International Journal of Foundations of Computer Science
Vol. 30, Nos. 6 & 7 (2019) 831–873
© World Scientific Publishing Company
DOI: 10.1142/S0129054119400203

Sensing as a Complexity Measure

Shaull Almagor*
Department of Computer Science, Oxford University
Oxford, OX1 3QD, UK
shaull.almagor@cs.ox.ac.uk

Denis Kuperberg
CNRS, ENS Lyon, Université de Lyon, LIP
Lyon 69364, France
denis.kuperberg@ens-lyon.fr

Orna Kupferman
School of Engineering and Computer Science
The Hebrew University, Jerusalem 9190416, Israel
orna@cs.huji.ac.il

Received 15 November 2017
Accepted 3 December 2018
Communicated by C. Câmpeanu and G. Pighizzini

The size of deterministic automata required for recognizing regular and ω-regular languages is a well-studied measure for the complexity of languages. We introduce and study a new complexity measure, based on the *sensing* required for recognizing the language. 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. We study the sensing cost of regular and ω-regular languages, as well as applications of the study in practice, especially in the monitoring and synthesis of reactive systems.

Keywords: Automata; regular languages; ω-regular languages; complexity; sensing; minimization.

1. Introduction

Studying the complexity of a formal language, there are several complexity measures to consider. When the language is given by means of a Turing Machine, the traditional measures are time and space demands. Theoretical interest as well as practical considerations have motivated additional measures, such as randomness (the number of random bits required for the execution) [15] or communication complexity (number and length of messages required) [14]. For regular and ω-regular

*Corresponding author.

languages, given by means of finite-state automata, the classical complexity measure is the size of a minimal deterministic automaton that recognizes the language.

We introduce and study 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. 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 [12]. In more practical applications, mathematical tools in signal processing are used to reconstruct information based on compressed sensing [7], 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" [16].

Our interest in regular sensing is motivated by the use of finite-state automata in reasoning about on-going behaviors of reactive systems. In particular, a big challenge in the design of monitors is an optimization of the sensing needed for deciding the correctness of observed behaviors. Our goal is to formalize regular sensing in the finite-state setting and to study the sensing complexity measure for regular and ω-regular languages.

We consider languages over alphabets of the form 2^P, for a finite set P of signals. Consider a deterministic automaton $\mathcal{A}$ 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. We extend the definition to automata by assuming a uniform distribution of the inputs.[a] Thus, the sensing cost of $\mathcal{A}$ is the limit of the expected sensing of runs over words of increasing length.[b] We show that this definition coincides with one that is based on the stationary distribution of the Markov chain induced by $\mathcal{A}$, which enables us to calculate the sensing cost of an automaton in polynomial time. The sensing cost of a language L, of either finite or infinite words, is then the infimum of the sensing costs of deterministic automata for L. In the case of infinite words, one can study different classes of automata, yet we show that the sensing cost is independent of the acceptance condition being used.

We start by studying the sensing cost of regular languages of finite words. For the complexity measure of size, the picture in the setting of finite words is

[a]Our study and results apply also to a non-uniform distribution on the letters, given by a Markov chain.

[b]Alternatively, one could define the sensing cost of $\mathcal{A}$ as the cost of its "most sensing" run. Such a worst-case approach is taken in Ref. [6], where the sensing cost needs to be kept under a certain budget in all computations, rather than in expectation. We find the average-case approach we follow appropriate for sensing, as the cost of operating sensors may well be amortized over different runs of the system, and requiring the budget to be kept under a threshold in every run may be too restrictive. Thus, the automaton must answer correctly for every word, but the sensing should be low only on average, and it is allowed to operate an expensive sensor now and then.

very clean: each language L has a unique minimal deterministic automaton (DFA), namely the *residual automaton* $\mathcal{R}_L$ whose states correspond to the equivalence classes of the Myhill-Nerode right-congruence relation for L. We show that minimizing the state space of a DFA can only reduce its sensing cost. Hence, the clean picture of the size measure is carried over to the sensing measure: the sensing cost of a language L is attained in the DFA $\mathcal{R}_L$. In particular, since DFAs can be minimized in polynomial time, we can construct in polynomial time a minimally-sensing DFA, and can compute in polynomial time the sensing cost of languages given by DFAs.

We then study the sensing cost of ω-regular languages, given by means of deterministic parity automata (DPAs). Recall the size complexity measure. There, the picture for languages of infinite words is not clean: A language needs not have a unique minimal DPA, and the problem of finding one is NP-complete [21]. It turns out that the situation is challenging also in the sensing measure. First, we show that different minimal DPAs for a language may have different sensing costs. In fact, bigger DPAs may have smaller sensing costs.

To see the intricacy in the case of ω-regular languages, consider a component in a vacuum-cleaning robot that monitors the dust collector and checks that it is empty infinitely often. The proposition *empty* indicates whether the collector is empty and a sensor needs to be activated in order to know its truth value. One implementation of the component would sense *empty* throughout the computation. This corresponds to the classical two-state DPA for "infinitely often *empty*". A different implementation can give up the sensing of *empty* for some fixed number k of states, then wait for *empty* to hold, and so forth. The bigger k is, the lazier is the sensing and the smaller the sensing cost is. As the example demonstrates, there may be a trade-off between the sensing cost of an implementation and its size. Other considerations, like a preference to have eventualities satisfied as soon as possible, enter the picture too.

Our main result is that despite the above intricacy, the sensing cost of an ω-regular language L is the sensing cost of the residual automaton $\mathcal{R}_L$ for L. It follows that the sensing cost of an ω-regular language can be computed in polynomial time. Unlike the case of finite words, it may not be possible to define L on top of $\mathcal{R}_L$. Interestingly, however, $\mathcal{R}_L$ does capture exactly the sensing required for recognizing L. The proof goes via a sequence $(\mathcal{B}_n)_{n=1}^{\infty}$ of DPAs whose sensing costs converge to that of L. The DPA $\mathcal{B}_n$ is obtained from a DPA $\mathcal{A}$ for L by a lazy sensing strategy that spends time in n copies of $\mathcal{R}_L$ between visits to $\mathcal{A}$, but spends enough time in $\mathcal{A}$ to ensure that the language is L.

In the context of formal methods, sensing has two appealing applications. The first is *monitoring*: we are given a computation and we have to decide whether it satisfies a specification. When the computations are over 2^P, we want to design a monitor that minimizes the expected average number of sensors used in the monitoring process. Monitoring is especially useful when reasoning about *safety* specifications [11]. There, every computation that violates the specification has a bad prefix — one all whose extensions are not in L. Hence, as long as the computation is a prefix of some word in L, the monitor continues to sense and examine the computation. Once a bad prefix is detected, the monitor declares an error and no further sensing is required. The second application is *synthesis*. Here, the set P of signals is partitioned into sets I and O of input and output signals, respectively. We are given a specification L over the alphabet $2^{I \cup O}$, and our goal is to construct an I/O *transducer* that realizes L. That is, for every sequence of assignments to the input signals, the transducer generates a sequence of assignments to the output signals so that the obtained computation is in L [18]. Our goal is to construct a transducer that minimizes the expected average number of sensors (of input signals) that are used along the interaction.

The definition of sensing cost described above falls short in the above two applications. For the first, the definition above does not distinguish between words in the language and words not in the language, whereas in monitoring we care only for words in the language. In particular, according to the definition above, the sensing cost of a safety language is always 0. For the second, the definition above considers automata and does not partition P into I and O, whereas synthesis refers to I/O-transducers. Moreover, unlike automata, correct transducers generate only computations in the language, and they need not generate all words in the language — only these that ensure receptiveness with respect to all sequences of inputs.

We thus continue and study sensing in the context of monitoring and synthesis. We suggest definitions that capture the intuition of "required number of sensors" in these settings and solve the problems of generating monitors and transducers that minimize sensing. For both settings, we focus on safety languages.

Consider, for example, a traffic monitor that has access to various sensors on roads and whose goal is to detect accidents. Once a road accident is detected, an alarm is raised to the proper authorities and the monitoring is stopped until the accident has been taken care of. The monitor can read the speed of cars along the roads, as well as the state of traffic lights. An accident is detected when some cars do not move even though no traffic light is stopping them. Sensing the speed of every car and checking every traffic light requires huge sensing. Our goal is to find a monitor that minimizes the required sensing and still detects all accidents. In the synthesis setting, our goal is extended to designing a transducer that controls the traffic lights according to the speed of the traffic in each direction, and satisfies some specification (say, give priority to slow traffic), while minimizing the sensing of cars.

We revise our definition as follows. Let us start with monitoring. Recall that the definition of sensing above assumes a uniform probability on the assignments to the signals, whereas in monitoring we want to consider instead more intricate probability spaces — ones that restrict attention to words in the language. As we show, there is more than one way to define such probability spaces, each leading to a different measure. We study two such measures. In the first, we sample a word randomly, letter by letter, according to a given distribution, allowing only letters that do not generate bad prefixes. In the second, we construct a sample space directly on the words in the language. We show that in both definitions, we can compute the sensing cost of the language in polynomial time, and that the minimal sensing cost is attained by a minimal-size automaton. Thus, luckily enough, even though different ways in which a computation may be given in an online manner calls for two definitions of sensing cost, the design of a minimally-sensing monitor is the same in the two definitions.

Let us continue to synthesis. Recall that there, given a specification over sets I and O of input and output signals, the goal is to construct a finite-state system that, given a sequence of input signals, generates a computation that satisfies the specification. In each moment in time, the system reads an assignment to the input signals, namely a letter in 2^I, which requires the activation of $|I|$ Boolean sensors. A well-studied special case of limited sensing is synthesis with *incomplete information*. There, the system can read only a subset of the signals in I, and should still generate only computations that satisfy the specification [5, 13]. A more sophisticated case of sensing in the context of synthesis is studied in Ref. [6], where the system can read some of the input signals some of the time. In more detail, sensing the truth value of an input signal has a cost, the system has a budget for sensing, and it tries to realize the specification while minimizing the required sensing budget.

The main challenge there is that we no longer need to consider all words in the language. This introduces a new degree of freedom, which requires different techniques than those used for the definition above. In particular, while a minimal-size transducer for a safety language can be defined on top of the state space of a minimal-size deterministic automaton for the language, this is not the case when we seek minimally-sensing transducers. In fact, we show that a minimally-sensing transducer for a safety language might be exponentially bigger than a minimal-size automaton for the language. Consequently, the problems of computing the minimal sensing cost and finding a minimally-sensing transducer are EXPTIME-complete even for specifications given by means of deterministic safety automata. On the positive side, a transducer that attains the minimal sensing cost always exists for safety specifications.

2. Preliminaries

2.1. *Automata and transducers*

A *deterministic automaton on finite words* (DFA, for short) is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \nrightarrow Q$ is a partial transition function, and $\alpha \subseteq Q$ is a set of accepting states. We sometimes refer to δ as a relation $\Delta \subseteq Q \times \Sigma \times Q$, with $\langle q, \sigma, q' \rangle \in \Delta$ iff $\delta(q, \sigma) = q'$. A run of $\mathcal{A}$ on a word $w = \sigma_1 \cdots \sigma_2 \cdots \sigma_m \in \Sigma^*$ 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. 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.

We sometimes refer also to nondeterministic automata (NFAs), where $\delta : Q \times \Sigma \to 2^Q$ suggests several possible successor states. Thus, an NFA may have several runs on an input word w, and it accepts w if at least one of them is accepting.

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 $\langle [u], a, [u \cdot a] \rangle \in \Delta_L$ 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.

A *deterministic automaton on infinite words* is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Q, q_0, and δ are as in DFA, and α is an acceptance condition. A run of $\mathcal{A}$ on an infinite input word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is defined as for automata on finite words, except that the sequence of visited states is now infinite. For a run $r = q_0, q_1, \ldots$, let $\mathit{inf}(r)$ denote the set of states that r visits infinitely often. Formally, $\mathit{inf}(r) = \{q : q = q_i$ for infinitely many i's$\}$. We consider the following acceptance conditions. In a *Büchi* automaton, the acceptance condition is a set $\alpha \subseteq Q$ and a run r is accepting iff $\mathit{inf}(r) \cap \alpha \neq \emptyset$. Dually, in a *co-Büchi* automaton, again $\alpha \subseteq Q$, but r is accepting iff $\mathit{inf}(r) \cap \alpha = \emptyset$. In a deterministic *looping* automaton, every run is accepting. Thus, a word is accepted if there is a run of the automaton on it.[c] Since every run is accepting, we omit the acceptance condition and write $\mathcal{A} = \langle \Sigma, Q, q_0, \delta \rangle$. Finally, a parity condition is a mapping $\alpha : Q \to [i, \ldots, j]$, for integers $i \leq j$, and a run r is accepting iff $\max_{q \in \mathit{inf}(r)}\{\alpha(q)\}$ is even. We use the acronyms NBA, DBA, NCA, DCA, DLA, NLA, NPA, and DPA to denote nondeterministic/deterministic Büchi/co-Büchi/looping/parity automata.

The definitions of right congruence $\sim_L$ as well as the residual automaton $\mathcal{R}_L$ naturally extend to languages $L \subseteq \Sigma^\omega$, by saying that for finite words $u_1, u_1 \in \Sigma^*$, we have $u_1 \sim_L u_2$ if for every $z \in \Sigma^\omega$ it holds that $u_1 \cdot z \in L$ iff $u_2 \cdot z \in L$. The relation $\sim_L$ remains an equivalence relation in this case, and has finite index for ω-regular languages. Therefore, the residual automaton $\mathcal{R}_L$ is well defined.

Here, however, $\mathcal{R}_L$ need not accept the language L, and we ignore its acceptance condition. Indeed, consider for example the language $L \subseteq \{a, b\}^\omega$ of words in which a occurs infinitely often, then it is easy to see that $\sim_L$ has a single equivalence class, and therefore $\mathcal{R}_L$ has a single state, and cannot recognize L with any acceptance condition.

For finite sets I and O of input and output signals, respectively, an *I/O transducer* is $\mathcal{T} = \langle I, O, Q, q_0, \delta, \rho \rangle$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times 2^I \to Q$ is a total transition function, and $\rho : Q \to 2^O$ is a labeling function on the states. The run of $\mathcal{T}$ on a word $w = i_0 \cdot i_1 \cdots \in (2^I)^\omega$ is the sequence

[c]Note that a looping automaton is a special case of Büchi with $\alpha = Q$.

of states $q_0, q_1, \ldots$ such that $q_{k+1} = \delta(q_k, i_k)$ for all $k \geq 0$. The *output* of $\mathcal{T}$ on w is then $o_1, o_2, \ldots \in (2^O)^\omega$ where $o_k = \rho(q_k)$ for all $k \geq 1$. Note that the first output assignment is that of q_1, and we do not consider $\rho(q_0)$. This reflects the fact that the environment initiates the interaction. The *computation of* $\mathcal{T}$ *on* w is then $\mathcal{T}(w) = i_0 \cup o_1, i_1 \cup o_2, \ldots \in (2^{I \cup O})^\omega$.

Note that the structure of each I/O-transducer $\mathcal{T}$ induces a DLA $\mathcal{A}_\mathcal{T}$ over the alphabet 2^I with a total transition relation. Thus, the language of the DLA is $(2^I)^\omega$, reflecting the receptiveness of $\mathcal{T}$.

2.2. *Safety languages*

Consider a language $L \subseteq \Sigma^\omega$. A finite word $x \in \Sigma^*$ is a *bad prefix* for L if for every $y \in \Sigma^\omega$, we have that $x \cdot y \notin L$. That is, x is a bad prefix if all its extensions are words not in L. The language L is then a *safety* language if every word not in L has a bad prefix. For a language L, let $\mathit{pref}(L) = \{x \in \Sigma^* :$ there exists $y \in \Sigma^\omega$ such that $x \cdot y \in L\}$ be the set of prefixes of words in L. Note that each word in Σ^* is either in $\mathit{pref}(L)$ or is a bad prefix for L. Since the set $\mathit{pref}(L)$ for a safety language L is *fusion closed* (that is, a word is in L iff all its prefixes are in $\mathit{pref}(L)$), an ω-regular language is safety iff it can be recognized by a DLA [22].

Consider a safety language L over sets I and O of input and output signals. We say that L is I/O*-realizable* if there exists an I/O transducer $\mathcal{T}$ all whose computations are in L. Thus, for every $w \in (2^I)^\omega$, we have that $\mathcal{T}(w) \in L$. We then say that $\mathcal{T}$ I/O*-realizes* L. When I and O are clear from the context, we omit them. In the *synthesis* problem, we get as input a safety language L over I and O, say by means of a DLA, and we are asked to return an I/O-transducer that realizes L or to declare that L is not I/O-realizable.

2.3. *Sensing*

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 an automaton $\mathcal{A} = \langle 2^P, Q, q_0, \delta, \alpha \rangle$. For a state $q \in Q$ and a signal $p \in P$, we say that p is *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 sensed in q if knowing its value may affect the destination of at least one transition from q. We use $\mathit{sensed}(q)$ to denote the set of signals sensed in q. The *sensing cost* of a state $q \in Q$ is $\mathit{scost}(q) = |\mathit{sensed}(q)|$.[d]

[d]We note that, alternatively, one could define the *sensing level* of states, with $\mathit{slevel}(q) = \frac{|\mathit{sensed}(q)|}{|P|}$. Then, for all states q, we have that $\mathit{slevel}(q) \in [0, 1]$. All our results hold also for this definition, simply by dividing the sensing cost by $|P|$.

Consider a deterministic automaton $\mathcal{A}$ over $\Sigma = 2^P$ (and over finite or infinite words). For a finite run $r = q_1, \ldots, q_m$ of $\mathcal{A}$, we define the sensing cost of r, denoted $scost(r)$, as $\frac{1}{m}\sum_{i=0}^{m-1} scost(q_i)$. That is, $scost(r)$ is the average number of sensors that $\mathcal{A}$ uses during r. Now, for a finite word w, we define the sensing cost of w in $\mathcal{A}$, denoted $scost_{\mathcal{A}}(w)$, as the sensing cost of the run of $\mathcal{A}$ on w. Finally, the sensing cost of $\mathcal{A}$ is the expected sensing cost of words of length that tends to infinity, where we assume that the letters in Σ are uniformly distributed. Thus, $scost(\mathcal{A}) = \lim_{m\to\infty} |\Sigma|^{-m} \sum_{w:|w|=m} scost_{\mathcal{A}}(w)$. Note that the definition applies to automata on both finite and infinite words, and that it independent of the acceptance condition of A. A-priori, it is not clear that $scost(\mathcal{A})$ is well defined, i.e., that the limit above always exist. As we shall show, however, the limit does always exist.

Two DFAs may recognize the same language and have different sensing costs. In fact, as we demonstrate in Example 1 below, in the case of infinite words two different minimal automata for the same language may have different sensing costs.

For a language L of finite or infinite words, the sensing cost of L, denoted $scost(L)$ is the minimal sensing cost required for recognizing L by a deterministic automaton. Thus, $scost(L) = \inf_{\mathcal{A}:L(\mathcal{A})=L} scost(\mathcal{A})$. For the case of infinite words, we allow $\mathcal{A}$ to be a deterministic automaton of any type. In fact, as we shall see, unlike the case of succinctness, the sensing cost is independent of the acceptance condition used.

Example 1. Let $P = \{a\}$. Consider the language $L \subseteq (2^{\{a\}})^\omega$ of all words with infinitely many a and infinitely many $\neg a$. In the following Fig. 1 we present two minimal DBAs for L with different sensing costs.

While all the states of the second automaton sense a, thus its sensing cost is 1, the signal a is not sensed in all the states of the first automaton, thus its sensing cost is strictly smaller than 1 (to be precise, it is $\frac{4}{5}$, as we shall see in Example 8).

Remark 2. Our study of sensing considers deterministic automata. The notion of sensing is less natural in the nondeterministic setting. From a conceptual point of view, we want to capture the number of sensors required for an actual implementation for recognizing the language. Technically, guesses can reduce the number of required sensors. To see this, take $P = \{a\}$ and consider the language $L = \texttt{True}^* \cdot a$. A DFA for L needs two states, both sensing a. An NFA for L can guess the position of the letter before the last one, where it moves to the only state that senses a. The sensing cost of such an NFA is 0 (for any reasonable extension of the definition of cost on NFAs).

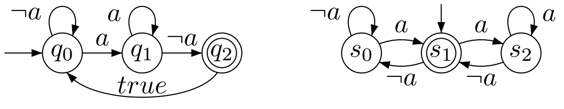

Fig. 1. Two minimal DBAs for L with different sensing costs.

2.4. *Probability, Markov chains, and Markov decision processes*

A Markov chain $\mathcal{M} = \langle S, P \rangle$ 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 [9], 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 $\pi_C(q) = \lim_{m\to\infty} \frac{E_m^C(q)}{m}$, where $E_m^C(q)$ is the average number of occurrences of q in a run of $M_{\mathcal{A}}$ of length m that starts anywhere in C [9]. 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 $\rho : \mathcal{C} \to [0,1]$ that maps each ergodic SCC C of $\mathcal{A}$ to the probability that $M_{\mathcal{A}}$ eventually reaches C, starting from the initial state. We can now define the *limiting distribution* $\pi : Q \to [0,1]$, as

$$\pi(q) = \begin{cases} 0 & \text{if } q \text{ is transient,} \\ \pi_C(q)\rho(C) & \text{if } q \text{ is in some } C \in \mathcal{C}_e. \end{cases}$$

Note that $\sum_{q \in Q} \pi(q) = 1$, and that if P is the matrix describing the transitions of $M_{\mathcal{A}}$ and π is viewed as a vector in $[0,1]^Q$, then $\pi P = \pi$ and in fact $\pi = \lim_{n\to\infty} \frac{1}{n} \sum_{m=0}^{n} v^0 P^m$ where v^0 is a vector with 1 in the coordinate corresponding q_0 and 0 in the other coordinates [9]. The formulations in [9] imply that the stationary, SCC-rechability, and 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 3. *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, $\pi(q) = \lim_{m\to\infty} \frac{E_m(q)}{m}$.*

Proof. Let $q \in Q$, and consider a random infinite run r in $M_{\mathcal{A}}$. If q is transient, then it is easy to see that $\lim_{m\to\infty} \frac{1}{m} E_m(q) = 0 = \pi(q)$, because with probability 1, the state q does not appear after some point in r. Otherwise, let $C \in \mathcal{C}_e$ be the ergodic SCC of q. The probability that r reaches C is given by $\rho(C)$. By the law of total expectation, and since q is reachable only if r reaches C, we have that $E_m(q) = \rho(C)E^C_{m-t}$ where t is the expected time by which r reaches C. Thus, $\lim_{m\to\infty} \frac{E_m(q)}{m} = \rho(C) \lim_{m\to\infty} \frac{E^C_{m-t}}{m} = \rho(C) \lim_{m\to\infty} \frac{E^C_m}{m} = \rho(C)\pi_C(q)$. $\square$

A Markov decision process (MDP) is $\mathcal{M} = \langle S, s_0, (A_s)_{s\in S}, \mathrm{P}, cost\rangle$ where S is a finite set of states, $s_0 \in S$ is an initial state, and A_s is a finite set of actions that are available in state $s \in S$. Let $A = \bigcup_{s\in S} A_s$. Then, $\mathrm{P} : S \times A \times S \nrightarrow [0, 1]$ is a partial transition probability function, defining for every two states $s, s' \in S$ and action $a \in A_s$, the probability of moving from s to s' when action a is taken. Accordingly, $\sum_{s'\in S} \mathrm{P}(s, a, s') = 1$. Finally, $cost : S \times A \nrightarrow \mathbb{N}$ is a partial cost function, assigning each state s and action $a \in A_s$, the cost of taking action a in state s.

An MDP can be thought of as a game between a player who chooses the actions and nature, which acts stochastically according to the transition probabilities.

A *policy* for an MDP $\mathcal{M}$ is a function $f : S^* \times S \to A$ that outputs an action given the history of the states, such that for $s_0, \ldots, s_n$ we have $f(s_0, \ldots, s_n) \in A_{s_n}$. Policies correspond to the strategies of the player. The *cost* of a policy f is the expected average cost of a random walk in $\mathcal{M}$ in which the player proceeds according to f. Formally, for $m \in \mathbb{N}$ and for a sequence of states $\tau = s_0, \ldots, s_{m-1}$, we define $\mathrm{P}_f(\tau) = \prod_{i=1}^{m-1} \mathrm{P}(s_{i-1}, f(s_0 \cdots s_{i-1}), s_i)$. Next, let $cost_m(f, \tau) = \frac{1}{m}\sum_{i=1}^{m} cost(s_i, f(s_1 \cdots s_i))$ and we define the cost of f as $cost(f) = \liminf_{m\to\infty} \frac{1}{m} \sum_{\tau:|\tau|=m} cost_m(f, \tau) \cdot \mathrm{P}_f(\tau)$.

A policy is *memoryless* if it depends only on the current state. We can describe a memoryless policy by $f : S \to A$. A memoryless policy f induces a Markov chain $\mathcal{M}^f = \langle S, P_f\rangle$ with $P_f(s, s') = \mathrm{P}(s, f(s), s')$. Let π be the limiting distribution of $\mathcal{M}^f$. It is not hard to prove (see e.g., [19]) that $cost(f) = \sum_{s\in S} \pi_s cost(s, f(s))$. Let $cost(\mathcal{M}) = \inf\{cost(f) : f \text{ is a policy for } \mathcal{M}\}$. That is, $cost(M)$ is the expected cost of a game played on $\mathcal{M}$ in which the player uses an optimal policy.

Theorem 4. *Consider an MDP $\mathcal{M}$. Then, $cost(\mathcal{M})$ can be attained by a memoryless policy, which can be computed in polynomial time.*

2.5. *Computing the sensing cost of an automaton*

Consider a deterministic automaton $\mathcal{A} = \langle 2^P, Q, \delta, q_0, \alpha\rangle$. The definition of $scost(\mathcal{A})$ by means of the expected sensing cost of words of length that tends to infinity does not suggest an algorithm for computing it. In this section we show that the definition coincides with a definition that sums the costs of the states in $\mathcal{A}$, weighted according to the limiting distribution, and show that this implies a polynomial-time algorithm for computing $scost(\mathcal{A})$. This also shows that the cost is well-defined for all automata.

Theorem 5. *For all automata $\mathcal{A}$, we have $scost(\mathcal{A}) = \sum_{q\in Q} \pi(q) \cdot scost(q)$, where π is the limiting distribution of $\mathcal{A}$.*

Proof. By Lemma 3, we have $\pi(q) = \lim_{m\to\infty} \frac{E_m(q)}{m}$, where $E_m(q)$ is the expected number of occurrences of q in a random m-step run. This can be restated in our case as $\pi(q) = \lim_{m\to\infty} \frac{1}{m|\Sigma|^m} \sum_{w:|w|=m} Occ_w(q)$, where $Occ_w(q)$ is the number of occurrences of q in the run of $\mathcal{A}$ on w. By definition, $scost(\mathcal{A}) = \lim_{m\to\infty} |\Sigma|^{-m} \sum_{w:|w|=m} scost_{\mathcal{A}}(w)$, and also $scost_{\mathcal{A}}(w) = \sum_{q\in Q} scost(q)\cdot Occ_w(q)$. From this, we get

$$\begin{aligned} scost(\mathcal{A}) &= \lim_{m\to\infty} |\Sigma|^{-m} \sum_{w:|w|=m} \sum_{q\in Q} scost(q) \cdot Occ_w(q) \\ &= \sum_{q\in Q} scost(q) \cdot \lim_{m\to\infty} |\Sigma|^{-m} \sum_{w:|w|=m} Occ_w(q) = \sum_{q\in Q} scost(q) \cdot \pi(q). \end{aligned}$$ □

Remark 6. It is not hard to see that if $\mathcal{A}$ is strongly connected, then π is the unique stationary distribution of $M_{\mathcal{A}}$ and is independent of the initial state of $\mathcal{A}$. Accordingly, $scost(\mathcal{A})$ is also independent of $\mathcal{A}$'s initial state in this special case.

Theorem 7. *Given an automaton $\mathcal{A}$, the sensing cost $scost(\mathcal{A})$ can be calculated in polynomial time.*

Proof. By Theorem 5, we have that $scost(\mathcal{A}) = \sum_{q\in Q} \pi(q)\cdot scost(q)$, where π is the limiting distribution of $\mathcal{A}$. By the definition of π, we have that $\pi(q) = \pi_C(q)\rho(C)$, if q is in some $C \in \mathcal{C}_e$. Otherwise, $\pi(q) = 0$. Hence, the computational bottleneck is the calculation of the SCC-reachability distribution $\rho : C \to [0,1]$ and the stationary distributions π_C for every $C \in \mathcal{C}_e$. It is well known that both can be computed in polynomial time via classic algorithms on matrices. For completeness, we give the details here.

The stationary distribution π_C of each ergodic SCC C can be computed in polynomial time by solving a system of linear equations. We show that the SCC-reachability distribution $\rho : \mathcal{C} \to [0,1]$ can also be calculated in polynomial time. First, if the initial state q_0 is in an ergodic SCC, the result is trivial. Otherwise, we proceed as follows. We associate with $\mathcal{A}$ the Markov chain $M'_{\mathcal{A}}$, in which we contract each ergodic SCC of $\mathcal{A}$ to a single state. That is, $M'_{\mathcal{A}}$ is obtained from $M_{\mathcal{A}}$ by replacing each $C \in \mathcal{C}_e$ by a single state q_C. Notice that $M'_{\mathcal{A}}$ is an *absorbing* Markov chain, thus it reaches a sink state with probability 1. Indeed, the probability of reaching an ergodic SCC in $M_{\mathcal{A}}$ is 1, and every SCC in $M_{\mathcal{A}}$ becomes a sink state in $M'_{\mathcal{A}}$.

By indexing the rows and columns in the transition matrix of $M'_{\mathcal{A}}$ such that transient states come before ergodic states, we can put the matrix in a normal form

$\begin{pmatrix} T & E \\ 0 & I \end{pmatrix}$, where T describes the transitions between transient states, E from transient to ergodic states, and I is the identity matrix of size $|\mathcal{C}_e|$. Note that, indeed, there are no transitions from ergodic states to transient ones, which explains the 0 matrix in the bottom left, and that I captures the fact the ergodic states are sinks. By [9], the entry at coordinates (q_t, q_C) in the matrix $B = (I - T)^{-1}E$ is the probability of reaching the sink q_C starting from the transient state q_t. Therefore, for every $C \in \mathcal{C}_e$, we have that $\rho(C) = B_{(q_0,q_C)}$. □

Example 8. Recall the first DBA described in Example 1. Its limiting distribution is $\pi(q_0) = \pi(q_1) = \frac{2}{5}$, $\pi(q_2) = \frac{1}{5}$. Accordingly, its cost is $1 \cdot \frac{2}{5} + 1 \cdot \frac{2}{5} + 0 \cdot \frac{1}{5} = \frac{4}{5}$.

3. The Sensing Cost of Regular Languages of Finite Words

In this section, we study the setting of finite words. We show that there, sensing minimization goes with size minimization, which makes things clean and simple, as size minimization for DFAs is a feasible and well-studied problem. We also study theoretical properties of sensing. In Sec. 3.1, we show that, surprisingly, abstraction of signals may actually increase the sensing cost of a language, and in Sec. 3.2 we study the effect of classical operations on regular languages on their sensing cost.

Consider a regular language $L \subseteq \Sigma^*$, with $\Sigma = 2^P$. Recall that the residual automaton $\mathcal{R}_L = \langle \Sigma, \langle L \rangle, \Delta_L, [\epsilon], \alpha \rangle$ is the minimal-size DFA that recognizes L. We claim that $\mathcal{R}_L$ also minimizes the sensing cost of L.

Lemma 9. *Consider a regular language $L \subseteq \Sigma^*$. For every DFA $\mathcal{A}$ with $L(\mathcal{A}) = L$, we have that $scost(\mathcal{A}) \geq scost(\mathcal{R}_L)$.*

Proof. Consider a word $u \in \Sigma^*$. After reading u, the DFA $\mathcal{R}_L$ reaches the state $[u]$ and the DFA $\mathcal{A}$ reaches a state q with $L(\mathcal{A}^q) = u^{-1}L$. Indeed, otherwise we can point to a word with prefix u that is accepted only in one of the DFAs. We claim that for every state $q \in Q$ such that $L(\mathcal{A}^q) = u^{-1}L$, it holds that $sensed([u]) \subseteq sensed(q)$. To see this, consider a signal $p \in sensed([u])$. By definition, there exists a set $S \subseteq P$ and words u_1 and u_2 such that $([u], S\backslash\{p\}, [u_1]) \in \Delta_L$, $([u], S \cup \{p\}, [u_2]) \in \Delta_L$, yet $[u_1] \neq [u_2]$. By the definition of $\mathcal{R}_L$, there exists $z \in (2^P)^*$ such that, w.l.o.g, $z \in u_1^{-1}L \backslash u_2^{-1}L$. Hence, as $L(\mathcal{A}^q) = u^{-1}L$, we have that $\mathcal{A}^q$ accepts $(S\backslash\{p\}) \cdot z$ and rejects $(S \cup \{p\}) \cdot z$. Let $\delta_{\mathcal{A}}$ be the transition function of $\mathcal{A}$. By the above, $\delta_{\mathcal{A}}(q, S\backslash\{p\}) \neq \delta_{\mathcal{A}}(q, S \cup \{p\})$. Therefore, $p \in sensed(q)$, and we are done. Now, $sensed([u]) \subseteq sensed(q)$ implies that $scost(q) \geq scost([u])$.

Consider a word $w_1 \cdots w_m \in \Sigma^*$. Let $r = r_0, \ldots, r_m$ and $[u_0], \ldots, [u_m]$ be the runs of $\mathcal{A}$ and $\mathcal{R}_L$ on w, respectively. Note that for all $i \geq 0$, we have $u_i = w_1 \cdot w_2 \cdots w_i$. For all $i \geq 0$, we have that $L(\mathcal{A}^{r_i}) = u_i^{-1}L$, implying that then $scost(r_i) \geq scost([u_i])$. Hence, $scost_{\mathcal{A}}(w) \geq scost_{\mathcal{R}_L}(w)$. Since this holds for every word in Σ^*, it follows that $scost(\mathcal{A}) \geq scost(\mathcal{R}_L)$. □

Since $L(\mathcal{R}_L) = L$, then $scost(L) \leq scost(\mathcal{R}_L)$. This, together with Lemma 9, enables us to conclude the following.

Theorem 10. *For every regular language $L \subseteq \Sigma^*$, we have $scost(L) = scost(\mathcal{R}_L)$.*

Finally, since DFAs can be size-minimized in polynomial time, Theorems 7 and 10 imply we can efficiently minimize also the sensing cost of a DFA and calculate the sensing cost of its language:

Theorem 11. *Given a DFA $\mathcal{A}$, the problem of computing $scost(L(\mathcal{A}))$ can be solved in polynomial time.*

3.1. *On monotonicity of sensing*

The example in Remark 2 suggests that there is a trade-off between guessing and sensing. Consider a DFA $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, with $\Sigma = 2^P$. For a state $q \in Q$ and a signal $p \in P$, let $\mathcal{A}_{q\downarrow p}$ be the NFA obtained from $\mathcal{A}$ by ignoring p in q. Thus, in state q, the NFA $\mathcal{A}_{q\downarrow p}$ guesses the value of p and proceeds to all the successors that are reachable with some value. Note that the guess introduces nondeterminism; Since we consider sensing in deterministic automata, we formally define $\mathcal{A}_{q\downarrow p}$ as the result of the determinization of the NFA above. We thus have $\mathcal{A}_{q\downarrow p} = \langle \Sigma, 2^Q, \{q_0\}, \delta', \alpha' \rangle$, where for every state $T \in 2^Q$ and letter $S \in 2^P$, we define $\delta'(T, S) = \bigcup_{t \in T} \delta(t, S)$ if $q \notin T$, and $\delta'(T, S) = \delta(q, S \backslash \{p\}) \cup \delta(q, S \cup \{p\}) \cup \bigcup_{t \in T \backslash \{q\}} \delta(t, S)$ if $q \in T$. Also, a state $T \subseteq Q$ is in α' iff $T \cap \alpha \neq \emptyset$. It is easy to see that $L(\mathcal{A}) \subseteq L(\mathcal{A}_{q\downarrow p})$. Since $\mathcal{A}_{q\downarrow p}$ is obtained from $\mathcal{A}$ by giving up some of its sensing, it is tempting to think that $scost(L(\mathcal{A}_{q\downarrow p})) \leq scost(L(\mathcal{A}))$. As we now show, however, sensing is not monotone. For two languages L and L', we say that L' is an abstraction of L if there is a DFA $\mathcal{A}$ such that $L(\mathcal{A}) = L$ and there is a state q and a signal p of $\mathcal{A}$ such that $L' = L(\mathcal{A}_{q\downarrow p})$.

Theorem 12. *Sensing is not monotone. That is, there is a language L and an abstraction L' of L such that $scost(L) \leq scost(L')$.*

Proof. Let $P = \{a, b, c\}$. Consider the language $L = a \cdot \texttt{True}^* \cdot b + (\neg a) \cdot \texttt{True}^* \cdot c$. It is not hard to see that $scost(L) = 1$. Indeed, a DFA for L has to sense a in its initial state and then has to always sense either b (in case a appears in the first letter) or c (otherwise).

Giving up the sensing of a in the initial state of a DFA for L we end up with the language $L' = (\texttt{True})^+ \cdot (b \vee c)$. It is not hard to see that $scost(L') = 2$. Indeed, every DFA for L' has to almost always sense both b and c. $\square$

We conclude that replacing a sensor with non-determinism may actually result in a language for which we need more sensors. This corresponds to the known fact that abstraction of automata may result in bigger (in fact, exponentially bigger) DFAs [2]. Also, while the above assumes an abstraction that over-approximates the original language, a dual argument could show that under-approximating the language (that is, defining $\mathcal{A}_{q\downarrow p}$ as a universal automaton) may result in a language with higher sensing cost.

3.2. *Operations on regular languages and their sensing cost*

It is well known that regular languages are closed under union, concatenation, and complementation. In this section, we study the effect of these operations on the sensing cost. We start with complementation. For every regular language L, a DFA for $comp(L) = \Sigma^*\backslash L$ can be obtained from a DFA for L by complementing the set of accepting states. In particular, this holds for $\mathcal{R}_L$, implying the following.

Lemma 13. *For every regular language L, we have that $scost(L) = scost(comp(L))$.*

Next, we consider the union of two regular languages.

Lemma 14. *For every pair of regular languages $L_1, L_2 \subseteq (2^P)^*$, we have $scost(L_1 \cup L_2) \leq scost(L_1) + scost(L_2)$. Moreover, this bound is tight.*

Proof. Consider the minimal DFAs $\mathcal{A}_1 = \langle 2^P, Q^1, \delta^1, q_0^1, \alpha^1\rangle$ and $\mathcal{A}_2 = \langle 2^P, Q^2, \delta^2, q_0^2, \alpha^2\rangle$ for L_1 and L_2, respectively. Let $\mathcal{B} = \langle 2^P, Q^1 \times Q^2, \delta, (q_0^1, q_0^2), (\alpha^1 \times Q^2) \cup (Q^1 \times \alpha^2)\rangle$ be their product DFA. Note that $L(\mathcal{B}) = L_1 \cup L_2$. We claim that for every state $\langle q, s\rangle \in Q^1 \times Q^2$, we have that $sensed(\langle q, s\rangle) \subseteq sensed(q) \cup sensed(s)$. Indeed, if $p \notin sensed(q) \cup sensed(s)$, then for every set $S \subseteq P\backslash\{p\}$, it holds that $\delta^1(q, S) = \delta^1(q, S \cup \{p\})$ and $\delta^2(s, S) = \delta^2(s, S \cup \{p\})$. Thus, $\delta(\langle q, s\rangle, S) = \delta(\langle q, s\rangle, S \cup \{p\})$, so $p \notin sensed\langle q, s\rangle$. We thus have that $scost(\langle q, s\rangle) \leq scost(q) + scost(s)$.

It follows that for every word $w \in (2^{I\cup O})^*$, we have that $scost_{\mathcal{B}}(w) \leq scost_{\mathcal{A}_1}(w) + scost_{\mathcal{A}_2}(w)$. Indeed, in every state in the run of $\mathcal{B}$ on w, the sensing is at most the sum of the sensings in the corresponding states in the runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ on w. Since this is true for every word in Σ^*, then taking the limit of the average cost yields the result.

As for tightness, a trivial example when L_1 is $\emptyset$. Then, $scost(L_1) = 0$ and $scost(L_1 \cup L_2) = scost(L_2) = scost(L_1) + scost(L_2)$. For a nontrivial example, take $P = \{p_1, p_2, \ldots, p_{2n}\}$, and let $L_1 = \texttt{True}^* \cdot (\bigvee_{1\leq i\leq n} p_{2i-1})$ and $L_2 = \texttt{True}^* \cdot (\bigvee_{1\leq i\leq n} p_{2i})$. Note that $scost(L_1) = scost(L_2) = n$. Indeed, a DFA for L_1 has to always sense $p_1, p_3, \ldots, p_{2n-1}$ in order to verify that the last letter is one of them, and similarly for L_2 and $p_2, p_4, \ldots, p_{2n}$. Also, $L_1 \cup L_2 = \texttt{True}^* \cdot (\bigvee_{p\in P} p)$, which has sensing cost $2n$. □

The following lemma shows that in general, $scost(L_1 \cup L_2)$ cannot be bounded from below using $scost(L_1)$ and $scost(L_2)$.

Lemma 15. *For every set P of signals, there are languages $L_1, L_2 \subseteq (2^P)^*$ with $scost(L_1) = scost(L_2) = |P|$ but $scost(L_1 \cup L_2) = 0$.*

Proof. Given P, let $L_1 = \mathtt{True}^* \cdot (\bigvee_{p\in P} p)$ and $L_2 = \mathtt{True}^* \cdot (\bigwedge_{p\in P} \neg p)$. It is not hard to see that $scost(L_1) = scost(L_2) = |P|$. Indeed, DFAs for L_1 and L_2 have to always sense all signals. On the other hand, $L_1 \cup L_2 = \mathtt{True}^*$, and thus $scost(L_1 \cup L_2) = 0$. □

Note that since sensing cost is preserved by complementation, the above bound on union of languages can be lifted to intersection. Indeed, $scost(L_1 \cap L_2) = scost(comp(L_1 \cap L_2)) = scost(comp(L_1) \cup comp(L_2)) \leq scost(comp(L_1)) + scost(comp(L_2)) = scost(L_1) + scost(L_2)$. Overall, we obtain the same bound as for union, that is $scost(L_1 \cap L_2) \leq scost(L_1) + scost(L_2)$.

We now consider the concatenation of two languages. The following lemma shows that the sensing level may increase from 0 to $|P|$ when concatenating languages, or conversely may decrease from $|P|$ to 0. This indicates that there is in general no way to infer an upper or lower bound on bound on $scost(L_1 \cdot L_2)$ from $scost(L_1)$ and $scost(L_2)$.

Lemma 16. (1) *There are languages $L_1, L_2 \subseteq (2^P)^*$ such that $scost(L_1) = scost(L_2) = 0$, yet $scost(L_1 \cdot L_2) = |P|$.*
(2) *There are languages $L_1, L_2 \subseteq (2^P)^*$ such that $scost(L_1) = scost(L_2) = |P|$, yet $scost(L_1 \cdot L_2) = 0$.*

Proof. Given P, let $\varphi = \bigvee_{p\in P} p$. Thus, φ stands for all letters in which one of the signals in P holds.

For the first claim, consider the languages $L_1 = \mathtt{True}^*$ and $L_2 = \varphi$. That is, L_2 contains all words of length 1 whose single letter includes one of the signals in P. It is not hard to see that $scost(L_1) = scost(L_2) = 0$. Indeed, a DFA for L_1 consists of a single accepting sink with no sensing, and a DFA for L_2 has a single ergodic component, which is a rejecting sink with no sensing. On the other hand, $L_1 \cdot L_2 = \mathtt{True}^* \cdot \varphi$, and has to always sense all signals.

For the second claim, consider the languages $L_1 = ((\neg\varphi)^* \cdot \varphi \cdot (\neg\varphi)^* \cdot \varphi)^* \cdot (\neg\varphi)^*$ of words having an even number of letters in φ, and $L_2 = (\neg\varphi)^* + (\neg\varphi)^*\varphi \cdot L_1$ of words having either no φ or an odd number of letters is φ's. We have $scost(L_1) = scost(L_2) = |P|$, as in both cases, the DFA must read each letter to keep track of the parity of the number of letters in φ. In the case of L_2, the ergodic component verifies that the number of letters in φ is odd, as with probability 1, a letter in φ occurs. We show that $L_1 \cdot L_2 = \Sigma^*$. Indeed, $L_1 \cdot L_2 \supseteq (L_1 \cdot \{\epsilon\}) \cup (\{\epsilon\} \cdot L_2) = L_1 \cup L_2$. Since any word has either an even number of letters in φ or an odd number of such letters, we have that $L_1 \cup L_2 = (2^P)^*$. Thus $L_1 \cdot L_2 = (2^P)^*$, and so $scost(L_1 \cdot L_2) = 0$. □

4. The Sensing Cost of ω-Regular Languages

For the case of finite words, we have a very clean picture: minimizing the state space of a DFA also minimizes its sensing cost. In this section we study the case of infinite words. There, the picture is much more complicated. In Example 1 we saw that

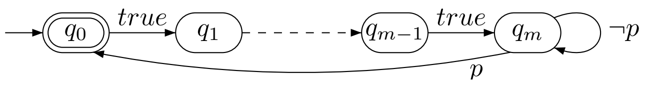

Fig. 2. The DBA $\mathcal{A}_m$.

different minimal DBAs may have a different sensing cost. We start this section by showing that even for languages that have a single minimal DBA, the sensing cost may not be attained by this minimal DBA, and in fact it may be attained only as a limit of a sequence of DBAs.

Example 17. Let $P = \{p\}$, and consider the language L of all words $w_1 \cdot w_2 \cdots$ such that $w_i = \{p\}$ for infinitely many i's. Thus, $L = (\texttt{True}^* \cdot p)^\omega$. A minimal DBA for L has two states. The minimal sensing cost for a two-state DBA for L is $\frac{2}{3}$ (the classical two-state DBA for L senses p in both states and thus has sensing cost 1. By taking $\mathcal{A}_1$ in the sequence we shall soon define we can recognize L by a two-state DBA with sensing cost $\frac{2}{3}$). Consider the sequence of DBAs $\mathcal{A}_m$ appearing in Fig. 2. The DBA $\mathcal{A}_m$ recognizes $(\texttt{True}^{\geq m} \cdot p)^\omega$, which is equivalent to L, yet enables a "lazy" sensing of p. Formally, the stationary distribution π for $\mathcal{A}_m$ is such that $\pi(q_i) = \frac{1}{m+1}$ for $0 \leq i \leq m-1$ and $\pi(q_m) = \frac{2}{m+1}$. In the states $q_0, \ldots, q_{m-1}$ the sensing cost is 0 and in q_m it is 1. Accordingly, $scost(\mathcal{A}_m) = \frac{2}{m+1}$, which tends to 0 as m tends to infinity.

Observe that the residual automaton $\mathcal{R}_L$ for the language $L = (\texttt{True}^* \cdot p)^\omega$ discussed in Example 17 has a single state, and therefore its sensing cost is 0, which happens to be the sensing cost of L. As we show in Sec. 4.1, this is not a coincidence, and in fact the residual automaton can be used to characterize the sensing cost even for ω-regular languages.

4.1. *Characterizing scost(L) by the residual automaton for L*

In this section we state and prove our main result, which characterizes the sensing cost of an ω-regular language by means of the residual automaton for the language:

Theorem 18. *For every ω-regular language $L \subseteq \Sigma^\omega$, we have $scost(L) = scost(\mathcal{R}_L)$.*

The proof is described over the following section. The first direction, showing that $scost(L) \geq scost(\mathcal{R}_L)$, is proved by similar considerations to those used in the proof of Lemma 9 for the setting of finite words.

Lemma 19. *For every ω-regular language $L \subseteq \Sigma^\omega$, we have $scost(L) \geq scost(\mathcal{R}_L)$.*

Proof. We prove that for every DPA $\mathcal{A}$ with $L(\mathcal{A}) = L$, we have that $scost(\mathcal{A}) \geq scost(\mathcal{R}_L)$. Consider a word $w \in \Sigma^\omega$ and a prefix $u \in \Sigma^*$ of w. After reading u, the DPA $\mathcal{R}_L$ reaches the state $[u]$ and the DPA $\mathcal{A}$ reaches a state q with $L(\mathcal{A}^q) = u^{-1}L$.

As in the case of finite words, for every state $q \in Q$ such that $L(\mathcal{A}^q) = u^{-1}L$, it holds that $\mathit{sensed}([u]) \subseteq \mathit{sensed}(q)$, implying that $\mathit{scost}(q) \geq \mathit{scost}([u])$. Now, since this holds for all prefixes u of w, it follows that $\mathit{scost}_{\mathcal{A}}(w) \geq \mathit{scost}_{\mathcal{R}_L}(w)$. Finally, since the latter holds for every word $w \in \Sigma^\omega$, it follows that $\mathit{scost}(\mathcal{A}) \geq \mathit{scost}(\mathcal{R}_L)$.

Note that the arguments in the proof are independent of the acceptance condition of $\mathcal{A}$ and apply also to stronger acceptance conditions, such as the Muller acceptance condition. □

Our main effort is to prove that $\mathit{scost}(L) \leq \mathit{scost}(\mathcal{R}_L)$. To show this, we construct, given a DPA $\mathcal{A}$ such that $L(\mathcal{A}) = L$, a sequence $(\mathcal{B}_n)_{n\geq 1}$ of DPAs such that $L(\mathcal{B}_n) = L$ for every $n \geq 1$, and $\lim_{n\to\infty} \mathit{scost}(\mathcal{B}_n) = \mathit{scost}(\mathcal{R}_L)$.

Broadly, the idea behind the construction is as follows. When reading a word, the DPA $\mathcal{B}_n$ mimics the operation of $\mathcal{A}$ for a certain duration. At some point, it proceeds to read n letters in a component that mimics $\mathcal{R}_L$, without effecting the acceptance of the word. It then goes back to mimicking $\mathcal{A}$, and so on. As n grows, $\mathcal{B}_n$ spends more time mimicking $\mathcal{R}_L$, and hence its cost tends to $\mathit{scost}(\mathcal{R}_L)$. However, there are now two contradicting goals to achieve: on the one hand, since $\mathcal{R}_L$ cannot be used to recognize L, we must ensure that $\mathcal{B}_n$ retains enough information, and spends enough time mimicking $\mathcal{A}$, in order to correctly accept or reject the word. On the other hand, we must bound the expected duration that $\mathcal{B}_n$ spends mimicking $\mathcal{A}$, otherwise the cost does not tend to $\mathit{scost}(\mathcal{R}_L)$. Finding the balance between these two requires a delicate construction, and a careful analysis of the structure of DPAs.

A key idea in this construction is that after some time spent in $\mathcal{R}_L$, we "lost track" of where we are exactly in $\mathcal{A}$. In order to regain some information when mimicking $\mathcal{A}$, we wait until an exact sequence of parity ranks (noted u_k in the proof) occurs, that guarantees that in the actual run of $\mathcal{A}$, a maximal index occured. This is possible thanks to a structural characterization of rank-optimal DPAs inspired from [17]. Since the length of u_k does not depend on n, with probability 1 this precise sequence u_k will be witnessed, and the automaton $\mathcal{B}_n$ will switch back to its $\mathcal{R}_L$ component.

We note that since the DPAs $\mathcal{B}_n$ have the same acceptance condition as $\mathcal{A}$, there is no trade-off between sensing cost and acceptance condition. More precisely, if L can be recognized by a DPA with parity ranks $[i, j]$ (in particular, if L is DBA-recognizable), then the sensing cost for $L(\mathcal{A})$ can be obtained by a DPA with parity ranks $[i, j]$.

Definition 20. A DPA $\mathcal{A}$ with ranks $[i, j]$ is *canonical* if

- it is strongly connected
- it is impossible to recognize $L(\mathcal{A})$ with a DPA $\mathcal{A}' \neq \mathcal{A}$ with ranks $[i, j]$ obtained from $\mathcal{A}$ by increasing the ranks of some states.

Lemma 21. *If L is recognized by a strongly connected DPA, then it is recognized by a canonical one, using possibly less ranks.*

Proof. Let $\mathcal{A}$ be a strongly connected DPA for L, with states Q and ranks $[i,j]$. A rank can be assigned to each state of Q via a ranking vector $v \in [i,j]^Q$. Let $\mathcal{A}_v$ be the automaton $\mathcal{A}$ where ranks are defined by the vector v. Let $V = \{v \in [i,j]^Q \mid L(\mathcal{A}_v) = L\}$, which is non-empty since $\mathcal{A} = \mathcal{A}_v$ for some v. Let v_m be a component-wise maximal element of V (there can be several choices for v_m). The automaton A_{v_m} is a canonical DPA for L. Notice that A_{v_m} possibly uses less ranks than $\mathcal{A}$. $\square$

We first assume that L is recognized by a strongly connected DPA. We will later show how to drop this assumption.

By Lemma 21, let $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, \alpha_{\mathcal{A}} \rangle$ be a canonical DPA for L. We can assume the lower rank of $\mathcal{A}$ is 0 or 1, by shifting ranks if necessary. Moreover, if the lower rank is 1, we consider the complement DPA, for which the lower rank is 0, and which is still canonical. Since canonical DPAs can be complemented by dualizing the acceptance condition, their sensing cost is preserved under complementation, so reasoning about the complemented DPA is sound. Thus from now on, we assume that $\mathcal{A}$ has ranks $[0,k]$. For $0 \leq i \leq k$, a cycle in $\mathcal{A}$ is called an *i-loop* if the maximal rank along the cycle is i. For $0 \leq i \leq j \leq k$, an *$[i,j]$-flower* is a state $q_{❀} \in Q$ such that for every $i \leq r \leq j$, there is an r-loop that goes through $q_{❀}$.

The following is an adaptation of a result from [17] to canonical DPAs:

Lemma 22. *Consider a canonical DPA* $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, \alpha_{\mathcal{A}} \rangle$ *with ranks* $[0,k]$. *Then, any state* $q_{❀}$ *of rank* 0 *is a* $[0,k]$*-flower.*

Proof. Let q be a state of rank 0, and assume q is not a $[0,k]$-flower. Since $\mathcal{A}$ is strongly connected, there must be a k-loop containing q. If there is no 0-loop containing q, then we can replace the rank of q by 1 in $\mathcal{A}$ without changing the accepted language. This contradicts the fact that $\mathcal{A}$ is canonical. This means there is a rank $i \in [1, k-1]$ such that there is no i-loop containing q.

Let $G_{<i}$ be the graph of $\mathcal{A}$ restricted to states with rank in $[0, i-1]$. Let C_q be the strongly connected component of q in G_i. Since there is a 0-loop containing q, C_q is not empty. We prove that adding 2 to the ranks of states in C_q does not change the language of $\mathcal{A}$, leading to a contradiction with the fact that $\mathcal{A}$ is canonical. Indeed, let $\mathcal{A}'$ be the automaton where ranks of states of C_q are lifted up by 2, and consider a run ρ' of $\mathcal{A}'$ and the corresponding run ρ of $\mathcal{A}$. We show that ρ' is accepting if and only if ρ is accepting. If ρ does not contain infinitely many states from C_q, or eventually stays in C_q, then the result is trivial. It remains to treat the case where ρ visits C_q as well as its complement infinitely many times. Let π be a path in $\mathcal{A}$ from C_q to C_q containing a state not in C_q. Then π can be extended to a loop containing q by appending a prefix and a suffix in C_q, since C_q is strongly connected. The maximal rank of π is at least $i+1$, since it cannot be $< i$ (that would imply π is contained in C_q) and it cannot be i (that would imply that π is an i-loop containing q). So any path leaving C_q and going back to it must contain a rank at least $i+1$. This implies that ρ contains infinitely many ranks above $i+1$, and therefore ρ is accepting if and only if ρ' is accepting, since only ranks below $i+1$ are modified. This shows that $L(\mathcal{A}') = L(\mathcal{A})$, contradicting the fact that $\mathcal{A}$ is canonical. □

Let $\Omega = [0, k]$, and let $\mathcal{A}$ be as in Lemma 22, and $q_{❀}$ be a Ω-flower in $\mathcal{A}$. For a word $w \in \Sigma^*$, let $\rho = s_1, s_1, \ldots, s_n$ be the run of $\mathcal{A}$ on w. If ρ ends in $q_{❀}$, we define the *$q_{❀}$-loop-abstraction of* w to be the rank-word $\text{abs}(w) \in \Omega^*$ of maximal ranks between successive visits to $q_{❀}$. Formally, let $w = y_0 \cdot y_1 \ldots y_t$ be a partition of w such that $\mathcal{A}$ visits the state $q_{❀}$ after reading the prefix $y_0 \ldots y_j$, for all $0 \le j \le t$, and does not visit $q_{❀}$ in other positions. Then, $\text{abs}(y_i)$, for $0 \le i \le t$, is the maximal rank read along y_i, and $\text{abs}(w) = \text{abs}(y_0) \cdot \text{abs}(y_1) \ldots \text{abs}(y_t)$. Recall that $\mathcal{R}_L = \langle \Sigma, \langle L \rangle, \Delta_L, [\epsilon], \alpha \rangle$, where $\langle L \rangle$ are the equivalence classes of the right-congruence relation on L, thus each state $[u] \in \langle L \rangle$ is associated with the language $u^{-1}L$ of words w such that $uw \in L$. We define a function $\varphi : Q \to \langle L \rangle$ that maps states of $\mathcal{A}$ to languages in $\langle L \rangle$ by $\varphi(q) = L(\mathcal{A}^q)$. Observe that φ is onto. We define a function $\gamma : \langle L \rangle \to Q$ that maps languages in $\langle L \rangle$ to states of $\mathcal{A}$ by arbitrarily choosing for every language $u^{-1}L \in \langle L \rangle$ a state in $\varphi^{-1}(u^{-1}L)$ (clearly γ is independent of u, and is therefore well-defined).

We define a sequence of words $u_0, \ldots, u_k \in \Omega^*$ as follows. The definition proceeds by an induction. Let $M = |Q| + 1$. First, $u_0 = 0^M$. Then, for $0 < i \le k$, we have $u_i = (i \cdot u_{i-1})^{M-1} \cdot i$. For example, if $|Q| = 2$, then $u_0 = 000$, $u_1 = 100010001$, $u_2 = 2100010001210001000012$, and so on. Let $\mathcal{P}$ be a DFA that accepts a (finite) word $w \in \Sigma^*$ iff the run of $\mathcal{A}$ on w ends in $q_{❀}$ and u_k is a suffix of $abs(w)$, for the word $u_k \in \Omega^*$ defined above.

We now turn to describe how to construct $\mathcal{P}$. Intuitively, this is done by combining a DFA over that alphabet Ω that recognizes $\Omega^* \cdot u_k$ with a DFA with state space $Q \times \Omega$ that records the highest rank visited between successive visits to $q_{❀}$ and thus abstracts words in Σ^*.

Let $\mathcal{H}_k = \langle \Omega, Q', q_0', \Delta', \alpha' \rangle$ be the minimal DFA that recognizes the language $\Omega^* \cdot u_k$. We can define $\mathcal{H}_k$ so that α' contains a single state q'_{acc}. Indeed, there is a single accepting Myhill-Nerode class of the language $\Omega^* \cdot u_k$.

Let $\mathcal{H}$ be the DFA with state space $Q \times \Omega$ and alphabet Σ that maintains in its state the highest rank seen since the last occurrence of $q_{❀}$ (or since the beginning of the word, if no $q_{❀}$ has been seen) in the run of $\mathcal{A}$ on the word. Thus, $\mathcal{H}$ is in state $\langle q, i \rangle$ iff the current state of $\mathcal{A}$ is q, and the highest rank that was visited by $\mathcal{A}$ since the last visit to $q_{❀}$ is i. Observe that simulating $\mathcal{H}$ when $\mathcal{A}$ is in an r-loop that started from $q_{❀}$, means that the next visit to $q_{❀}$ will make $\mathcal{H}$ reach the state $\langle q_{❀}, r \rangle$.

Formally, $\mathcal{H} = \langle \Sigma, Q \times \Omega, \langle q_0, 0 \rangle, \Delta_{\mathcal{H}}, Q \times \Omega \rangle$, where $\Delta_{\mathcal{H}}$ is defined as follows.

- For every state $\langle q, i \rangle$ where $q \neq q_{❀}$, and for every $\sigma \in \Sigma$, we have $\langle \langle q, i \rangle, \sigma, \langle s, \max\{i, i'\} \rangle \rangle \in \Delta_{\mathcal{H}}$ where s is such that $\langle q, \sigma, s \rangle \in \Delta$, and $i' = \alpha_{\mathcal{A}}(s)$.

- For a state $\langle q_{❀}, i\rangle$ and for $\sigma \in \Sigma$, we have $\langle\langle q_{❀}, i\rangle, \sigma, \langle s, i'\rangle\rangle \in \Delta_{\mathcal{H}}$ where s is such that $\langle q_{❀}, \sigma, s\rangle \in \Delta$, and $i' = \alpha_{\mathcal{A}}(s)$.

We obtain $\mathcal{P}$ by composing $\mathcal{H}$ with $\mathcal{H}_k$ as follows. In every step of a run of $\mathcal{A}$, the DFA $\mathcal{P}$ advances in the DFA $\mathcal{H}$, while the DFA $\mathcal{H}_k$ only advances when we visit $q_{❀}$, and it advances according to the highest rank stored in $\mathcal{H}$.

Formally, $\mathcal{P} = \langle\Sigma, Q_{\mathcal{P}}, t_0, \Delta_{\mathcal{P}}, \{t_{acc}\}\rangle$, where $Q_{\mathcal{P}} = Q \times \Omega \times Q'$, $t_0 = \langle q_0, 0, q'_0\rangle$, $t_{acc} = \langle q_{❀}, k, q'_{acc}\rangle$ and the transition relation is defined as follows. For every state $\langle q, i, s\rangle \in Q_{\mathcal{P}}$ and letter $\sigma \in \Sigma$, we have $\langle\langle q, i, s\rangle, \sigma, \langle q', i', s'\rangle\rangle \in \Delta_{\mathcal{P}}$, where $\langle q', i'\rangle$ is such that $\langle\langle q, i\rangle, \sigma, \langle q', i'\rangle\rangle \in \Delta_{\mathcal{H}}$, and s' is such that $\langle s, i', s'\rangle \in \Delta'$ if $q' = q_{❀}$, while $s' = s$ if $q \neq q_{❀}$.

We can now turn to the construction of the DPAs $\mathcal{B}_n$. Recall that $\mathcal{A} = \langle\Sigma, Q, q_0, \Delta, \alpha_{\mathcal{A}}\rangle$, and let $\mathcal{P} = \langle\Sigma, Q_{\mathcal{P}}, t_0, \Delta_{\mathcal{P}}, \{t_{acc}\}\rangle$. For $n \geq 1$, we define $\mathcal{B}_n = \langle\Sigma, Q_n, \langle q_0, t_0\rangle, \Delta_n, \alpha_n\rangle$ as follows. The states of $\mathcal{B}_n$ are $Q_n = (\langle L\rangle \times \{1, \ldots, n\}) \cup (Q \times (Q_{\mathcal{P}} \setminus \{t_{acc}\}))$, where t_{acc} is the unique accepting state of $\mathcal{P}$. We refer to the two components in the union as the $\mathcal{R}_L$*-component* and the $\mathcal{A}$*-component*, respectively. The transitions of $\mathcal{B}_n$ are defined as follows.

- Inside the $\mathcal{R}_L$-component: for every transition $\langle [u], a, [u']\rangle \in \Delta_L$ and $i \in \{1, \ldots, n-1\}$, there is a transition $\langle([u], i), a, ([u'], i+1)\rangle \in \Delta_n$.
- From the $\mathcal{R}_L$-component to the $\mathcal{A}$-component: for every transition $\langle [u], a, [u']\rangle \in \Delta_L$, there is a transition $\langle([u], n), a, (\gamma([u']), t_0)\rangle \in \Delta_n$.
- Inside the $\mathcal{A}$-component: for all transitions $\langle q, a, q'\rangle \in \Delta$ and $\langle t, a, t'\rangle \in \Delta_{\mathcal{P}}$ with $t' \neq t_{acc}$, there is a transition $\langle(q, t), a, (q', t')\rangle \in \Delta_n$.
- From the $\mathcal{A}$-component to the $\mathcal{R}_L$-component: for all transitions $\langle q, a, q'\rangle \in \Delta$ and $\langle t, a, t_{acc}\rangle \in \Delta_{\mathcal{P}}$, there is a transition $\langle(q, t), a, (\varphi(q'), 1)\rangle \in \Delta_n$.

The acceptance condition of $\mathcal{B}_n$ is induced by that of $\mathcal{A}$. Formally $\alpha_n(q, t) = \alpha_{\mathcal{A}}(q)$, for states $(q, t) \in Q \times Q_{\mathcal{P}}$, and $\alpha_n([u], i) = 0$ for states $([u], i) \in \langle L\rangle \times \{1, \ldots, n\}$.

The idea behind the construction of $\mathcal{B}_n$ is as follows. The automaton $\mathcal{B}_n$ stays in $\mathcal{R}_L$ for n steps, then proceeds to a state in $\mathcal{A}$ with the correct residual language, and simulates $\mathcal{A}$ until the ranks corresponding to the word u_k have been seen. It then goes back to $\mathcal{R}_L$, by projecting the current state of $\mathcal{A}$ onto its residual in $\langle L\rangle$. The bigger n is, the more time a run spends in the $\mathcal{R}_L$-component, making $\mathcal{R}_L$ the more dominant factor in the sensing cost of $\mathcal{B}_n$. As n tends to infinity, the sensing cost of $\mathcal{B}_n$ tends to that of $\mathcal{R}_L$. The technical challenge is to define $\mathcal{P}$ in such a way so that even though the run spends less time in the $\mathcal{A}$ component, we can count on the ranks visited during this short time in order to determine whether the run is

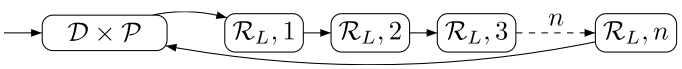

Fig. 3. The DPA $\mathcal{B}_n$.

accepting. We are now going to formalize this intuition, and we start with the most challenging part of the proof, namely the equivalence of $\mathcal{B}_n$ and $\mathcal{A}$. The proof is decomposed into the three Lemmas 23–25. Lemma 23 shows that $\mathcal{B}_n$ correctly keeps track of the residual language. Then, Lemmas 24 and 25 show that $\mathcal{B}_n$ correctly recognizes L. Technically, the latter lemmas are a case split according to whether or not the $\mathcal{R}_L$ component is visited infinitely often.

Lemma 23. *Consider a word $u \in \Sigma^*$ such that the run of $\mathcal{B}_n$ on u reaches the $\mathcal{A}$-component in state $\langle q, t\rangle$. Then, $L(\mathcal{A}^q) = u^{-1}L$.*

Proof. We prove a stronger claim, namely that if the run of $\mathcal{B}_n$ on u ends in the $\mathcal{R}_L$-component in a state $\langle s, i\rangle$, then $s = [u]$, and if the run ends in the $\mathcal{A}$-component in a state $\langle q, t\rangle$, then $L(\mathcal{A}^q) = u^{-1}L$. The proof proceeds by induction on $|u|$ as follows.

For $u = \epsilon$, the claim is trivial, as $\mathcal{B}_n$ starts in $\langle q_0, t_0\rangle$. Consider the word $u \cdot \sigma$ for $u \in \Sigma^*$ and $\sigma \in \Sigma$. By the induction hypothesis, if the run of $\mathcal{B}_n$ on u ends in an $\mathcal{R}_L$ component in state $\langle s, i\rangle$, then $s = [u]$. If $i < n$, then, by the definition of $\mathcal{R}_L$, the next state is $\langle [u \cdot \sigma], i + 1\rangle$, we are done. If $i = n$ then the next state is $\langle \gamma([u \cdot \sigma]), t_0\rangle$. By the definition of γ, we have $L(\mathcal{A}^{\gamma([u\cdot\sigma])}) = (u \cdot \sigma)^{-1}L$, so we are done.

We continue to the case the run of $\mathcal{B}_n$ on u ends in the $\mathcal{A}$-component. If the run ends in a state $\langle p, t\rangle$ such that $\langle t, \sigma, t_{acc}\rangle \notin \Delta_{\mathcal{P}}$, then, by the induction hypothesis, we have that $L(\mathcal{A}^p) = u^{-1}L$. Reading σ, we move to a state $\langle p', t'\rangle$ such that $\langle p, \sigma, p'\rangle \in \Delta$, thus $L(\mathcal{A}^{p'}) = (u\cdot\sigma)^{-1}L$, and we are done. Otherwise, $\langle t, \sigma, t_{acc}\rangle \in \Delta_{\mathcal{P}}$ and the next state of $\mathcal{B}_n$ is $\langle \varphi(p'), 1\rangle$. By the definition of φ, we have $\varphi(p') = [u \cdot \sigma]$, and we are done. □

Lemma 24. *If the run of $\mathcal{B}_n$ on a word $w \in \Sigma^\omega$ visits the $\mathcal{R}_L$-component only finitely many times, then $w \in L$ iff $w \in L(\mathcal{B}_n)$.*

Proof. Let $u \in \Sigma^*$ be a prefix of w such that the run of $\mathcal{B}_n$ on w stays forever in the $\mathcal{A}$-component after reading u. Let $(q, t) \in Q_n$ be the state reached by $\mathcal{B}_n$ after reading u. By Lemma 23, we have $L(\mathcal{A}^q) = u^{-1}L$. Since the run of $\mathcal{B}_n$ from (q, t) stays in the $\mathcal{A}$-components where it simulates the run of $\mathcal{A}$ from q, then $\mathcal{A}^q$ accepts the suffix $w^{|u|}$ iff $\mathcal{B}_n^{(q,t)}$ accepts $w^{|u|}$. It follows that $w \in L$ iff $w \in L(\mathcal{B}_n)$. □

The complicated case is when the run of $\mathcal{B}_n$ on w does visit the $\mathcal{R}_L$-component infinitely many times. This is where the special structure of $\mathcal{P}$ guarantees that the sparse visits in the $\mathcal{A}$-component are sufficient for determining acceptance.

Lemma 25. *If the run of $\mathcal{B}_n$ on a word $w \in \Sigma^\omega$ visits the $\mathcal{R}_L$-component infinitely many times, then $w \in L$ iff $w \in L(\mathcal{B}_n)$.*

Proof. Let $\tau = s_1, s_2, s_3, \ldots$ be the run of $\mathcal{B}_n$ on w and let $\rho = q_1, q_2, q_3 \ldots$ be the run of $\mathcal{A}$ on w. We denote by $\tau[i, j]$ the infix $s_i, \ldots, s_j$ of τ. We also extend $\alpha_{\mathcal{A}}$

to (infixes of) runs by defining $\alpha_{\mathcal{A}}(\tau[i,j]) = \alpha_{\mathcal{A}}(s_i), \ldots, \alpha_{\mathcal{A}}(s_j)$. For a rank-word $u \in \Omega^*$, we say that an infix $\tau[i,j]$ is a *u-infix* if $\alpha_{\mathcal{A}}(\tau[i,j]) = u$.

If $v = \tau[i,j]$, for some $0 \leq i \leq j$, is a part of a run of $\mathcal{A}$ that consists of loops around $q_{❀}$, we define the *loop type of* v to be the word in Ω^* that describes the highest rank of each simple loop around $q_{❀}$ in v. An infix of τ whose loop type is u_i for some $0 \leq i \leq k$ is called a *u_i-loop-infix*.

By our assumption, τ contains infinitely many u_k-infixes. Indeed, by the definition of $\mathcal{P}$, otherwise τ gets trapped in the $\mathcal{A}$-component. We proceed by establishing a connection between u_i-loop-infixes of τ and the corresponding infixes of ρ, for all $0 \leq i \leq k$.

Let $i \in \{0, \ldots, k\}$, and consider a u_i-loop-infix, By the definition of u_i, such a u_i-loop-infix consists of a sequence of $M = |Q| + 1$ i-loops in τ, with loops of lower ranks between them. We can write $w = xvw'$, where $v = w[c,d]$ is the sub word that corresponds to the u_i-loop-infix. Let $u'_i = \alpha_{\mathcal{A}}(\rho[c,d])$ be the ranks of ρ in its part that corresponds to v.

By our choice of M, we can find two indices $c \leq j < l \leq d$ such that the pairs $\langle (q_j, t), q'_j \rangle$ and $\langle (q_l, t'), q'_l \rangle$ reached by (τ, ρ) in indices j and l, respectively, satisfy $q_j = q_l = q_{❀}$ and $q'_j = q'_l$. Additionally, being a part of the run on a u_i-loop-infix, the highest rank seen between q_j and q_l in τ is i. We write $v = v_1 v_2 v_3$, where $v_1 = v[1,j]$, $v_2 = v[j+1,l]$, and $v_3 = v[l+1,|v|]$. Thus, the loop type of v_2 is in $(iu_{i-1})^+ i$, with the convention $u_{-1} = \epsilon$.

Consider the runs μ and η of $\mathcal{A}^{q_j}$ and of $\mathcal{A}^{q'_j}$ on v_2^ω, respectively. These runs are loops labeled by v_2, where the highest rank in μ is i. By Lemma 23, $L(\mathcal{A}^{q_j}) = L(\mathcal{A}^{q'_j})$, so the highest rank in η must have same parity (odd or even) as i.

Thus, we showed that for every $i \in \{0, \ldots, k\}$, and for every u_i-loop-infix v of τ, there is an infix of v with loop-type in $(iu_{i-1})^+ i$, such that the infix of ρ corresponding to v has highest rank of same parity as i.

We want to show that rank k is witnessed on ρ during every u_k-infix of τ. Assume by way of contradiction that this is not the case. This means that there is some u_k-infix v' in τ such that all ranks visited in ρ along v' are at most $k-2$. Indeed, since the highest rank has to be of the same parity as k, which has the same parity as k, it cannot be $k-1$. By the same argument, within v' there is an infix v'' of u_{k-1} of the form $((k-1)(u_{k-2}))^+(k-1)$ in which the highest rank in ρ is of the same parity as $k-1$. As v'' is also an infix of v', the highest rank in ρ along v'' is at most $k-2$. Thus, the highest rank along v'' is at most $k-3$. By continuing this argument by induction down to 0, we reach a contradiction (in fact it is reached at level 1), as no rank below 0 is available.

We conclude that the run ρ witnesses a rank k in any u_k-infix of τ. Since τ contains infinitely many u_k-infixes, then ρ contains infinitely many ranks k, and, depending on the parity of k, either both ρ and τ are rejecting or both are accepting.

This conclude the proof that $w \in L$ iff $w \in L(\mathcal{B}_n)$. $\square$

We proceed to show that the sensing cost of the sequence of DPAs $\mathcal{B}_n$ indeed converges to that of $\mathcal{R}_L$.

Lemma 26. $\lim_{n\to\infty} scost(\mathcal{B}_n) = scost(\mathcal{R}_L)$.

Proof. Since $\mathcal{A}$ is strongly connected, $q_{❀}$ is reachable from every state in $\mathcal{A}$. Also, since $q_{❀}$ is a $[0,k]$-flower, we can construct a sequence of loops around $q_{❀}$ whose ranks correspond to the word u_k. Thus, t_{acc} is reachable from every state in the $\mathcal{A}$-component. This implies that $\mathcal{B}_n$ is strongly connected, and therefore, a run of $\mathcal{B}_n$ is expected to traverse both components infinitely often, making the $\mathcal{R}_L$-component more dominant as n grows, implying that $\lim_{n\to\infty} scost(\mathcal{B}_n) = scost(\mathcal{R}_L)$.

We now turn to formalize this intuition, by carefully analyzing $\mathcal{B}_n$'s Markov chain. Consider the Markov chain that corresponds to $\mathcal{B}_n$, and let T_n be its transition matrix. For a vector $v = (v_1, \ldots, v_m)$, let $\|v\| = \sum_{i=1}^{m} v_i$. The sensing cost of $\mathcal{B}_n$ is computed using the limiting distribution π_n of $\mathcal{B}_n$. Since $\mathcal{B}_n$ is strongly connected, it has a unique stationary distribution. Thus π_n is obtained as a solution of the equation $\pi_n T_n = \pi_n$, subject to the constraint $\|\pi_n\| = 1$. We denote by $x_n = (x_{n,1}, \ldots, x_{n,d})$ the sub-vector of π_n that corresponds to the $\mathcal{A}$-component, and denote by $y_{n,i}$ the sub-vector that corresponds to the i-th $\mathcal{R}_L$-component. For every $1 \le i < n$, it is easy to see that $\|y_{n,i}\| = \|y_{n,i+1}\|$. Indeed, all the transitions from the i-th copy of $\mathcal{R}_L$ are to the $(i+1)$-th copy. Thus, $\|y_{n,i}\|$ is independent of i. Let $a_n = \|y_{n,1}\| \ge 0$ and $b_n = \|x_n\| \ge 0$. Observe that for every n, we have that $na_n + b_n = 1$, so in particular, $\lim_{n\to\infty} a_n = 0$.

Let $\epsilon > 0$. By the definition of $\mathcal{P}$, we always enter the first $\mathcal{R}_L$-component in the state $[q_{❀}]$ of $\mathcal{R}_L$ — the state corresponding to $L(\mathcal{A}^{q_{❀}})$. Let τ_0 be the distribution over the states of $\mathcal{R}_L$ in which $[q_{❀}]$ is assigned probability 1 and the other states of $\mathcal{R}_L$ are assigned 0, and let $\theta = (\theta_1, \ldots, \theta_l)$ be the unique stationary distribution of $\mathcal{R}_L$. Let R be the matrix associated with the Markov chain of $\mathcal{R}_L$, and let $\tau_i = \tau_0 R^i$ for every $i \ge 1$. By [9], there exists n_0 such that for every index $i \ge n_0$ and $1 \le j \le l$, we have that $|\tau_{i,j} - \theta_j| \le \epsilon$. Note that for all n and i, it holds that $y_{n,i} = \tau_i$.

Let $\{q_1, \ldots, q_d\}$ be the states in the $\mathcal{A}$-component. Since $\mathcal{P}$ is strongly connected, then for every $1 \le i, j \le d$ there is a path from q_i to q_j with at most $d-1$ transitions. Since there are at most $|\Sigma|$ edges leaving each state, the probability of taking each edge along such a path is at least $\mu = \frac{1}{|\Sigma|}$. Therefore, the probability of reaching q_j from q_i is at least μ^{d-1}. Consider the maximal entry in x_n (w.l.o.g $x_{n,1}$). It holds that $x_{n,1} \ge \frac{\|x_n\|}{d} = \frac{b_n}{d}$. Therefore, for all $1 \le j \le d$, we have $x_{n,j} \ge \mu^{d-1} x_{n,1} \ge \frac{\mu^{d-1}}{d} b_n$.

Recall that t_{acc} is reachable from all the states in the $\mathcal{A}$-component. Therefore, there is at least one transition from some state q_j of the $\mathcal{A}$-component to the first $\mathcal{R}_L$-component. This means that $a_n \ge \mu \cdot x_{n,j} \ge \frac{\mu^d}{d} b_n$, implying that $b_n \le \mu^{-d} \cdot d \cdot a_n$, which tends to 0 when n tends to ∞.

We now consider the cost of $\mathcal{B}_n$, for $n \ge n_0$. Clearly, the maximal cost of a state is $|P|$. Let c_j be the cost of the state indexed j in $\mathcal{R}_L$, and let $\tau_i = (\tau_{i,1}, \ldots, \tau_{i,l})$.

Then,

$$\begin{aligned} scost(\mathcal{B}_n) &\leq b_n|P| + n_0 a_n|P| + a_n \sum_{i=n_0}^{n}\sum_{j=1}^{d} \tau_{i,j}c_j \\ &\leq b_n|P| + n_0 a_n|P| + a_n \sum_{i=n_0}^{n}\sum_{j=1}^{d} (\theta_j+\epsilon)c_j. \end{aligned}$$

Therefore, when $n \to \infty$, as $a_n \to 0$ and $b_n \to 0$, we get $scost(\mathcal{B}_n) \leq (n - n_0)a_n \sum_{j=1}^{d} \theta_j c_j + O(\epsilon) + o(1)$. But we know $na_n + b_n = 1$, and $b_n \to 0$, so $na_n \to 1$, and therefore $(n - n_0)a_n \to 1$. We get $scost(\mathcal{B}_n) \leq scost(\mathcal{R}_L) + O(\epsilon) + o(1)$. Furthermore, by Lemmas 24 and 25, for all n we have $L(\mathcal{B}_n) = L(\mathcal{A})$, thus $scost(\mathcal{R}_L) \leq scost(\mathcal{B}_n)$.

Since the above holds for all $\epsilon > 0$, we conclude that $\lim_{n\to\infty} scost(\mathcal{B}_n) = scost(\mathcal{R}_L)$. □

Lemmas 24, and 25 put together ensure that for languages accepted by strongly connected DPAs, we have that $L(\mathcal{B}_n) = L$, so with Lemma 26, we get $scost(L) = scost(\mathcal{R}_L)$.

It is left to remove the assumption about L being recognizable by strongly connected DPAs.

Assume then that $\mathcal{A}$ is not a strongly connected DPA, and let $C_1, \ldots, C_l$ be its ergodic SCCs. For every $1 \leq i \leq l$ and $q \in C_i$, let L_i^q be the language recognized by C_i, with q as an initial state. Note that the residual automata R_i^q of languages L_i^q only differ in their initial states. We refer to R_i as the *common automaton* where no initial state is defined. For every $1 \leq i \leq l$ and $q \in C_i$, we have $scost(L_i^q) = scost(R_i)$.

We can now apply the construction above on R_i, which works simultaneously for all initial states. This yields automata $(\mathcal{B}_{1,n}, \ldots, \mathcal{B}_{l,n})_{n\geq 1}$ with no initial states specified, such that for every $1 \leq i \leq l$:

(1) $\lim_{n\to\infty} scost(B_{i,n}) = scost(R_i)$
(2) For all $q \in C_i$ and $n \geq 1$, there is a state q_n in $B_{i,n}$ such that $B_{i,n}$ with q_n as initial state recognizes exactly L_i^q.

Let $\mathcal{A}_n$ be the DPA obtained from $\mathcal{A}$ by replacing each ergodic SCC C_i by $\mathcal{B}_{i,n}$, with the entry points to $\mathcal{B}_{i,n}$ being chosen to preserve the correct residual language. Formally, a transition (p, a, q) of $\mathcal{A}$ is replaced by (p, a, q_n) in $\mathcal{A}_n$, where q_n is as defined in (2) above.

This construction ensures that for all $n \geq 1$, we have $L(\mathcal{A}_n) = L(\mathcal{A})$. Indeed, if the run on a word enters a component C_i in $\mathcal{A}$, the corresponding run in $\mathcal{A}_n$ enters a component $\mathcal{B}_{i,n}$ in a state q_n that matches the correct residual language. Then, the correctness of the construction in the strongly-connected case guarantees that the word is accepted if and only if it is in L.

It remains to show that $\lim_{n\to\infty} scost(\mathcal{A}_n) \to scost(\mathcal{R}_L)$, from which will follow that $scost(L) = scost(\mathcal{R}_L)$.

Let ρ (resp. ρ_n) be the SCC-reachability distribution of $\mathcal{A}$ (resp. $\mathcal{A}_n$). Recall that the ergodic components $(C_i)_{1\le i\le l}$ in $\mathcal{A}$ are replaced by $(\mathcal{B}_{i,n})_{1\le i\le l}$ in $\mathcal{A}_n$, and the transient component are left unchanged. Thus, for every $1 \le i \le l$ and $n \ge 1$, we have that $\rho(C_i) = \rho_n(\mathcal{B}_{i,n})$. By Theorem 5, we obtain $scost(\mathcal{A}_n) = \sum_{i=1}^{l} \rho(C_i) scost(\mathcal{B}_{i,n})$. When n tends to ∞, we get $\sum_{i=1}^{l} \rho(C_i) scost(R_i)$ (by (1) above).

Finally, let $\mathcal{A}_R$ be the DPA obtained from $\mathcal{A}$ by replacing each SCC C_i by its residual automaton R_i, again keeping the entry points to R_i consistent with residuals (here there is no choice: the states of R_i are exactly the possible residuals).

Since the SCC-reachability distribution in $\mathcal{A}$ and $\mathcal{A}_R$ coincide, it follows that $scost(\mathcal{A}_R) = \sum_{i=1}^{l} \rho(C_i) scost(R_i) = \lim_{n\to\infty} scost(\mathcal{A}_n)$. It remains to show that $\mathcal{A}_R$ has same cost as the residual automaton $\mathcal{R}_L$ of L, and we can conclude that $\lim_{n\to\infty} scost(\mathcal{A}_n) = scost(\mathcal{R}_L)$, and finally $scost(L) \le scost(\mathcal{R}_L)$. Since the opposite inequality is always true by Lemma 19, we get $scost(L) = scost(\mathcal{R}_L)$.

Lemma 27. *$scost(\mathcal{A}_R) = scost(\mathcal{R}_L)$.*

Proof. Let $D_1, \ldots, D_k$ be the ergodic SCCs of $\mathcal{R}_L$. For every $1 \le i \le l$ and $q \in C_I$, there exists j_i^q such that L_i^q is a state of $D_{j_i^q}$. Moreover, j_i^q does not depend on q, since both C_i and D_j are strongly connected. Thus, every C_i can be mapped to some D_{j_i} such that the states of D_{j_i} are exactly L_i^q for $q \in C_i$. In fact, for each i, the automata R_i and D_{j_i} are exactly the same, except for their initial states.

Let ρ be the SCC-reachability distribution of $\mathcal{A}$ (or equivalently $\mathcal{A}_R$) and let σ the SCC-reachability distribution of $\mathcal{R}_L$. Since the residual languages must match in $\mathcal{A}$ and $\mathcal{R}_L$, then for every $1 \le j \le k$, we have $\sigma(D_j) = \sum_{j_i=j} \rho(C_i)$. Therefore, $scost(\mathcal{R}_L) = \sum_{j=1}^{k} \sigma(D_j) scost(D_j) = \sum_{j=1}^{k} \sum_{j_i=j} \rho(C_i) scost(D_j) = \sum_{i=1}^{l} \rho(C_i) scost(R_i) = scost(\mathcal{A}_R)$. □

Remark 28. All our results can be easily extended to a setting with a non-uniform distribution on the letters given by any Markov chain, or with a different cost for each input in each state. We can also use a decision tree to read the inputs instead of reading them simultaneously, defining for instance a cost of 1.5 if the state starts by reading a, then if a is true it also reads b.

4.2. *Attainability of the minimal sensing cost*

As we have shown in Example 17, the minimal sensing is sometimes attained only as a limit of an infinite sequence of automata. It is thus of interest to determine when the minimal sensing cost is actually attained by a concrete automaton. It is tempting to think that when the minimal sensing cost is attained, then it is attained

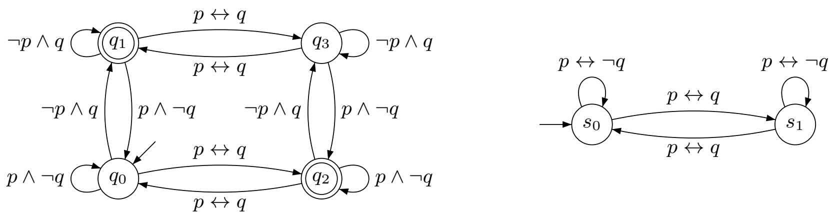

Fig. 4. An NBA with attainable minimal sensing cost and its residual automaton.

by adding an acceptance condition on top of $\mathcal{R}_L$. As we now show, this is not the case.

Example 29. Consider the DBA $\mathcal{A}$ in the left of Fig. 4. Recall that we denote sets of letters by a propositional formula that characterize them. Thus, $p \leftrightarrow q$ stands for the letters $\emptyset$ or $\{p, q\}$, $\neg p \wedge q$ for $\{q\}$, and $p \wedge \neg q$ for $\{p\}$.The DBA $\mathcal{A}$ accepts a word $w \in (2^{\{p,q\}})^\omega$ iff one of the following holds:

(1) w has infinitely many occurrences of $p \leftrightarrow q$,
(2) w has an even number of occurrences of $p \leftrightarrow q$ and infinitely many occurrences of $\neg p \wedge q$,
(3) w has an odd number of occurrences of $p \leftrightarrow q$ and infinitely many occurrences of $p \wedge \neg q$.

By Theorems 18 and 7, it can be shown that $scost(L(\mathcal{A})) = scost(\mathcal{A}) = 2$. Thus, the minimal sensing cost for $L(\mathcal{A})$ is attained by $\mathcal{A}$. The residual automaton $\mathcal{R}_L$ is depicted in the right, and it is easy to verify that no parity condition can be put on top of $\mathcal{R}_L$ such that its language becomes $L(\mathcal{A})$.

Example 29 suggests that deciding whether the minimal sensing cost is attained is not straightforward. Observe, however, that even asking whether the minimal sensing is attained by an accepting condition on top of $\mathcal{R}_L$ is not trivially tractable. Indeed, it is easy to show that this can be solved in NP: simply guess an acceptance condition, and check the equivalence of the resulting automata. However, we now show that this can in fact be decided in polynomial time.

Theorem 30. *Given a DPA $\mathcal{A}$, the problem of deciding whether there exists a parity acceptance condition on top of $\mathcal{R}_L$ such that $L(\mathcal{R}_L) = L(\mathcal{A})$ can be solved in polynomial time.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, \Delta, q_0, \alpha \rangle$ and let $L = L(\mathcal{A})$. Recall that each state of $\mathcal{R}_L$ corresponds to a residual language $u^{-1}L$ for some $u \in \Sigma^*$. The states of $\mathcal{R}_L$ thus induce a partition of Q by associating with every state $q \in Q$ the residual language $\{w \in \Sigma^\omega : w \in L(\mathcal{A}^q)\}$. We thus refer to the states of $\mathcal{R}_L$ as subsets of Q.

We describe an algorithm that assigns parity ranks to the states of $\mathcal{R}_L$ to obtain a DPA $\mathcal{B}$ such that $L(\mathcal{B}) = L$, or outputs that no such parity ranking exists. The algorithm proceeds as follows.

(1) Compute $\mathcal{R}_L$, and the mapping from states of $\mathcal{A}$ to the states of $\mathcal{R}_L$.
(2) Find a state $S \subseteq Q$ of $\mathcal{R}_L$ such that either for every $q \in S$, all cycles through q in $\mathcal{A}$ are accepting, or all are rejecting. If no such state exists, return that no parity ranking exists.
(3) Assign S the maximal available parity rank p that is either odd or even, depending on the behavior of the cycles. Note that the initial maximal parity rank is the number of states of $\mathcal{R}_L$.
(4) Decrease the maximal available rank to $p - 1$. Remove S from $\mathcal{R}_L$ and from $\mathcal{A}$, and go back to Step (2) with the resulting automata.
(5) When all states of $\mathcal{R}_L$ have been assigned a parity rank, return the resulting automaton.

We now prove that the algorithm can be implemented in polynomial time, and that it is correct. We start with the complexity analysis. Steps (1), (3), and (4) can clearly be computed in polynomial time. Step (2) can be computed in polynomial time by noticing that the complement problem, namely determining whether a given state of $\mathcal{A}$ has both an accepting and a rejecting cycle can be done in NL, by guessing two cycles and verifying the property. Since NL$\subseteq$ P, we can also compute Step (2) in polynomial time.

We proceed to show the correctness of the algorithm. For the first direction, assume the algorithm computes a parity ranking on $\mathcal{R}_L$, thus returning a DPA $\mathcal{B}$. We claim that $L(\mathcal{B}) = L(\mathcal{A})$. Indeed, consider a word $w \in \Sigma^\omega$, and consider the runs of $\mathcal{B}$ and $\mathcal{A}$ on w. Let $\rho : Q \to 2^Q$ be the mapping of the states of $\mathcal{A}$ to their corresponding states in $\mathcal{B}$. Consider the maximal priority p that occurs infinitely often during the run of $\mathcal{B}$ on w, and let $S \subseteq Q$ be the state associated with this priority (note that the algorithm outputs a unique rank to each state). Assume S received its rank p during iteration i of the algorithm. Since S has the maximal rank visited infinitely often along the run of $\mathcal{B}$ on w, it follows that for every state q visited infinitely often during the run of $\mathcal{A}$ on w it holds that $\rho(q)$ was not assigned a parity rank by the algorithm before iteration i. Thus, the cycles through the states of S that occur infinitely often in the run of $\mathcal{A}$ on w are either all accepting or all rejecting, according to which the rank p is determined. We conclude that the run of $\mathcal{A}$ on w is accepting/rejecting similarly to the run of $\mathcal{B}$. We conclude that $L(\mathcal{B}) = L(\mathcal{A})$.

Conversely, suppose the algorithm returns that no parity ranking exists, in Step (2) in iteration i. This means that for every state $S \subseteq Q$ of $\mathcal{R}_L$ that was not yet assigned with a parity ranking, there exist states $q, r \in S$ (possibly with $q = r$) and cycles c_q, c_r through q and r respectively, such that c_q is accepting and c_r is rejecting. Moreover, c_r and c_q do not go through states that were previously given a parity rank by the algorithm.

Consider the states $\{S_1, \ldots, S_k\} \subseteq 2^Q$ of $\mathcal{R}_L$ that were not assigned a parity rank by iteration i. Since c_r and c_q induce cycles in those states, it follows that there exists a cycle in $\mathcal{R}_L$ composed of those states. Assume by way of contradiction that there exists a parity ranking function α on $\mathcal{R}_L$ such that the resulting DPA $\mathcal{B}$ satisfies $L(\mathcal{B}) = L(\mathcal{A})$. Let $S_k = \arg\max\{\alpha(S_j) : 1 \leq j \leq k\}$ and assume w.l.og. that $\alpha(S_k)$ is even. Then, every cycle through S_k that uses only the states in $\{S_1, \ldots, S_k\}$ is accepting. This implies that in particular, every cycle through every state $q \in S_k$ in $\mathcal{A}$ is accepting, in contradiction to the observation above. We conclude that there does not exist any parity ranking on $\mathcal{R}_L$, so the algorithm is correct. $\square$

5. Monitoring

As described in Sec. 2, the definition of sensing takes into an account all words in $(2^P)^\omega$, regardless their membership in the language. In monitoring, we restrict attention to words in the language, as once a violation is detected, no further sensing is required. In particular, in safety languages, violation amounts to a detection of a bad prefix, and indeed safety languages are the prominent class of languages for which monitoring is used [11].

As it turns out, however, there are many approaches to define the corresponding probability space. We suggest here two. Let $\mathcal{A}$ be a DLA and let $L = L(\mathcal{A})$.

(1) **[Letter-based]** At each step, we uniformly draw a "safe" letter — one with which we are still generating a word in *pref*(L), thereby iteratively generating a random word in L.
(2) **[Word-based]** At the beginning, we uniformly draw a word in L.

We denote the sensing cost of $\mathcal{A}$ in the letter- and word-based approaches *lcost*$(\mathcal{A})$ and *wcost*$(\mathcal{A})$, respectively. The two definitions yield two different probability measures on L, as demonstrated in Example 31 below.

Example 31. Let $P = \{a\}$ and consider the safety language $L = a^\omega + (\neg a) \cdot (\mathit{True})^\omega$. That is, if the first letter is $\{a\}$, then the suffix should be $\{a\}^\omega$, and if the first letter is $\emptyset$, then all suffixes result in a word in L. Consider the DLA $\mathcal{A}$ for L in Fig. 5.

In the letter-based definition, we initially draw a letter from $2^{\{a\}}$ uniformly, i.e., either a or $\neg a$ with probability $\frac{1}{2}$. If we draw $\neg a$, then we move to q_1 and stay there forever. If we draw a, then we move to q_2 and stay there forever. Since $\mathit{scost}(q_1) = 0$ and $\mathit{scost}(q_2) = 1$, and we reach q_1 and q_2 w.p $\frac{1}{2}$, we get $\mathit{lcost}(\mathcal{A}) = \frac{1}{2}$.

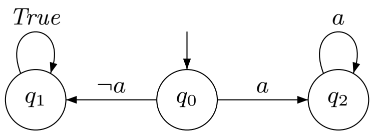

Fig. 5. A DLA for $a^\omega + (\neg a) \cdot (\mathit{True})^\omega$.

In the word-based definition, we assign a uniform probability to the words in L. In this case, almost all words are not a^ω, and thus the probability of a^ω is 0. This means that we will get to q_1 with probability 1, and thus $wcost(\mathcal{A}) = 0$.

As a more realistic example, recall our traffic monitor in Sec. 1. There, the behavior of the cars is the random input, and the two approaches can be understood as follows. In the letter-based approach, we assume that the drivers do their best to avoid accidents regardless of the history of the traffic and the traffic lights so far. Thus, after every safe prefix, we assume that the next input is also safe. In the word-based approach, we assume that the city is planned well enough to avoid accidents. Thus, we a-priori set the distribution to safe traffic behaviors according to their likelihood.

We now define the two approaches formally.

The Letter-Based Approach

Consider a DLA $\mathcal{A} = \langle \Sigma, Q, \delta, q_0 \rangle$. For a state $q \in Q$, let $avail(q)$ be the set of letters available in q, namely letters that do not cause $\mathcal{A}$ to get stuck. Formally, $avail(q) = \{\sigma \in \Sigma : \delta(q, \sigma) \text{ is defined}\}$. We model the drawing of available letters by the Markov chain $\mathcal{M}_\mathcal{A} = \langle Q, P \rangle$, where the probability of a transition from state q to state q' in $\mathcal{M}_\mathcal{A}$ is $P(q, q') = \frac{|\{\sigma \in \Sigma : \delta(q,\sigma) = q'\}|}{|avail(q)|}$. Let π be the limiting distribution of $\mathcal{M}_\mathcal{A}$. We define $lcost(\mathcal{A}) = \sum_{q \in Q} \pi(q) \cdot scost(q)$.

Since computing the limiting distribution can be done in polynomial time, we have the following.

Theorem 32. *Given a DLA $\mathcal{A}$, the sensing cost $lcost(\mathcal{A})$ can be calculated in polynomial time.*

The Word-Based Approach

Consider a DLA $\mathcal{A} = \langle 2^P, Q, q_0, \delta \rangle$ recognizing a non-empty safety language L. Recall that $scost(\mathcal{A}) = \lim_{n\to\infty} \frac{1}{|\Sigma|^n} \sum_{u \in \Sigma^n} scost_\mathcal{A}(u)$, which coincides with $\mathbb{E}[scost_\mathcal{A}(u)]$ where $\mathbb{E}$ is the expectation with respect to the standard measure on Σ^ω. Our goal here is to replace this standard measure with one that restricts attention to words in L. Thus, we define $wcost(\mathcal{A}) = \mathbb{E}[scost(u) \mid u \in L]$. For $n \geq 0$, let $pref(L, n)$ be the set of prefixes of L of length n. Formally, $pref(L, n) = pref(L) \cap \Sigma^n$. As in the case of the standard measure, the expectation-based definition coincides with one that that is based on a limiting process: $wcost(\mathcal{A}) = \lim_{n\to\infty} \frac{1}{|pref(L,n)|} \sum_{u \in pref(L,n)} scost_\mathcal{A}(u)$. Thus, the expressions for $scost$ and $wcost$ are similar, except that in the expectation-based definition we add conditional probability, restricting attention to words in L, and in the limiting process we replace Σ^n by $pref(L, n)$.

Note that the term $\frac{1}{|pref(L,n)|}$ is always defined, as L is a non-empty safety language. In particular, the expectation is well defined even if L has measure 0 in Σ^ω.

Theorem 33. *Given a DLA $\mathcal{A}$, we can compute $wcost(\mathcal{A})$ in polynomial time.*

Proof. We will use here formal power series on one variable z, a classical tool for graph and automata combinatorics. They can be thought of as polynomials of infinite degree.

For states $p, q \in Q$ and for $n \in \mathbb{N}$, let $\#paths(p, q, n)$ denote the number of paths (each one labeled by a distinct word) of length n from p to q in $\mathcal{A}$. We define the generating functions: $C_{p,q}(z) = \sum_{n\in\mathbb{N}} \#paths(p, q, n)z^n$ and $F_q(z) = C_{q_0,q}(z) \sum_{p\in Q} C_{q,p}(z)$. Let $[z^n]F_q(z)$ be the coefficient of z^n in $F_q(z)$. By the definition of $C_{q_0,q}$, we get

$$[z^n]F_q(z) = \sum_{k=0}^{n} \#paths(q_0, q, k) \sum_{p\in Q} \#paths(q, p, n-k).$$

Therefore, $[z^n]F_q(z)$ is the total number of times the state q is used when listing all paths of length n from q_0.

Thus, we have

$$\sum_{u\in pref(L,n)} scost(u) = \frac{1}{n} \sum_{q\in Q} scost(q)[z^n]F_q(z).$$

Finally, let $S(z) = \sum_{p\in p} C_{q_0,p}(z)$. Then,

$$wcost(\mathcal{A}) = \lim_{n\to\infty} \frac{1}{n \cdot [z^n]S(z)} \sum_{q\in Q} scost(q)[z^n]F_q(z).$$

By [8], for every $p, q \in Q$, we can compute in polynomial time (using standard algorithms on matrices) rational expressions for $C_{p,q}(z)$. The base case is for computing coefficients in the same irreducible aperiodic SCC represented by a matrix M: it suffices to compute the matrix $R(z) = (Id - zM)^{-1}$, its coefficient (p, q) is $C_{p,q}(z)$. For instance if $\{p\}$ is a SCC in $\mathcal{A}$ with a self-loop labeled by k letters, then $C_{p,p}(z) = \frac{1}{1-kz}$. Other $C_{p,q}$ are then computed from this base case via standard operations on rational functions. In particular, from [8] there is a period $d \leq |Q|$ such that for every $i \in \{0, \dots, d-1\}$ and for all rational functions $Q(z) \in \{S(z)\} \cup \bigcup_{q\in Q} \{F_q(z)\}$ considered here, we can compute in polynomial time γ, k, and λ such that $[z^{nd+i}]Q(z) \sim \gamma(nd+i)^k \lambda^{nd+i}$ (where for functions $f, g : \mathbb{N} \to \mathbb{R}$ we have $f(n) \sim g(n)$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$). We remind the formula that allows us to do so. Let $Q(z) = \frac{A(z)+B(z)\left(1-\frac{z}{r}\right)^{-j}}{z^i}$ be a rational function of convergence radius r, where $i, j \in \mathbb{N}$, $A(z)$ and $B(z)$ have convergence radius strictly greater than r, and $B(r) \neq 0$. Then we have

$$[z^n]Q(z) \sim \frac{B(r)}{(j-1)! \cdot r^i} n^{j-1} (1/r)^n.$$

Notice that r will in general be a real algebraic number, that will be represented in our algorithm as a root of a polynomial with integer coefficients. Standard operations as sum, product, and comparisons on algebraic numbers (represented by polynomials) can be done in polynomial time, using techniques as described in [20].

Therefore, we can compute asymptotic equivalents of the form $\alpha \cdot n^k \cdot \lambda^n$ with α, λ real algebraic and $k \in \mathbb{N}$ [8], for both $[z^n]S(z)$ and $\frac{1}{n}\sum_{q \in Q} \mathit{scost}(q)[z^n]F_q(z)$, performing an averaging operation if $d > 1$. Finally, we can compute the wanted limit thanks to these asymptotic equivalents, thereby achieving the polynomial-time computation of $\mathit{wcost}(\mathcal{A})$. □

Sensing Cost of Languages

For a safety language L, we define $\mathit{lcost}(L) = \inf\{\mathit{lcost}(\mathcal{A}) : \mathcal{A}$ is a DLA for $L\}$, and similarly for $\mathit{wcost}(L)$. Different DLAs for a language L may have different sensing costs. We show that the minimal sensing cost in both approaches is attained at the minimal-size DLA. We first need some definitions and notations.

Consider a safety language $L \subseteq \Sigma^\omega$. Note that for safety languages, there is at most one Myhill-Nerode class $[u]$, namely the class of bad prefixes, such that $u^{-1}L = \emptyset$. We denote this class $[\perp]$. The automaton $\mathcal{R}_L = \langle \Sigma, \langle L\rangle \backslash \{[\perp]\}, \delta_L, [\epsilon]\rangle$ is then the unique minimal-size DLA for L.

Consider a DLA $\mathcal{A} = \langle \Sigma, Q, q_0, \delta\rangle$ such that $L(\mathcal{A}) = L$. For a state $s = [u] \in \langle L\rangle \backslash \{[\perp]\}$ of $\mathcal{R}_L$, we associate with s a set $\mathit{states}(\mathcal{A}, s) = \{q \in Q : L(\mathcal{A}^q) = u^{-1}L\}$. That is, $\mathit{states}(\mathcal{A}, s) \subseteq Q$ contains exactly all states that $\mathcal{A}$ can be in after reading a word that leads $\mathcal{R}_L$ to $[u]$.

The following claims are simple exercises.

Proposition 34. *Consider a safety language L and a DLA $\mathcal{A}$ for it.*

(1) *The set $\{\mathit{states}(\mathcal{A}, s) : s \in \langle L\rangle \backslash \{[\perp]\}\}$ forms a partition of the states of $\mathcal{A}$.*
(2) *For every state $s \in \langle L\rangle \backslash \{[\perp]\}$ of $\mathcal{R}_L$, letter $\sigma \in \Sigma$, and state $q \in \mathit{states}(\mathcal{A}, s)$, we have $\delta(q, \sigma) \in \mathit{states}(\mathcal{A}, \delta_L(s, \sigma))$.*

Lemma 35. *Consider a safety language $L \subseteq \Sigma^\omega$. For every DLA $\mathcal{A}$ with $L(\mathcal{A}) = L$, we have that $\mathit{lcost}(\mathcal{A}) \geq \mathit{lcost}(\mathcal{R}_L)$ and $\mathit{wcost}(\mathcal{A}) \geq \mathit{wcost}(\mathcal{R}_L)$*

Proof. We start with *lcost*. Consider a finite word $u \in \Sigma^*$ that is not a bad prefix for L. After reading u, the DLA $\mathcal{R}_L$ reaches the state $[u]$ and the DLA $\mathcal{A}$ reaches a state q with $L(\mathcal{A}^q) = u^{-1}L$. Indeed, otherwise we can point to a word with prefix u that is accepted only in one of the DLAs. We claim that for every state $q \in Q$ such that $L(\mathcal{A}^q) = u^{-1}L$, it holds that $\mathit{sensed}([u]) \subseteq \mathit{sensed}(q)$. To see this, consider a signal $p \in \mathit{sensed}([u])$. By definition, there exists a set $S \subseteq P$ and words u_1 and u_2 such that $([u], S\backslash\{p\}, [u_1]) \in \Delta_L$, $([u], S \cup \{p\}, [u_2]) \in \Delta_L$, yet $[u_1] \neq [u_2]$. By the definition of $\mathcal{R}_L$, there exists $z \in (2^P)^*$ such that, w.l.o.g, $z \in u_1^{-1}L \backslash u_2^{-1}L$. Hence, as $L(\mathcal{A}^q) = u^{-1}L$, we have that $\mathcal{A}^q$ accepts $(S\backslash\{p\}) \cdot z$ and rejects $(S \cup \{p\}) \cdot z$. Let $\delta_\mathcal{A}$ be the transition function of $\mathcal{A}$. By the above, $\delta_\mathcal{A}(q, S\backslash\{p\}) \neq \delta_\mathcal{A}(q, S \cup \{p\})$. Therefore, $p \in \mathit{sensed}(q)$, and we are done. Now, $\mathit{sensed}([u]) \subseteq \mathit{sensed}(q)$ implies that $\mathit{scost}(q) \geq \mathit{scost}([u])$. Since our assumption on q is only that $L(\mathcal{A}^q) = u^{-1}L$, we get that $q \in \mathit{states}(\mathcal{A}, [u])$. Thus, we conclude that for every $s \in \langle L\rangle \backslash \{[\perp]\}$ and for every $q \in \mathit{states}(\mathcal{A}, s)$ we have that $\mathit{scost}(q) \geq \mathit{scost}(s)$.

Next, consider the Markov chains $\mathcal{M}_{\mathcal{A}}$ and $\mathcal{M}_{\mathcal{R}_L}$, and let P and R be their respective transition matrices. We index the rows and columns of P (resp. R) by Q (resp. $\langle L\rangle\setminus\{[\bot]\}$). Let $v^0 \in [0,1]^Q$ be the initial vector for $\mathcal{A}$, thus $v^0(q_0) = 1$ and $v^0(q) = 0$ for $q \neq q_0$. Similarly, let $u^0([\epsilon]) = 1$ and $u^0([w]) = 0$ for all $[w] \neq [\epsilon]$. For $m \in \mathbb{N}$, let $v^m = v^0 P^m$ and $u^m = u^0 R^m$.

We claim that for every $s \in \langle L\rangle\setminus\{[\bot]\}$ it holds that $u^m(s) = \sum_{q\in states(\mathcal{A},s)} v^m(q)$.

The proof of the claim proceeds by an induction on m. For $m = 0$, we have $q_0 \in [\epsilon]$, and the claim follows trivially. Assume correctness for m, we prove the claim for $m+1$.

Consider states $s, s' \in \langle L\rangle\setminus\{[\bot]\}$. For every state $q \in states(\mathcal{A}, s)$ it holds that

$$\begin{aligned} R_{s,s'} &= \frac{|\{\sigma : \delta_L(s,\sigma) = s'\}|}{dom(s)} \\ &= \frac{|\{\sigma : \delta(q,\sigma) = q' \in states(\mathcal{A},s')\}|}{dom(q)} = \sum_{q'\in states(\mathcal{A},s')} P_{q,q'} \qquad (1) \end{aligned}$$

where the second equality follows from Observation 34 and by observing that $dom(q) = dom(s)$. The latter holds since if $\sigma \in dom(q)$, then there exists $q' \in Q$ such that $q' = \delta(q,\sigma)$, so $q' \in states(\delta_L(s,\sigma))$ and $\sigma \in dom(s)$, so $dom(q) \subseteq dom(s)$, and conversely - if $\sigma \in dom(s)$ then by Proposition 34 we have that $\sigma \in dom(q)$, so $dom(s) \subseteq dom(q)$.

Now, for every state $s' \in \langle L\rangle\setminus\{[\bot]\}$, we have that

$$\begin{aligned} u^{m+1}(s') &= \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} R_{s,s'} u^m(s) \\ &= \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} R_{s,s'} \sum_{q\in states(\mathcal{A},s)} v^m(q) && \text{(Induction Hypothesis)} \\ &= \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} \sum_{q\in states(\mathcal{A},s)} R_{s,s'} v^m(q) \\ &= \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} \sum_{q\in states(\mathcal{A},s)} \sum_{q'\in states(\mathcal{A},s')} P_{q,q'} v^m(q) && \text{(Equation (1))} \\ &= \sum_{q'\in states(\mathcal{A},s')} \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} \sum_{q\in states(\mathcal{A},s)} P_{q,q'} v^m(q) \\ &= \sum_{q'\in states(\mathcal{A},s')} \sum_{q\in Q} P_{q,q'} v^m(q) && \text{(Proposition 34)} \\ &= \sum_{q'\in states(\mathcal{A},s')} v^{m+1}(q') \end{aligned}$$

and the induction is complete.

Now, let π and τ be the limiting distributions of $\mathcal{M}_{\mathcal{A}}$ and $\mathcal{M}_{\mathcal{R}_L}$ respectively, then $\pi = \lim_{n\to\infty} \frac{1}{n}\sum_{m=1}^{n} v^0 P^n$ and $\tau = \lim_{n\to\infty} \frac{1}{n}\sum_{m=1}^{n} u^0 R^n$, and by the above we have that $\tau(s) = \sum_{q\in states(\mathcal{A},s)} \pi(q)$.

Since $scost(q) \geq scost(s)$ for every $q \in states(\mathcal{A}, s)$, we conclude that

$$\begin{aligned} lcost(\mathcal{A}) &= \sum_{q\in Q} \pi(q)scost(q) = \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} \sum_{q\in states(\mathcal{A},s)} \pi(q)scost(q) \\ &\geq \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} \sum_{q\in states(\mathcal{A},s)} \pi(q)scost(s) \\ &= \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} scost(s) \sum_{q\in states(\mathcal{A},s)} \pi(q) \\ &= \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} scost(s)\tau(s) = lcost(\mathcal{R}_L). \end{aligned}$$

We proceed to *wcost*. Following similar arguments as above, we see that for every finite word w, we have $scost_{\mathcal{A}}(w) \geq scost_{\mathcal{R}_L}(w)$. Therefore, for any $n \geq 0$ we have $\sum_{w\in pref(L)\cap\Sigma^n} scost_{\mathcal{A}}(w) \geq \sum_{w\in pref(L)\cap\Sigma^n} scost_{\mathcal{R}_L}(w)$, and finally $wcost(\mathcal{A}) \geq wcost(\mathcal{R}_L)$. This shows that $wcost(\mathcal{R}_L) = wcost(L)$. $\square$

Lemma 35 and Theorems 32 and 33 allow us to conclude with the following.

Theorem 36. *Given a DLA $\mathcal{A}$, we can compute $lcost(L(\mathcal{A}))$ and $wcost(L(\mathcal{A}))$ in polynomial time.*

Example 37. Consider the DLA $\mathcal{A}$ over the alphabet $2^{\{a,b\}}$ appearing in Fig. 6.

Clearly, $\mathcal{A}$ is a minimal automaton for $L = (\neg a \vee \neg b)^\omega + (\neg a \vee \neg b)^* \cdot (a \wedge b) \cdot a^\omega$. By Lemma 9, we can calculate the sensing cost of $\mathcal{A}$ in order to find the sensing cost of L.

Clearly, $scost(q_0) = 2$ and $scost(q_1) = 1$. We start by computing $lcost(\mathcal{A})$. The corresponding Markov chain $M_{\mathcal{A}}$ has only one ergodic component $\{q_1\}$, so we obtain $lcost(\mathcal{A}) = scost(q_1) = 1$.

The computation of $wcost(\mathcal{A})$ is more intricate. First, note that $\neg a \vee \neg b$ corresponds to 3 letters, $a \wedge b$ to 1 letter, and a to 2 letters.

We have $C_{q_0,q_0}(z) = \frac{1}{1-3z}$, $C_{q_1,q_1}(z) = \frac{1}{1-2z}$, and $C_{q_0,q_1}(z) = C_{q_0,q_0}(z)C_{q_1,q_1}(z) = \frac{1}{(1-3z)(1-2z)}$, whereas $C_{q_1,q_0}(z) = 0$.

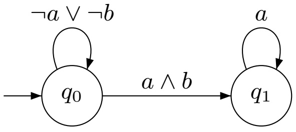

Fig. 6. A DLA for $(\neg a \vee \neg b)^\omega + (\neg a \vee \neg b)^* \cdot (a \wedge b) \cdot a^\omega$.

Moreover, we have

$$S(z) = C_{q_0,q_0}(z) + C_{q_0,q_1}(z) = \frac{2-5z}{(1-3z)(1-2z)}$$

$$F_{q_0}(z) = C_{q_0,q_0}(z)(C_{q_0,q_0}(z) + C_{q_0,q_1}(z)) = \frac{2-5z}{(1-3z)^2(1-2z)}$$

$$F_{q_1}(z) = C_{q_0,q_1}(z)C_{q_1,q_1}(z) = \frac{1}{(1-3z)(1-2z)}.$$

Using standard algorithms on rational functions, we get $[z^n]S(z) \sim \frac{2-5/3}{1-2/3}3^n = 3^n$, $[z^n]F_{q_0}(z) \sim n3^n$ and $[z^n]F_{q_1}(z) \sim 3^{n+1}$. We finally obtain $wcost(\mathcal{A}) = \lim_{n\to\infty} \frac{2\cdot n3^n + 1\cdot 3^{n+1}}{n3^n} = 2$.

Note that *wcost*, unlike *scost* and *lcost*, allows to take into a consideration the cost of transient components when a long word in L is likely to spend time in them. In particular, if the self-loop on q_0 has been labeled by two letters, say by b, rather than by three, then q_0 and q_1 would have participated equally and we would have gotten $wcost(\mathcal{A}) = 3/2$.

6. Synthesis

In the setting of synthesis, the signals in P are partitioned into sets I and O of input and output signals. An I/O-transducer $\mathcal{T}$ senses only input signals and we define its sensing cost as the sensing cost of the DLA $\mathcal{A}_\mathcal{T}$ it induces.

We define the *I/O-sensing cost* of a realizable specification $L \in (2^{I\cup O})^\omega$ as the minimal cost of an I/O-transducer that realizes L. Thus, $scost_{I/O}(L) = \inf\{scost(\mathcal{T}) : \mathcal{T}$ is an I/O-transducer that realizes $L\}$. In this section we consider the problem of finding a minimally-sensing I/O-transducer that realizes a language L given by a DLA.

The realizability problem for a DLA specifications can be solved in polynomial time. Indeed, given a DLA $\mathcal{A}$, we can view $\mathcal{A}$ as a game between a system, which controls the outputs, and an environment, which controls the inputs. We look for a strategy for the system that never reaches an undefined transition. This amounts to solving a turn-based safety game, which can be done in polynomial time.

When sensing is introduced, it is not enough for the system to win this game, as it now has to win while minimizing the sensing cost. Intuitively, not sensing some inputs introduces incomplete information to the game: once the system gives up sensing, it may not know the state in which the game is and knows instead only a set of states in which the game may be. In particular, unlike usual realizability, a strategy that minimizes the sensing need not use the state space of the DLA. We start with an example illustrating this.

Example 38. Consider the DLA $\mathcal{A}$ appearing in Fig. 7. The DLA is over $I = \{p, q\}$ and $O = \{a\}$. A realizing transducer over the structure of $\mathcal{A}$ (see $\mathcal{T}_1$ in Fig. 8)

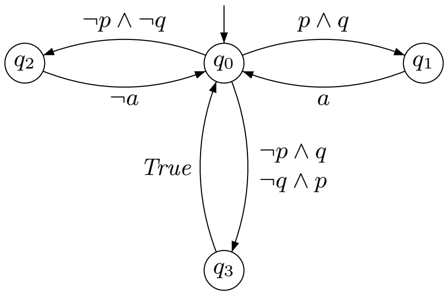

Fig. 7. The DLA $\mathcal{A}$ in Example 38.

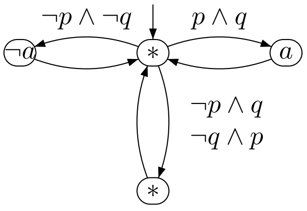

Fig. 8. The transducer $\mathcal{T}_1$ for $\mathcal{A}$.

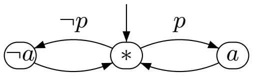

Fig. 9. The transducer $\mathcal{T}_2$ for $\mathcal{A}$.

senses p and q, responds with a if $p \wedge q$ was sensed and responds with $\neg a$ if $\neg p \wedge \neg q$ was sensed. In case other inputs are sensed, the response is arbitrary (denoted $*$ in the figure). As $\mathcal{T}_1$ demonstrates, every transducer that is based on the structure of $\mathcal{A}$ senses two input signals (both p and q) every second step, thus its sensing cost is 1. As demonstrated by the transducer $\mathcal{T}_2$ in Fig. 9, it is possible to realize $\mathcal{A}$ with sensing cost of $\frac{1}{2}$ by only sensing p every second step. Indeed, knowing the value of p is enough in order to determine the output. Note that $\mathcal{T}_2$ may output sometimes a and sometimes $\neg a$ after reading assignments that causes $\mathcal{A}$ to reach q_3. Such a behavior cannot be exhibited by a transducer with the state-structure of $\mathcal{A}$.

Solving games with incomplete information is typically done by some kind of a subset-construction, which involves an exponential blow up. Unlike usual games with incomplete information, here the strategy of the system should not only take care of the realizability but also decides which input signals should be sensed, where the goal is to obtain a minimally sensing transducer. In order to address these multiple objectives, we first construct an MDP in which the possible policies are all winning for the system, and corresponds to different choices of sensing. An optimal policy in this MDP then induces a minimally-sensing transducer.

Theorem 39. *Consider a DLA $\mathcal{A}$ over $2^{I\cup O}$. If $\mathcal{A}$ is realizable, then there exists an MDP $\mathcal{M}$ in which an optimal strategy corresponds to a minimally-sensing I/O-transducer that realizes $\mathcal{A}$. The MDP $\mathcal{M}$ has size exponential in $|\mathcal{A}|$ and can be computed in time exponential in $|\mathcal{A}|$.*

Proof. Consider a DLA $\mathcal{A} = \langle 2^{I\cup O}, Q, q_0, \delta\rangle$. We obtain from $\mathcal{A}$ an MDP $\mathcal{M} = \langle \mathcal{S}, \text{START}, A, \text{P}, \mathit{cost}\rangle$, where $\mathcal{S} = (2^Q \times \{0,1,\perp\}^I) \cup \{\text{START}\}$, and $A = 2^I \times 2^O$. Intuitively, when $\mathcal{M}$ is in state $\langle S, \ell\rangle$, for $S \subseteq Q$ and $\ell : I \to \{0,1,\perp\}$, then $\mathcal{A}$ can be in every state in S, and for each input signal $b \in I$, we have that either b is true ($\ell(b) = 1$), b is false ($\ell(b) = 0$), or b is not sensed ($\ell(b) = \perp$). The action (o, i) means that we now output o and in the next state we will sense only inputs in i. For $\star \in \{\perp, 0, 1\}$, we define $\ell_\star = \{b \in I : \ell(b) = \star\}$.

We define the actions so that an action $\langle o, i\rangle$ is available in state $\langle S, \ell\rangle$ if for every $q \in S$ and $i' \subseteq \ell_\perp$, we have that $\delta(q, \ell_1 \cup i' \cup o)$ is defined. That is, an action is available if its o component does not cause $\mathcal{A}$ to get stuck no matter what the assignment to the signals that are not sensed is.

The transition probabilities are defined as follows. Consider a state $\langle S, \ell\rangle$, and an available action $\langle o, i\rangle$. Let $S' = \bigcup_{q\in S} \bigcup_{i'\subseteq \ell_\perp} \{\delta(q, \ell_1 \cup i' \cup o)\}$. Recall that by taking action $\langle o, i\rangle$, we decide that in the next state we will only sense signals in i. For $i \subseteq I$, we say that an assignment $\ell' : I \to \{0,1,\perp\}$ *senses* i if $\ell'_1 \cup \ell'_0 = i$. Note that there are $2^{|i|}$ assignments that sense i. Accordingly, we have $\text{P}(\langle S, \ell\rangle, \langle o, i\rangle, \langle S', \ell'\rangle) = 2^{-|i|}$ for every $\ell' : I \to \{0,1,\perp\}$ that senses i. That is, a transition from $\langle S, \ell\rangle$ with $\langle o, i\rangle$ goes to the set of all possible successors of S under inputs that are consistent with ℓ and the output assignment o, and the ℓ' component is selected with uniform distribution among all assignments that sense i. The cost function depends on the number of signals we sense, thus $\mathit{cost}(\langle S, \ell\rangle) = |\ell_1 \cup \ell_0|$.

Finally, in the state START we only choose an initial set of input signals to sense. Thus, for every ℓ such that $\ell_1 \cup \ell_0$, we have $\text{P}(\text{START}, \langle o, i\rangle, \langle \{q_0\}, \ell\rangle) = 2^{-|i|}$. Note that START is not reachable from any state in $\mathcal{M}$, and thus its cost is irrelevant. We arbitrarily set $\mathit{cost}(\text{START}) = 0$.

We now turn to prove that $\mathit{cost}(\mathcal{M}) = \mathit{scost}_{I,O}(\mathcal{A})$ and that a minimal-cost policy f in $\mathcal{M}$ induces a minimally-sensing I/O-transducer that realizes $\mathcal{A}$. Intuitively, we prove this by showing a correspondence between transducers and policies, such that the sensing cost of a transducer $\mathcal{T}$ equals the value of the policy it corresponds to in $\mathcal{M}$.

Consider a memoryless minimal-cost policy f. We construct from f a transducer $\mathcal{T} = \langle I, O, \mathcal{S}, \text{START}, \mu, \rho\rangle$, where μ and ρ are defined as follows. For sets $i_1 \subseteq i \subseteq I$, we say that an assignment $\ell : I \to \{0,1,\perp\}$ the (unique) i-sensed i_1-true assignment if for every signal $b \in I$, we have that $\ell(b)$ is $\perp$ if $b \notin i$, is 1 if $b \in i_1$, and is 0 if $b \in i \backslash i_1$.

Let $f(\text{START}) = \langle o_0, i_0\rangle$. We arbitrarily[e] set $\rho(\text{START})$ to o_0. For input $i \in 2^I$, we set $\mu(\text{START}, i) = \langle\{q_0\}, \ell_0\rangle$, for the i_0-sensed i-true assignment ℓ_0.

Next, consider a state $\langle S, \ell\rangle$ and an input $i \in 2^I$. Let $\langle o, i'\rangle = f(\langle S, \ell\rangle)$. We define $\rho(\langle S, \ell\rangle) = o$ and $\mu(\langle S, \ell\rangle, i) = \langle S', \ell'\rangle$, where $S' = \bigcup_{q\in S} \bigcup_{i_1 \subseteq \ell_\perp} \delta(q, \ell_1 \cup i_1 \cup o)$ and ℓ' is the i'-sensed i-true assignment.

We claim that $scost(\mathcal{T}) = cost(f)$. To see this, let $\mathcal{M}'$ be the Markov Chain obtained from $\mathcal{M}$ by fixing the action in each state according to f, and let $\mathcal{T}'$ be the Markov chain obtained from $\mathcal{T}$ by assigning uniform distributions to the input signals. It is easy to see that the Markov chains $\mathcal{M}'$ and $\mathcal{T}'$ are identical (with the exception of START, which, as we mentioned, does not affect the cost). Thus, $cost(f) = scost(\mathcal{T})$. Since $cost(\mathcal{M}) = cost(f)$, we can conclude that there is a transducer $\mathcal{T}$ that realizes $\mathcal{A}$ and for which $scost(\mathcal{T}) = cost(\mathcal{M})$. Thus, $scost_{I,O}(\mathcal{A}) \le cost(\mathcal{M})$.

Conversely, consider a transducer $\mathcal{T}$ for $\mathcal{A}$. By following the set of sensed input signals and the output at each state, $\mathcal{T}$ induces a (possibly non-memoryless) policy f in $\mathcal{M}$. Moreover, as above, $cost(f) = scost(\mathcal{T})$. Thus, $scost(\mathcal{T}) \ge cost(\mathcal{M})$. Since this holds for all transducers $\mathcal{T}$, it follows that $scost_{I,O}(\mathcal{A}) \ge cost(\mathcal{M})$.

Finally, we observe that the size of $\mathcal{M}$ is single exponential in the size of $\mathcal{A}$, and that we can construct $\mathcal{M}$ in time exponential in the size of $\mathcal{A}$. □

Theorem 40. *A minimally-sensing transducer for a realizable DLA $\mathcal{A}$ has size tightly exponential in $|\mathcal{A}|$.*

Proof. The upper bound follows from Theorem 4 applied to the MDP constructed in Theorem 39.

For the lower bound, we describe a family of realizable DLAs $\mathcal{A}_1, \mathcal{A}_2, \ldots$ such that for all $k \ge 1$, the DLA $\mathcal{A}_k$ has $1 + \sum_{i=1}^{k} p_i$ states, yet a minimally-sensing transducer for it requires at least $\prod_{i=1}^{k} p_i$ states, where $p_1, p_2, \ldots$ are prime numbers. Intuitively, $\mathcal{A}_k$ is constructed as follows. In the initial state q_{reset}, the inputs signals determine a number $1 \le i \le k$, and $\mathcal{A}_k$ moves to component i, which consists of a cycle of length p_i. In every state j in component i, the output signals must acknowledge that $\mathcal{A}_k$ is in state $0 \le j < p_i$ of component i. Furthermore, we force a sensing of 1 in every state except for q_{reset} by requiring a signal to be acknowledged in every step. Finally, we can go back to q_{reset} only with a special output signal, which can be outputted only in state 0 of an i component.

Thus, a realizing transducer essentially only chooses which signals to read in q_{reset}. We show that 0 bits can be read, but in that case we need $\prod_{i=1}^{k} p_i$ states. Indeed, the transducer needs to keep track of the location in all the i components simultaneously, which means keeping track of the modulo from each p_i. Since every combination of such modulos is possible, the transducer needs $\prod_{i=1}^{k} p_i$ states.

[e]Since the output in START is ignored, this is indeed arbitrary.

We now turn to formalize this intuition. We define $\mathcal{A}_k = \{2^{I\cup O}, Q, q_{\text{reset}}, \delta\}$, where

- $I = \{i_1, \ldots, i_{\lceil \log k \rceil}\} \cup \{dI\}$, and we view 2^I as $\{1, \ldots, k\} \times \{dI, \neg dI\}$. Then, $O = \bigcup_{i=1}^{k} \{o_{i,1}, \ldots, o_{i,\lceil \log p_i \rceil}\} \cup \{dO, e\}$, and we view it as $(\times_{i=1}^{k}\{0, \ldots, p_i - 1\}) \times \{dO, \neg dO\} \times \{e, \neg e\}$. Thus, a letter $\sigma \in 2^{I\cup O}$ is a pair $\sigma = \langle \sigma_I, \sigma_O \rangle$ with $\sigma_I = \langle m, dI \rangle$ where $1 \le m \le k$ and $dI \in \{0, 1\}$, and with $\sigma_O = \langle r_1, \ldots, r_k, dO, e \rangle$ with $0 \le r_i < p_i - 1$ for all $1 \le i \le k$, $dO \in \{0, 1\}$, and $e \in \{0, 1\}$.
- $Q = \{q_{\text{reset}}\} \cup \bigcup_{i=1}^{k} \{q_{i,0}, \ldots, q_{i,p_i-1}\}$.
- The transition function δ is defined as follows. Consider a letter $\sigma = \langle \langle m, dI \rangle, \langle r_1, \ldots, r_k, dO, e \rangle \rangle$. In state q_{reset}, we have $\delta(q_{\text{reset}}, \sigma) = q_{m,0}$. For a state $q_{i,j}$, we have

$$\delta(q_{i,j}, \sigma) = \begin{cases} q_{i,(j+1) \bmod p_i} & e = 0 \wedge r_i = j \wedge dI = dO \\ q_{\text{reset}} & j = 0 \wedge e = 1 \wedge r_i = j \wedge dI = dO \end{cases}$$

  Note that $\delta(q_{i,j}, \sigma)$ is undefined in all other cases, thus when $dI \neq dO$, when $j \neq 0$ and $e = 1$, and when $r_i \neq j$.

Intuitively, in order to take a transition from $q_{i,j}$, r_i has to match j, and $dI = dO$. Providing that, if $e = 0$ then the transition progresses along the cycle in the i component, and if $e = 1$ and the state is $q_{i,0}$, then the run moves to q_{reset}.

Consider a transducer $\mathcal{T}$ with $scost(\mathcal{T}) < 1$ that realizes $\mathcal{A}_k$. We show that $\mathcal{T}$ must have at least $\prod_{i=1}^{k} p_i$ states. Observe that whenever $\mathcal{A}_k$ is in a state that is different from q_{reset}, the transducer $\mathcal{T}$ must sense at least dI in order to match dO. Thus, the only state that $\mathcal{A}_k$ visits and in which the sensing cost can be lower than 1 (i.e. 0) is q_{reset}. Thus, $\mathcal{T}$ senses 0 in q_{reset}, and moreover, $\mathcal{A}_k$ has to visit q_{reset} infinitely often in order for the 0 sensing to reduce the sensing cost. Sensing 0 in q_{reset} means that in the next step, $\mathcal{A}_k$ can be in $q_{i,0}$ for every $1 \le i \le k$. Since $\mathcal{T}$ has to output $r_i = j$ in every $q_{i,j}$, then $\mathcal{T}$ has to keep track of j, which runs from 0 to $p_i - 1$. Since $\mathcal{T}$ does not "know" the value of i, then $\mathcal{T}$ has to keep track of every reachable combination of $r_1, \ldots, r_k$ with $0 \le r_i \le p_i - 1$ for every $1 \le i \le k$. From the Chinese remainder theorem, every such possible combination of $r_1, \ldots, r_k$ is reachable. Thus, $\mathcal{T}$ needs at least $\prod_{i=1}^{k} p_i$ states.

Finally, note that that there is a transducer $\mathcal{T}$ that realizes $\mathcal{A}_k$. Indeed, $\mathcal{T}$ has $\prod_{i=1}^{k} p_i$ states and outputs e, thus causing $\mathcal{A}_k$ to return to q_{reset}, every $\prod_{i=1}^{k} p_i$ steps. □

We now turn to study the complexity of the problem of finding a minimally-sensing transducer. By the construction in Theorem 39 and the polynomial time algorithm from Theorem 4, we have the following.

Theorem 41. *Consider a realizable DLA $\mathcal{A}$ over $2^{I\cup O}$. We can calculate $scost_{I,O}(\mathcal{A})$ and return a minimally-sensing I/O-transducer that realizes $\mathcal{A}$ in time exponential in $|\mathcal{A}|$.*

In order to complete the picture, we consider the corresponding decision problem. Given a DLA $\mathcal{A}$ over $2^{I \cup O}$ and a threshold γ, the sensing problem in the open setting is to decide whether $scost_{I,O}(\mathcal{A}) < \gamma$.

Theorem 42. *The sensing problem for DLAs in the open setting is EXPTIME-complete.*

Proof. The upper bound follows from Theorem 41. For the lower bound, we show that the problem is EXPTIME hard even for a fixed γ. Given a DLA specification $\mathcal{A}$ over $2^{I \cup O}$, we show that it is EXPTIME-hard to decide whether there exists a transducer $\mathcal{T}$ that realizes $\mathcal{A}$ with $scost(\mathcal{T}) < 1$. We show a reduction from the problem of deciding the nonemptiness of an intersection of finite deterministic tree automata proved to be EXPTIME-hard in [10]. The idea is similar to that of Theorem 40, where a reset state is used to select an object, and a transducer can ignore the inputs in this state by using a response which is acceptable in every possible selected object.

A deterministic automaton on finite trees (DFT) is $\mathcal{U} = \langle \Sigma, Q, \delta, q_0, F \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q \times Q$ is a transition function, and $F \subseteq Q$ is a set of accepting states. We refer to the left and right components of δ as $\delta_{\triangleleft}$ and $\delta_{\triangleright}$. For example, when $\delta(q, \sigma) = \langle q_l, q_r \rangle$, we write $\delta_{\triangleleft}(q, \sigma) = q_l$. An DFT runs on Σ-*trees*. A (binary) Σ-tree is $T = \langle \tau, \ell \rangle$ where $\tau \subseteq \{\triangleleft, \triangleright\}^*$ is prefix-closed: for every $x \cdot \sigma \in \tau$ it holds that $x \in \tau$, and $\ell : \tau \to \Sigma$ is a labeling function. For simplicity, we require that for every $x \in \tau$, either $\{x\triangleleft, x\triangleright\} \subseteq \tau$, or $\{x\triangleleft, x\triangleright\} \cap \tau = \emptyset$, in which case x is a leaf. Given a tree $T = \langle \tau, \ell \rangle$, the *run* of $\mathcal{U}$ on T is a Q-tree $\langle \tau, \ell' \rangle$ where $\ell'(\epsilon) = q_0$, and for every $x \in \tau$ such that x is not a leaf, we have $\delta(\ell'(x), \ell(x)) = \langle \ell'(x\triangleleft), \ell'(x\triangleright) \rangle$. A run is *accepting* if every leaf is labeled by an accepting state. A Σ-tree T is accepted by $\mathcal{U}$ if the run of $\mathcal{U}$ on T is accepting.

The nonempty-intersection problem is to decide, given as input DFTs $\mathcal{U}_1, \ldots, \mathcal{U}_n$, whether their intersection is nonempty, that is whether $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$. Given $\mathcal{U}_1, \ldots, \mathcal{U}_n$, we construct a specification DLA $\mathcal{A}$ such that $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$ iff $scost_{I,O}(\mathcal{A}) < 1$. We assume w.l.o.g. that $L(\mathcal{U}_t) \neq \emptyset$ for all $1 \leq t \leq n$.

We construct $\mathcal{A}$ as follows. Initially, the inputs specify an index $1 \leq t \leq n$. Then, the transducer should respond with a tree in $L(\mathcal{U}_t)$. This is done by challenging the transducer with a branch in the tree, until some reset input signal is true, and the process repeats. Now, if $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$, the transducer can ignore the input signals that specify the index t and just repeatedly output a tree in the intersection. On the other hand, if $\bigcap_{t=1}^{n} L(\mathcal{U}_t) = \emptyset$, the transducer must sense some information about the specified index.[f]

We now formalize this intuition. For $1 \leq t \leq n$, let $\mathcal{U}_t = \langle 2^J, Q^t, \delta^t, q_0^t, F^t \rangle$. Note that we assume w.l.o.g that the alphabet of all the DFTs is 2^J. We construct a

[f]Note that since a tree in the intersection of DFTs may be exponentially bigger than the DFTs, the lower bound here also suggests an alternative lower bound to the exponential size of a minimally-sensed transducer, now with a polynomial set of signals (as opposed to the proof of Theorem 40).

specification DLA $\mathcal{A} = \langle 2^{I\cup O}, Q, q_0, \delta\rangle$ as follows. The set of states of $\mathcal{A}$ is $Q = \bigcup_{t=1}^{n} Q^t \cup \{\textsc{reset}\}$. Assume w.l.o.g that $n = 2^k$ for some $k \in \mathbb{N}$. We define $I = \{b_1, \ldots, b_k\} \cup \{dI\}$ and $O = J \cup \{dO, e\}$. The input signal dI and the output signal dO denote the direction of branching in the tree. For clarity, in an input letter $i \in I$ we write $i(dI) = \triangleleft$ (and $i(dI) = \triangleright$) to indicate that $dI \notin i$ (and $dI \in i$). We use a similar notation for dO.

We define the transition function as follows. In state RESET, we view the inputs $b_1, \ldots, b_k$ as a binary encoding of a number $t \in \{1, \ldots, n\}$. Then, $\delta(\textsc{reset}, t) = q_0^t$. Next, consider a state $q \in Q^t$, and consider letters $i \subseteq I$ and $o \subseteq O$. We define δ as follows:

$$\delta(q, i \cup o) = \begin{cases} \textsc{reset} & q \in F \wedge e \in o \wedge o(dO) = i(dI) \\ \delta^t_{\triangleleft}(q, o \cap J) & e \notin o \wedge o(dO) = i(dI) = \triangleleft \\ \delta^t_{\triangleright}(q, o \cap J) & e \notin o \wedge o(dO) = i(dI) = \triangleright \end{cases}$$

Note that $\delta(q, i \cup o)$ is undefined when $o(dO) \neq i(dI)$ or when $q \notin F$ and $e \in o$. Intuitively, in state RESET, an index $1 \leq t \leq n$ is chosen. From then on, in a state $q \in Q^t$, we simulate the run of $\mathcal{U}_t$ on the left or right branch of the tree, depending on the signal dI. The next letter is outputted in o, and additionally, we require that dO matches dI.

We claim that $scost_{I,O}(\mathcal{A}) < 1$ iff $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$. In the first direction, assume that $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$, and let T be a tree such that $T \in \bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$. Consider the following transducer $\mathcal{T}$: in the state RESET it does not sense any inputs, and then it outputs a branch of T according to the signal dI, while always acknowledging the dI bit with the correct dO. When the end of the branch is reached, it outputs e. Since T is accepted by every DFT U^t, it follows that $\mathcal{T}$ realizes $\mathcal{A}$. Moreover, let l be the longest branch in T, then every l steps at most, $\mathcal{T}$ visits a state corresponding to RESET, in which it senses nothing. Thus, $\mathcal{T}$ senses 1 for at most l steps, and then 0. It follows that $scost(\mathcal{T}) \leq \frac{l}{l+1} = 1 - \frac{1}{l+1} < 1$.

Conversely, observe that in every state $q \in Q \backslash \{\textsc{reset}\}$, a realizing transducer must sense at least 1 signal, namely dI. Thus, the only way to get sensing cost of less than 1 is to visit RESET infinitely often (in fact, with bounded sparsity), and to sense 0 in RESET. However, sensing 0 in RESET means that the next state could be the initial state of any of the n DFTs. Moreover, visiting RESET again means that at some point e was outputted in an accepting state of one of the DFTs. Thus, the transducer outputs a tree that is accepted in every DFT, so $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$.

Finally, observe that the reduction is clearly polynomial, and thus we conclude that deciding whether $scost_{I,O}(\mathcal{A}) < 1$ is EXPTIME-hard. □

6.1. *Non-uniform distributions*

In Remark 28 we address non-uniform distributions for the *scost* setting. While there the adaptation of the proofs is straightforward, in the case of *lcost* and *wcost*, and in the open setting, one needs to proceed with care to get similar results. In this section we consider non-uniform distributions, and show that incorporating them in the model does not involve any significant changes in our techniques. We start by defining the model.

In the closed setting, we are given a *labeled Markov chain* $\mathcal{M} = \langle S, P, \ell \rangle$ where ℓ is a labeling function $\ell : S \to \Sigma$. The probability of a letter $l \in \Sigma$ in state s is $P_s(l) = \sum_{s' \in S : \ell(s') = l} P(s, s')$. We assume that the probability has full support. That is, for every $s \in S$ and $l \in \Sigma$ it holds that $P_s(l) > 0$, and that all probabilities are rational. We lift the probability to words as follows. A sequence of states $\pi = s_1, \ldots, s_n$ induces the word $w_\pi = \ell(s_1) \cdots \ell(s_n)$. The probability of a word w from a state s is then $P_s(w) = \sum_{\pi : w = w_\pi} P_s(\pi)$, where $P_s(\pi) = \prod_{1 \leq i \leq n} P(s_{i-1}, s_i)$, where we set $s_0 = s$ and $\pi = s_1, \ldots, s_n$. The distribution on Σ^* is lifted to a distribution on Σ^ω based on cylinder sets, in the usual manner.

In the open setting, the specification is over $2^{I \cup O}$. We are given an MDP $\mathcal{M} = \langle S, s_0, (A_s)_{s \in S}, \mathrm{P}, cost \rangle$ and a labeling function $\ell : S \to 2^{AP}$, where $A_s = 2^O$ for every $s \in S$. Then, for a sequence of outputs $\rho \in (2^O)^*$ (which corresponds to a strategy for the MDP), the induced Markov chain defined the probability space on $(2^I)^*$, similarly to the above.

In the following we explain how to adapt the various results of the paper to the setting of non-uniform inputs.

6.1.1. *The letter based approach — adapting Theorem 32*

In the letter based approach, we still sample letters, restricting to those that keep us within the language. However, instead of sampling them uniformly, we need to take into account the Markov chain describing the distribution. Thus, we are given a DLA $\mathcal{A} = \langle \Sigma, Q, \delta, q_0 \rangle$ and a labeled Markov chain $\mathcal{M} = \langle S, P, \ell \rangle$ as above. Instead of constructing $\mathcal{M}_\mathcal{A}$ as in Sec. 5, we construct $\mathcal{M}_\mathcal{A}$ as follows. We start with the product of $\mathcal{A}$ and $\mathcal{M}$. That is, the state space is $Q \times S$, there is a transition between $\langle q, s \rangle$ and $\langle q', s' \rangle$ if $\delta(q, \sigma) = q'$ in $\mathcal{A}$, and the probability of the transition is determined according to $\mathcal{M}$. Since δ is only a partial function, we re-normalize the transition probabilities. Computing the sensing cost $lcost(\mathcal{A})$ under $\mathcal{M}$ proceeds by finding a limiting distribution in this chain, similarly to Theorem 32.

6.1.2. *The letter based approach — adapting Theorem 33*

Computing $wcost(\mathcal{A})$ is done similarly to the case of a uniform distribution, with the difference being the construction of the generating function $C_{p,q}(z)$. Instead of having the coefficient of z^n being $\#paths(p, q, n)$, we need to account for the probabilities of the different paths. To do so, we consider again the product of $\mathcal{A}$ and the states S of the Markov chain describing the distribution. Consider a state $\langle q, s \rangle$. Since the transition probabilities in $Q \times S$ are assumed to be rational, we assume that these probabilities are all multiples of some common denominator p. Then, when computing the number of paths from $\langle p, s\rangle$ to $\langle q, s'\rangle$, we count each path according to its multiple of p. The rest of the computation is the same as the proof of Theorem 33.

6.1.3. *The open setting — adapting Theorem 39*

Consider a realizable DLA specification $\mathcal{A}$ over $2^{I\cup O}$, and an MDP $\mathcal{M} = \langle S, s_0, (A_s)_{s\in S}, \mathrm{P}, cost\rangle$ describing the input distribution. Instead of directly constructing an MDP from $\mathcal{A}$ (using a uniform distribution), we construct the MDP from $\mathcal{A}$ using the probabilities defined by $\mathcal{M}$, as follows. The state space of the MDP is $\mathcal{S} = (2^Q \times \{0, 1, \bot\}^I \times S) \cup \{\textsc{start}\}$. Next, Consider a state $\langle R, \ell, s\rangle$ and an action $\langle o, i\rangle$. For an available transition to state $\langle R', \ell', s'\rangle$ the transition probability is defined as the probability to read ℓ'_1 in s'. That is, $P_s(\ell'_1)$. The rest of the proof follows the same lines as that of Theorem 39. We note that the size of a memoryless strategy may now depend (polynomially) on the size of the distribution MDP.

Acknowledgments

The paper gives an overview of the technical results in the papers [3] and [4]. The research leading to these results has received funding from the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013, ERC Grant No. 278410).

References

[1] S. Almagor, U. Boker and O. Kupferman, Discounting in LTL, in *Proc. 20th TACAS*, LNCS 8413 (Springer, 2014), pp. 424–439.
[2] G. Avni and O. Kupferman, When Does Abstraction Help? in *IPL* **113** (2013) 901–905.
[3] S. Almagor, D. Kuperberg and O. Kupferman, Regular sensing, in *Proc. 34th FST & TCS*, *LIPIcs* **29** (2014) 161–173.
[4] S. Almagor, D. Kuperberg and O. Kupferman, The sensing cost of monitoring and synthesis, in *Proc. 35th FST & TCS*, *LIPIcs* **35** (2015) 380–393.
[5] K. Chatterjee and R. Majumdar, Minimum attention controller synthesis for omega-regular objectives, in *FORMATS* (2011) 145–159.
[6] K. Chatterjee, R. Majumdar and T. A. Henzinger, Controller synthesis with budget constraints, in *Proc. 11th International Workshop on Hybrid Systems: Computation and Control*, LNCS 4981 (Springer, 2008), pp. 72–86.
[7] D. L. Donoho, Compressed sensing, *IEEE Trans. Inform. Theory* **52** (2006) 1289–1306.
[8] P. Flajolet and R. Sedgewick, Analytic combinatorics: Functional equations, rational and algebraic functions, 2001.
[9] C. Grinstead and J. Laurie Snell, 11:Markov chains, in *Introduction to Probability* (American Mathematical Society, 1997).
[10] J. Goubault, Rigid E-unifiability is DEXPTIME-complete, In *9th LICS* (1994) 498–506.

[11] K. Havelund and G. Rosu, Efficient monitoring of safety properties, *STT&T* **6**(2) (2004) 18–173.
[12] G. Kindler, *Property Testing, PCP, and Juntas*, PhD thesis (Tel Aviv University, 2002).
[13] O. Kupferman and M. Y. Vardi, Church's problem revisited, *The Bulletin of Symbolic Logic* **5**(2) (1999) 245–263.
[14] E. Kushilevitz and N. Nisan, *Communication Complexity* (Cambridge University Press, 1997).
[15] C. Mauduit and A. Sárköz, On finite pseudorandom binary sequences. I. Measure of pseudorandomness, the legendre symbol, *Acta Arith.* **82**(4) (1997) 365–377.
[16] S. Muthukrishnan, Theory of data stream computing: where to go, in *Proc. 30th PODS* (2011), pp. 317–319.
[17] D. Niwinski and I. Walukiewicz, Relating hierarchies of word and tree automata, in *STACS*, LNCS 1373 (Springer, 1998).
[18] A. Pnueli and R. Rosner, On the synthesis of a reactive module, in *Proc. 16th POPL* (1989), pp. 179–190.
[19] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (John Wiley & Sons, 2014).
[20] M. F. Roy and A. Szpirglas, Complexity of the computation on real algebraic numbers, *J. Symb. Comput.* **10**(1) (1990) 39–52.
[21] S. Schewe, Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete, in *Proc. 30th FST & TCS*, LIPIcs 8 (2010), pp. 400–411.
[22] A. P. Sistla, Safety, liveness and fairness in temporal logic, *Formal Aspects of Computing* **6** (1994) 495–511.