is taken and sensing is performed according to the sensing tree. Then, the lazy sensing cost of a DFA is the limit of expected sensing of runs overs words of increasing length, with the best possible choice of the allowed data structure. For example, in the static-global class, this best possible choice is a single vector of the signals in P, maintaining a linear order that is used by all states of the DFA. Finally, the sensing cost of a regular language L is the infimum of sensing costs of a DFA for L.

We examine the different classes of lazy sensing and the saving they enable, with respect to each other and with respect to naive sensing. We also examine the trade offs between sensing cost and size. Our results show that the good properties of naive sensing are preserved in lazy sensing. In particular, the lazy sensing cost of a DFA can be calculated by using the stationary distribution of its induced Markov chain. Also, saving sensing does not conflict with saving size: in all four classes, the lazy sensing cost of a regular language can be attained in the minimal automaton recognizing the language.

Due to the lack of space, some proofs are omitted and can be found in the full version, in the authors' URLs.

2 Defining Lazy Sensing

2.1 Deterministic Finite Automata

A *deterministic automaton on finite words* (DFA, for short) is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q$ is a transition function, and $\alpha \subseteq Q$ is a set of accepting states. A run of $\mathcal{A}$ on a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_m \in \Sigma^*$ is the sequence of states $q_0, q_1, \ldots, q_m$ such that $q_{i+1} = \delta(q_i, \sigma_{i+1})$ for all $i \geq 0$. The run is accepting if $q_m \in \alpha$. A word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if the run of $\mathcal{A}$ on w is accepting. For $i \geq 0$, we use $w[1, i]$ to denote the prefix $\sigma_1 \cdot \sigma_2 \cdots \sigma_i$ of w, and use $\delta(w[1, i])$ to denote the state q_i that $\mathcal{A}$ visits after reading the prefix $w[1, i]$. Note that $w[1, 0] = \epsilon$. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. For a state $q \in Q$, we use $\mathcal{A}^q$ to denote $\mathcal{A}$ with initial state q.

2.2 Potentially Sensed Signals

We study languages over an alphabet $\Sigma = 2^P$, for a finite set P of signals. A letter $\sigma \in \Sigma$ corresponds to a truth assignment to the signals. When we define languages over Σ, we use predicates on P in order to denote sets of letters. For example, if $P = \{a, b, c\}$, then the expression $(\texttt{True})^* \cdot a \cdot b \cdot (\texttt{True})^*$ describes all words over 2^P that contain a subword $\sigma_a \cdot \sigma_b$ with $\sigma_a \in \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$ and $\sigma_b \in \{\{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}\}$.

Consider a DFA $\mathcal{A} = \langle 2^P, Q, q_0, \delta, \alpha \rangle$. For a state $q \in Q$ and a signal $p \in P$, we say that p is *potentially sensed in* q if there exists a set $S \subseteq P$ such that $\delta(q, S \setminus \{p\}) \neq \delta(q, S \cup \{p\})$. Intuitively, a signal is potentially sensed in q if knowing its value may affect the destination of at least one transition from q. We use $psensed(q)$ to denote the set of signals potentially sensed in q.

Recall the situation in Example 1. For $S = \emptyset$, we have $\delta(q_0, S \cup \{a\}) = q_1$ and $\delta(q_0, S \setminus \{a\}) = q_3$, so a is potentially sensed in q_0. Also, $\delta(q_0, S \cup \{b\}) = q_2$ and $\delta(q_0, S \setminus \{b\}) = q_3$, so b is also potentially sensed in q_0.

In the naive sensing setting, studied in [1], sensing of input signals happens simultaneously; that is, we sense together all of the signals whose truth value might affect the decision to which state to proceed. Accordingly, the notions of a sensed signal in [1] and our definition above of a potentially sensed signal coincide. In the following sections, we formalize the notion of *lazy sensing*, where sensing need not be simultaneous.

2.3 Sensing Trees

The main feature of lazy sensing is a data structure termed *sensing tree*, which directs the order in which signals are sensed. A *sensing tree* is a labeled tree $T = \langle V, E, \tau \rangle$, where V is a set of vertices, $E \subseteq V \times \{\texttt{True}, \texttt{False}\} \times V$ is a set of directed labeled edges, and $\tau : V \to P \cup Q$ is a labelling function. Each vertex $v \in V$ is either *internal*, in which case it has exactly two children, v_{left} and v_{right}, with $\langle v, \texttt{False}, v_{left} \rangle$ and $\langle v, \texttt{True}, v_{right} \rangle$, or is a *leaf*, in which case

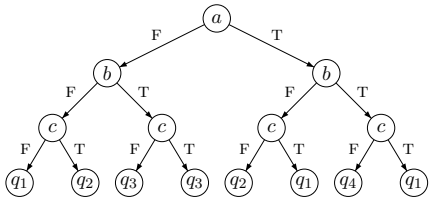


Fig. 1: The sensing tree T.

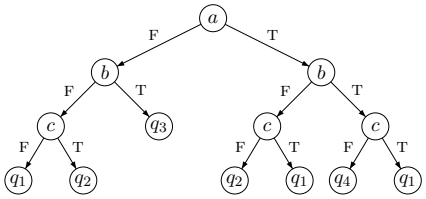


Fig. 2: Reducing the tree T.

it has no children. Let $Int(T)$ and $Leaves(T)$ denote the sets of internal vertices and leaves of T, respectively. We assume that T has a single *root* – a vertex with no incoming edges.

The labelling function τ labels internal vertices by signals in P and labels leaves by states in Q. The function τ is such that for each signal $p \in P$ and leaf $\ell \in V$, the single path from the root to ℓ includes at most one vertex labeled p. Accordingly, each subset $S \in 2^P$ corresponds to a single leaf, namely the leaf reached by following the path that corresponds to the assignment S. Formally, reading an input $S \in 2^P$, we start from the root of the tree, and then in each step, we sense the signal p that labels the current vertex. If it is `True` (i.e., $p \in S$), we proceed to the right child. If it is `False` (i.e., $p \notin S$), we proceed to the left child. By the requirement on τ, we encounter each signal at most once. In particular, as some signals may not appear in the traversed path, the above process may reach a leaf before all signals have been sensed. Let $f_T : 2^P \to Leaves(T)$ map each $S \in 2^P$ to the leaf that corresponds to S. We sometimes refer to a sensing tree also as a function $T : 2^P \to Q$, where for every $S \in 2^P$, we have that $T(S) = \tau(f_T(S))$, thus each assignment is mapped to the label of the leaf that corresponds to S. Also, for $S \in 2^P$, we use $sensed(T, S)$ for the set of signals sensed in the process of finding $T(S)$. Note that $|sensed(T, S)|$ is the length of the path from the root to $f_T(S)$.

Example 2. Let $P = \{a, b, c\}$ and $Q = \{q_1, q_2, q_3, q_4\}$. The tree T appearing in Figure 1 represents the function f with $f(\emptyset) = f(\{a, c\}) = f(\{a, b, c\}) = q_1$, $f(\{c\}) = f(\{a\}) = q_2$, $f(\{b\}) = f(\{b, c\}) = q_3$, and $f(\{a, b\}) = q_4$. □

We assume that sensing trees are *reduced*: they do not include redundant tests, namely internal vertices whose two children root identical subtrees. A sensing tree may be reduced in polynomial time by repeatedly replacing an internal vertex with two identical children by one of its children. It is not hard to see that the order in which such replacements are applied is not important.[3]

[3] Note that the above definition of a reduced tree is *syntactic*, in the sense it examines whether subtrees are identical. An alternative *semantic* definition removes a vertex if its two children root subtrees that represent the same function. Since the order of the signals along different paths may be different, two subtrees that represent the same function need not be identical, even if both are reduced. Thus, the semantic definition may result in smaller sensing trees. However, reducing trees according to

Example 3. Consider the sensing tree T from Figure 1. The vertex reached with the assignment $a = \texttt{False}$ and $b = \texttt{True}$ has two identical successors. By reducing T, we obtain the sensing tree T' in Figure 2. □

A *layout* is a sensing tree $L = \langle V, E, \tau \rangle$ in which τ is not defined for the leaves. Accordingly, a layout cannot be reduced, and all its paths include vertices that label all signals in P. A sensing tree T *follows* a layout L if T is obtained from L by reducing the sensing tree obtained by extending τ to the leaves. Intuitively, L directs the required sensing in T, but some tests that exist in L can be skipped in T.

2.4 The Sensing Cost of a Sensing Tree

Consider a sensing tree $T = \langle V, E, \tau \rangle$. The *sensing cost* of T is the expected number of signals that are sensed when evaluating an assignment $S \in 2^P$. Recall that we assume that each signal is valid with probability $\frac{1}{2}$. Thus, the probability of each assignment is $\frac{1}{2^{|P|}}$. Accordingly, the sensing cost of T, denoted $scost(T)$, is $scost(T) = \frac{1}{2^{|P|}} \sum_{S \in 2^P} |sensed(T, S)|$.

An equivalent definition of $scost(T)$ is based on a discounted sum of the vertices in T. For $v \in V$, let $depth(v)$ denote the length of the path from the root to v. Thus, the depth of the root is 0, the depth of its children is 1, and so on. Since the probability to reach the internal vertex v when reading an assignment $S \in 2^P$ that is chosen uniformly at random, is $2^{-depth(v)}$, we have the following.

Lemma 1. *For every sensing tree T, we have that $scost(T) = \sum_{v \in Int(T)} 2^{-depth(v)}$.*

Example 4. The sensing cost of the tree T' from Figure 2 is $\frac{1}{8} \cdot (3 + 3 + 2 + 2 + 3 + 3 + 3 + 3) = 2\frac{3}{4}$. Using discounted sum, we get $1 + 2 \cdot \frac{1}{2} + 3 \cdot \frac{1}{4} = 2\frac{3}{4}$. □

2.5 Static vs. Dynamic Sensing Trees

The sensing tree T' from Figure 2 is such that the labelling function τ follows the same order of the signals in P in all its branches. Indeed, all branches first sense a, then b, and then c, possibly skipping some of the signals (specifically, skipping c after reading $a = \texttt{False}$ and $b = \texttt{True}$). This corresponds to situations where the order of signals sensed is decided in advance and is *static*. In contrast, the order of signals sensed may be *dynamic* and depends on the valuation of signals sensed earlier.

Example 5. Consider the function f represented by the sensing tree T' from Figure 2. The two sensing trees appearing in Figure 3 represent f too. Both are reduced. The tree on the left is static, and it follows the order $c < b < a$. It is reduced, and still its sensing cost is 3, as all tree signals are read in all

the semantic definition is more complex. All our results apply also to the semantic definition.

assignments. The tree on the right is dynamic: When $a = \texttt{False}$, the next signal to sense is b. When $a = \texttt{True}$, the next signal to sense is c. Its sensing cost is $2\frac{1}{2}$, which is in fact the minimal sensing cost required for evaluating f. □

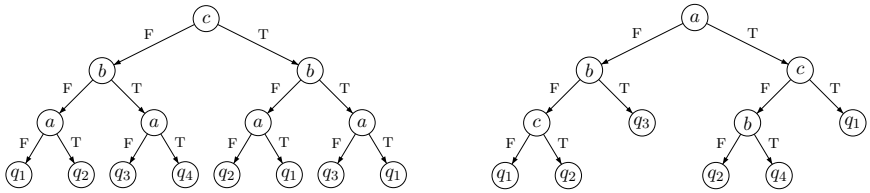


Fig. 3: A static (left) and a dynamic (right) sensing tree for f.

Note that, like a sensing tree, a layout may be static or dynamic. In particular, a static layout corresponds to a permutation on P. Indeed, such a layout is a sensing tree in which the vertices along all paths from the root to a leaf are labeled by all signals in P, with all paths follow the same ordering.

2.6 The Sensing Cost of a DFA and a Regular Language

Consider a DFA $\mathcal{A} = \langle 2^P, Q, q_0, \delta, \alpha \rangle$. Essentially, the sensing cost of $\mathcal{A}$ is the expected number of signals that $\mathcal{A}$ needs to sense in each transition when it runs on a random long word. Defining the sensing cost of $\mathcal{A}$, we first have to define the expected number of signals that $\mathcal{A}$ needs to sense in each state $q \in Q$. In [1], this is the number of potentially sensed signals in q. Defining the lazy sensing cost of $\mathcal{A}$, we allow the states to maintain sensing trees that represent the transition function. Indeed, the function $\delta : Q \times 2^P \to Q$ induces, for each state $q \in Q$, a function $\delta_q : 2^P \to Q$, where for every assignment $S \in 2^P$, we have that $\delta_q(S) = \delta(q, S)$. We distinguish between four classess, induced by the following two parameters.

- *Static vs. Dynamic*. That is, whether the sensing trees for δ_q are static or dynamic.
- *Global vs. Local*. That is, whether the sensing trees of the different states follow the same layout.

We denote the four classes by SG, SL, DG, and DL.

Let $\mathcal{T}$ be the set of all sensing trees (over P and Q, which we omit from the notation). A *legal choice of sensing trees* for the DFA $\mathcal{A}$ is a function $\gamma : Q \to \mathcal{T}$, such that for every state $q \in Q$, the sensing tree $\gamma(q)$ represents the function δ_q, and the following hold. Note that we can view γ as a mapping of states to layouts, which are then reduced to sensing trees. In particular, note that there is a unique way to reduce a layout to a sensing tree for a given function $f : 2^P \to Q$.

- In the LD class, there are no restrictions on γ.
- In the LS class, the image of γ contains only static sensing trees.

– In the GD class, all the sensing trees in the image of γ follow the same layout.
– In the GS class, all the sensing trees in the image of γ follow the same layout, which is static.

Consider a DFA $\mathcal{A}$. Let γ be a choice of sensing trees for $\mathcal{A}$. For a word $w = w_1 \cdots w_m \in (2^P)^*$, the *sensing cost of w by $\mathcal{A}$ with respect γ* is

$$scost_{\mathcal{A},\gamma}(w) = \frac{1}{m} \sum_{i=0}^{m-1} |sensed(\gamma(\delta(w[1,i]), w_{i+1}))|.$$

That is, $scost_{\mathcal{A},\gamma}(w)$ is the average number of signals that a state in the run of $\mathcal{A}$ on w senses when it reads w using the sensing trees chosen by γ. Note that the definition does not take into account the last state in the run, namely $\delta(w[1,m])$, as indeed no letter is read in it.

The *sensing cost of $\mathcal{A}$ with respect to γ* is then defined as the expected sensing cost of words of length tending to infinity, when the letters in 2^P are uniformly distributed. Formally,

$$scost(\mathcal{A},\gamma) = \lim_{m\to\infty} |2^P|^{-m} \sum_{w\in(2^P)^m} scost_{\mathcal{A},\gamma}(w).$$

That is, $scost(\mathcal{A},\gamma)$ is the expected sensing cost of words of length tending to infinity, when the letters in 2^P are uniformly distributed.

Now, the *sensing cost of $\mathcal{A}$* is the sensing cost of $\mathcal{A}$ using an optimal legal choice $\gamma : Q \to \mathcal{T}$ of sensing trees. Formally, for every class $\zeta \in \{LD, LS, GD, GS\}$, we define $\zeta scost(\mathcal{A})$ as $\min\{\zeta scost(\mathcal{A},\gamma) : \gamma \in Q^{\mathcal{T}} \text{ is legal in } \zeta\}$.

Finally, the *sensing cost of a regular language $L \subseteq (2^P)^*$* is the infimum of the sensing costs of DFAs that recognize L. That is, for every class $\zeta \in \{LD, LS, GD, GS\}$, we have that $\zeta scost(L) = \inf\{\zeta scost(\mathcal{A}) : L(\mathcal{A}) = L\}$. We use infimum in the definition since the number of DFAs recognizing L is unbounded. In fact, a-priori, there is no guarantee that $\zeta scost(L)$ is attained by a DFA.

3 Probability-Based Definition of Lazy-Sensing Cost

The definition of sensing cost of a DFA in Section 2.6 is not effective, in the sense it does not suggest a way to calculate the sensing cost of a DFA. In this section we describe an alternative definition, which does suggest such a way. Essentially, while the definition in Section 2.6 refers to the sensing cost of words of increasing length, our definition here refers to the sensing costs of states visited by random walks on the DFA. We first need some definitions and notations about probability.

A *Markov chain* $M = \langle S, P\rangle$ consists of a finite state space S and a stochastic transition matrix $P : S \times S \to [0,1]$. That is, for all $s \in S$, we have $\sum_{s'\in S} P(s,s') = 1$.

Consider a directed graph $G = \langle V, E \rangle$. A *strongly connected component* (SCC) of G is a maximal (with respect to containment) set $C \subseteq V$ such that for all $x, y \in C$, there is a path from x to y. An SCC (or state) is *ergodic* if no other SCC is reachable from it, and is *transient* otherwise.

An automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ induces a directed graph $G_{\mathcal{A}} = \langle Q, E \rangle$ in which $\langle q, q' \rangle \in E$ iff there is a letter σ such that $q' = \delta(q, \sigma)$. When we talk about the SCCs of $\mathcal{A}$, we refer to those of $G_{\mathcal{A}}$. Recall that we assume that the letters in Σ are uniformly distributed, thus $\mathcal{A}$ also corresponds to a Markov chain $M_{\mathcal{A}}$ in which the probability of a transition from state q to state q' is $p_{q,q'} = \frac{1}{|\Sigma|}|\{\sigma \in \Sigma : \delta(q, \sigma) = q'\}|$. Let $\mathcal{C}$ be the set of $\mathcal{A}$'s SCC, and $\mathcal{C}_e \subseteq \mathcal{C}$ be the set of its ergodic SCC's.

Consider an ergodic SCC $C \in \mathcal{C}_e$. Let P_C be the matrix describing the probability of transitions in C. Thus, the rows and columns of P_C are associated with states, and the value in coordinate q, q' is $p_{q,q'}$. By [4], there is a unique probability vector $\pi_C \in [0,1]^C$ such that $\pi_C P_C = \pi_C$. This vector describes the *stationary distribution* of C: for all $q \in C$ it holds that $\pi_C(q) = \lim_{m \to \infty} \frac{E^C_m(q)}{m}$, where $E^C_m(q)$ is the average number of occurrences of q in a run of $M_{\mathcal{A}}$ of length m that starts anywhere in C [4]. Thus, intuitively, $\pi_C(q)$ is the probability that a long run that starts in C ends in q. In order to extend the distribution to the entire Markov chain of $\mathcal{A}$, we have to take into account the probability of reaching each of the ergodic components. The *SCC-reachability distribution* of $\mathcal{A}$ is the function $\rho : \mathcal{C}_e \to [0,1]$ that maps each ergodic SCC C of $\mathcal{A}$ to the probability that $M_{\mathcal{A}}$ eventually reaches C, starting from the initial state. The *limiting distribution* $\pi : Q \to [0,1]$ is now defined by $\pi(q) = 0$, if q is transient, and $\pi(1) = \pi_C(q) \cdot \rho(C)$, if q is in some $C \in \mathcal{C}_e$. By [4], the limiting distributions can be computed in polynomial time by solving a system of linear equations.

Intuitively, the limiting distribution of state q describes the probability of a run on a random and long input word to end in q. Formally, we have the following lemma.

Lemma 2. [1] *Let $E_m(q)$ be the expected number of occurrences of a state q in a run of length m of $M_{\mathcal{A}}$ that starts in q_0. Then, $\pi(q) = \lim_{m \to \infty} \frac{E_m(q)}{m}$.*

The alternative definition is based on the following lemma, see proof in the full version.

Lemma 3. *Let $\mathcal{A} = \langle 2^P, Q, q_0, \delta, \alpha \rangle$ be a DFA, and let γ be a choice of sensing trees for $\mathcal{A}$. Then,*

$$scost(\mathcal{A}, \gamma) = \lim_{m \to \infty} |2^P|^{-m} \sum_{w \in (2^P)^m} \frac{1}{m} \sum_{i=0}^{m-1} scost(\gamma(\delta(w[1,i])).$$

Lemma 3 enables us to follow the exact same considerations in [1], thus computing the lazy-sensing cost of a DFA by examining its induced Markov chain. Formally, we have the following.

Theorem 1. *Let $\mathcal{A}$ be a DFA with alphabet 2^P, state space Q, and limiting distribution $\pi : Q \to [0,1]$. Then, for every choice γ of sensing trees for $\mathcal{A}$, we have that $scost(\mathcal{A}, \gamma) = \sum_{q \in Q} \pi(q) \cdot scost(\gamma(q))$.*

4 Lazy-Sensing Cost vs. Size

In this section we examine the trade-off between the size of a DFA and its sensing cost in the four lazy classes of sensing. It is shown in [1] that in the naive setting of sensing, namely when all signals are read simultaneously, minimizing the size of a DFA goes hand in hand with minimizing its sensing cost. Thus, minimal naive sensing is attained in a minimal-size DFA. In this section, we show that this good news is carried over to lazy sensing.

Consider a language $L \subseteq \Sigma^*$. For two finite words u_1 and u_2, we say that u_1 and u_2 are *right L-indistinguishable*, denoted $u_1 \sim_L u_2$, if for every $z \in \Sigma^*$, we have that $u_1 \cdot z \in L$ iff $u_2 \cdot z \in L$. Thus, $\sim_L$ is the Myhill-Nerode right congruence used for minimizing automata. For $u \in \Sigma^*$, let $[u]$ denote the equivalence class of u in $\sim_L$ and let $\langle L \rangle$ denote the set of all equivalence classes. Each class $[u] \in \langle L \rangle$ is associated with the *residual language* $u^{-1}L = \{w : uw \in L\}$. When L is regular, the set $\langle L \rangle$ is finite, and induces the *residual automaton* of L, defined by $\mathcal{R}_L = \langle \Sigma, \langle L \rangle, \delta^L, [\epsilon], \alpha \rangle$, with $\delta^L([u], a) = [u \cdot a]$, for all $[u] \in \langle L \rangle$ and $a \in \Sigma$. Also, α contains all classes $[u]$ with $u \in L$. The DFA $\mathcal{R}_L$ is well defined and is the unique minimal DFA for L.

Lemma 4. *Consider a regular language $L \subseteq \Sigma^*$. For every DFA $\mathcal{A}$ with $L(\mathcal{A}) = L$ and lazy-sensing class $\zeta \in \{LD, LS, GD, GS\}$, it holds that $\zeta scost(\mathcal{R}_L) \leq \zeta scost(\mathcal{A})$.*

Proof: Let $\mathcal{A} = \langle 2^P, Q, q_0, \delta, \alpha \rangle$ be a DFA such that $L(\mathcal{A}) = L$. Consider a reachable state $q \in Q$. Let $u \in (2^P)^*$ be a word such that $\mathcal{A}$ reaches the q after reading u, thus $q = \delta^*(q_0, u)$. Recall that $\mathcal{R}_L$ reaches the state $[u]$ after reading u. We claim that for every layout T of a sensing tree over P, we have that $scost(T_{[u]}) \leq scost(T_q)$, where $T_{[u]}$ is the sensing tree obtained from T by reducing it according to the transitions of $\mathcal{R}_L$ from $[u]$, and T_q is the sensing tree obtained from T by reducing it according to the transitions of $\mathcal{A}$ from q.

By Lemma 1, for every sensing tree T, we have $scost(T) = \sum_{v \in Int(T)} 2^{-depth(v)}$. Accordingly, it suffices to prove that for all letters $\sigma, \sigma' \in 2^P$ if $\delta(q, \sigma) = \delta(\sigma')$, then $\delta^L([u], \sigma) = \delta^L([u], \sigma')$. Indeed, this would guarantee that every vertex that is deleted from the layout T when it is reduced to T_q is also deleted when T is reduced to $T_{[u]}$. In the full version, we prove this claim. □

Since $L(\mathcal{R}_L) = L$, then for every class $\zeta \in \{LD, LS, GD, GS\}$, we have that $\zeta scost(L) \leq \zeta scost(\mathcal{R}_L)$. Thus, together with Lemma 4, we can conclude with the following.

Theorem 2. *For every regular language $L \subseteq (2^P)^*$ and lazy-sensing class $\zeta \in \{LD, LS, GD, GS\}$, we have that $\zeta scost(L) = \zeta scost(\mathcal{R}_L)$.*

5 Comparing the Different Sensing Classes

In this section we examine the saving of sensing that the lazy classes enable. We start by comparing the lazy classes with the setting in [1], where all signals are sensed simultaneously, and continue to examine the relations among the four lazy classes.

5.1 Lazy vs. Naive Sensing

Recall that in the setting of [1], which we refer to as *naive sensing*, the sensing cost of a DFA $\mathcal{A}$ is defined as follows (we use the prefix N for naive).

$$Nscost(\mathcal{A}) = \lim_{m\to\infty} |2^P|^{-m} \sum_{w\in(2^P)^m} \frac{1}{m} \sum_{i=0}^{m-1} |psensed(\delta(w[1,i]))|,$$

as all the potentially sensed signals must in fact be sensed. The sensing cost of a regular language in the naive sensing setting is then defined as $Nscost(L) = \inf\{Nscost(\mathcal{A}) : L(\mathcal{A}) = L\}$. We first show that, as expected, the sensing cost in all lazy classes is never higher than the naive one.

Theorem 3. *For every DFA $\mathcal{A}$ over an alphabet 2^P and for every lazy-sensing class $\zeta \in \{LD, LS, GD, GS\}$, we have that $\zeta scost(\mathcal{A}) \leq Nscost(\mathcal{A})$.*

Proof: We prove the theorem for the static classes LS and GS. Since every choice function γ that is legal in these classes is legal also in the corresponding dynamic class, the result for LD and GD follows. Let $\mathcal{A} = \langle 2^P, Q, q_0, \delta, \alpha\rangle$, and let γ be a choice of sensing trees for the DFA $\mathcal{A}$ that is legal with respect to $\zeta \in \{LS, GS\}$. We claim that for every state $q \in Q$, if $p \notin psensed(q)$, then there is no internal vertex with the label p in $\gamma(q)$. Since this holds for all choices function γ, in particular these that attain $\zeta scost(\mathcal{A})$, the theorem follows.

In order to prove the claim, consider a state q and let $\gamma(q) = \langle V, E, \tau\rangle$. Consider an internal vertex $v \in V$ such that $\tau(v) = p$. If $p \notin psensed(q)$, then for every $S \in 2^P$, we have that $\delta(q, S \setminus \{p\}) = \delta(q, S \cup \{p\})$. Therefore, regardless of ζ, the subtrees of $\langle V, E, \tau\rangle$ with roots v_{left} and v_{right} calculate the same function. Since ζ is static, this implies that v_{left} and v_{right} root identical subtrees, and so we can reduce $\langle V, E, \tau\rangle$ by redirecting the edge that enters v to v_{left}. □

Corollary 1. *For every regular language $L \subseteq (2^P)^*$ and lazy-sensing class $\zeta \in \{LD, LS, GD, GS\}$, we have that $\zeta scost(L) \leq Nscost(L)$.*

We now show that, on the one hand, there are cases where lazy sensing is not helpful (Theorem 4), and, on the other hand, there are cases where the saving that lazy sensing enables is unbounded (Theorem 5).

Theorem 4. *For every finite set P of signals, there is a regular language $L \subseteq (2^P)^*$ such that for every lazy-sensing class $\zeta \in \{LD, LS, GD, GS\}$, we have that $\zeta scost(L) = Nscost(L)$.*

Proof: Let $L = \{w_1 \cdots w_m \in (2^P)^* : m \geq 1 \text{ and } |w_m| \text{ is even}\}$ be the language of all words in $(2^P)^*$ that end with a letter that consists of an even number of signals. A DFA $\mathcal{A}$ for L must sense all the signals in P in all states. Indeed, the DFA $\mathcal{A}$ has to identify, in all states, whether the current input letter consists of an even number of signals. Thus, for every state of q of $\mathcal{A}$ and choice function γ, we have that $scost(\gamma(q)) = |P|$. By Theorem 1 and the definition of the naive sensing cost of a DFA, we conclude that $\zeta scost(\mathcal{A}) = Nscost(\mathcal{A}) = |P|$. Since the above holds for every DFA $\mathcal{A}$ recognizing L, the result follows. □

Theorem 5. *For every $n \geq 1$, there is a regular language L_n over $2^{\{p_1,\ldots,p_n\}}$ such that $Nscost(L_n) = n$, yet for every lazy-sensing class $\zeta \in \{LD, LS, GD, GS\}$, we have that $\zeta scost(L_n) < 2$.*

Proof: Let $P_n = \{p_1, ..., p_n\}$ be a set of n signals, let $\sigma = P_n$, and let L_n be the language of all words with an even number of occurrences of the letter σ. A minimal DFA $\mathcal{A}_n$ that recognizes L_n consists of two states, keeping track of the parity of occurrences of σ, see Figure 4 below.

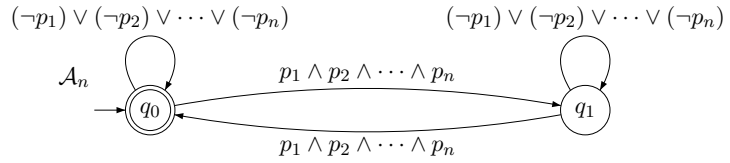
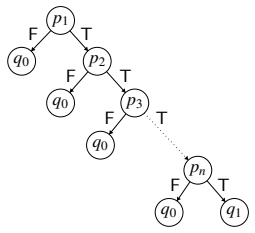


Fig. 4: Lazy sensing is better than naive sensing.

It is easy to see that $psensed(q_0) = psensed(q_1) = P_n$. Indeed, for every $p_i \in P_n$ we have $\delta(q_0, P_n \cup \{p_i\}) = q_1 \neq q_0 = \delta(q_0, P_n \setminus \{p_i\})$ and $\delta(q_1, P_n \cup \{p_i\}) = q_0 \neq q_1 = \delta(q_1, P_n \setminus \{p_i\})$. Since q_0 and q_1 are the only states of $\mathcal{A}_n$, it follows that $Nscost(\mathcal{A}_n) = n$. Also, as the naive sensing cost of a regular language is attained in the minimal DFA recognizing the language [1], it follows that $Nscost(L_n) = n$.

We now consider the lazy sensing cost of L_n. By Theorem 2, here too we can consider the DFA $\mathcal{A}_n$. It is easy to see that a sensing tree T of minimal sensing cost for each of the states q_i, with $i \in \{0, 1\}$, consists of n internal vertices, one in each height from 0 to $n-1$, labeled by all of the signals in P_n in some arbitrary order.
For each such internal vertex, its left child is a leaf labeled q_i. If the vertex is in height different from $n-1$, its right child is another internal vertex. If it is in height $n-1$, its right child is a leaf labeled with q_{1-i}. The figure on the right shows such a minimal sensing tree for the state q_0.



It is not hard to see that the suggested sensing tree is legal in all classes ζ. Indeed, the same layout is used for q_0 and q_1, and the tree follows an order on

P that is independent of the values read. By Lemma 1, we have that $scost(T) = \sum_{v \in Int(T)} 2^{-depth(v)} = \sum_{i=0}^{n-1} 2^{-i} = 2 - \frac{1}{2^{n-1}}$. Thus, by Theorems 2 and 1, for all classes $\zeta \in \{LD, LS, GD, GS\}$, we have that $\zeta scost(L_n) = scost(\mathcal{A}_n) = 2 - \frac{1}{2^{n-1}} < 2$. □

5.2 Comparison of the Different Lazy-Sensing Classes

In this section, we compare the sensing costs in the different lazy sensing classes. First, since every choice function that is legal in the global classes is legal in the local ones, and every choice function that is legal in the static classes is legal in the dynamic ones, we immediately have the following.

Theorem 6. *For every regular language $L \subseteq (2^P)^*$, the following holds*

$$(i) \quad LDscost(L) \leq GDscost(L), \qquad (ii) \quad LSscost(L) \leq GSscost(L),$$
$$(iii) \quad LDscost(L) \leq LSscost(L), \qquad (iv) \quad GDscost(L) \leq GSscost(L).$$

Theorem 4 implies that there are languages for which the sensing costs in the four classes coincide. In the following, we describe cases where the inequalities in Theorem 6 are strict. In addition, we show that the local-static class and the global-dynamic class are incomparable: there is a language L with $LSscost(L) < GDscost(L)$ and also a language L with $LSscost(L) > GDscost(L)$.

We start with the advantage of the local classes over the global ones:

Lemma 5. *There exists a regular language $L \subseteq (2^P)^*$ such that $LDscost(L) = LSscost(L) < GDscost(L) = GSscost(L)$.*

Proof: Let $P = \{a, b\}$ and consider the DFA $\mathcal{A}$ with alphabet 2^P shown in Figure 5. It can be easily verified that $\mathcal{A}$ is a minimal DFA, for example using

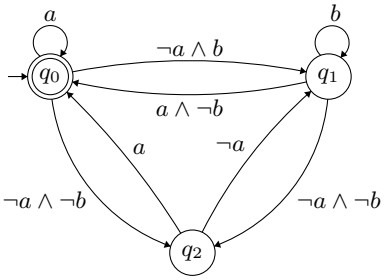


Fig. 5: Local lazy sensing is better than global one.

the standard DFA minimization algorithm. In the full version, we prove that $LDscost(\mathcal{A}) = LSscost(\mathcal{A}) < GDscost(\mathcal{A}) = GSscost(\mathcal{A})$, which, by Theorem 2, implies that $L(\mathcal{A})$ satisfies the conditions in the lemma. □

We continue with the advantage of the dynamic classes over the static ones:

Lemma 6. *There exists a regular language* $L \subseteq (2^P)^*$ *such that* $LDscost(L) = GDscost(L) < LSscost(L) = GSscost(L)$.

Proof: Let $P = \{a, b, c\}$ and consider the DFA $\mathcal{A}$ with alphabet 2^P shown in Figure 6.

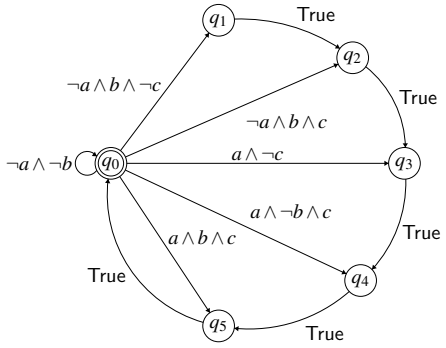


Fig. 6: Dynamic lazy sensing is better than static one.

Again, it can be verified that $\mathcal{A}$ is minimal, thus it is left to prove that $LDscost(\mathcal{A}) = GDscost(\mathcal{A}) < LSscost(\mathcal{A}) = GSscost(\mathcal{A})$, which we do in the full version. □

6 Directions for Future Research

We introduced lazy sensing for deterministic finite automata. We studied the basic problems about the setting, namely a study of four natural classes of lazy sensing, their comparison with naive sensing, and the trade-off between minimizing the sensing cost of a DFA and minimizing its size. We left open several interesting problems, which we discuss below.

Computing lazy sensing cost: In [1], it is shown that the naive sensing cost of a DFA can be calculated in polynomial time using standard Markov chain algorithms. Accordingly, the naive sensing cost of a regular language can also be calculated in polynomial time using the classical minimization algorithm for DFA. In order to compute the sensing cost in lazy-sensing classes, one also needs to find the optimal sensing trees for a given DFA.

The involved questions now depend on the lazy-sensing class. For the SL class, the problem is strongly related to the problem of finding an optimal ordering for the variables in a BDD, and the complexity depends on the way the transition function of the DFA is given. In the GS class, there is the extra requirement that the same order is used in the transition functions of all states. Then, in

the dynamic classes, the layouts we may use need not follow an ordering for the variables, and techniques from the theory of BDDs are less relevant.

Random lazy sensing: While dynamic and local lazy sensing may save more than static and global lazy sensing, they require the maintenance of more complex data structures. In addition to studying lazy sensing classes with some bounded level of dynamics or locality, it is interesting to examine a stochastic approach, where the signal to be sensed next is chosen randomly. It is not hard to see that our results in Sections 4 and 5.1 apply also to the random lazy setting, when we examine the expected sensing cost of a DFA, with expectation now referring to both the input words and the order in which signals are sampled.

References

1. S. Almagor, D. Kuperberg, and O. Kupferman. Regular sensing. In *Proc. 34th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 29 of *LIPIcs*, pages 161–173. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2014.
2. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
3. D.L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52:1289–1306, 2006.
4. C. Grinstead and J. Laurie Snell. 11:markov chains. In *Introduction to Probability*. American Mathematical Society, 1997.
5. G. Kindler. *Property Testing, PCP, and Juntas*. PhD thesis, Tel Aviv University University, 2002.
6. D.M. Miller and R. Drechsler. On the construction of multiple-valued decision diagrams. In *32nd IEEE International Symposium on Multiple-Valued Logic*, pages 245–253. IEEE Computer Society, 2002.
7. J. Minker and Rita G. Minker. Optimization of boolean expressions-historical developments. *IEEE Ann. Hist. Comput.*, 2(3):227–238, 1980.
8. S. Muthukrishnan. Theory of data stream computing: where to go. In *Proc. 30th Symposium on Principles of Database Systems*, pages 317–319, 2011.