Robustness Analysis of String Transducers[*]

Roopsha Samanta[1], Jyotirmoy V. Deshmukh[2**], and Swarat Chaudhuri[3]

[1] University of Texas at Austin roopsha@cs.utexas.edu
[2] University of Pennsylvania djy@cis.upenn.edu
[3] Rice University swarat@rice.edu

Abstract. Many important functions over strings can be represented as finite-state string transducers. In this paper, we present an automata-theoretic technique for algorithmically verifying that such a function is *robust* to uncertainty. A function encoded as a transducer is defined to be robust if for each small (i.e., bounded) change to any input string, the change in the transducer's output is proportional to the change in the input. Changes to input and output strings are quantified using weighted generalizations of the Levenshtein and Manhattan distances over strings. Our main technical contribution is a set of decision procedures based on reducing the problem of robustness verification of a transducer to the problem of checking the emptiness of a reversal-bounded counter machine. The decision procedures under the generalized Manhattan and Levenshtein distance metrics are in PSPACE and EXPSPACE, respectively. For transducers that are Mealy machines, the decision procedures under these metrics are in NLOGSPACE and PSPACE, respectively.

1 Introduction

Many tasks in computing involve the evaluation of functions from strings to strings. Such functions are often naturally represented as finite-state string transducers [12, 17, 2, 21]. For example, inside every compiler is a transducer that maps user-written text to a string over tokens, and authors of web applications routinely write transducers to sanitize user input. Systems for natural language processing use transducers for executing morphological rules, correcting spelling, and processing speech. Many of the string algorithms at the heart of computational biology or image processing are essentially functional transducers.

The transducer representation of functions has been studied thoroughly over the decades, and many decision procedures and expressiveness results about them are known [17, 21]. Less well-studied, however, is the behavior of finite-state transducers under *uncertain inputs.* The data processed by real-world transducers often contains small amounts of error or uncertainty. The real-world images handled by image processing engines are frequently noisy, DNA strings

[*] This research was partially supported by CCC-CRA Computing Innovation Fellows Project, NSF Award 1162076 and NSF CAREER award 1156059.

[**] Jyotirmoy Deshmukh is now a researcher at Toyota Technical Center, Gardena, CA.

that transducers in computational biology process may be incomplete or incorrectly sequenced, and text processors must account for wrongly spelled keywords. Clearly, it is desirable that such random noise in the input does not cause a transducer to behave unpredictably. However, this is not mandated by traditional correctness properties: a transducer may have a "correct" execution trace on every individual input, but its output may be highly sensitive to even the minutest perturbation to these inputs.

One way to ensure that a transducer behaves reliably on uncertain inputs is to show that it is *robust*, as formalized in [15, 4, 6]. Informally, robustness means that small perturbations to the transducer's inputs can only lead to small changes in the corresponding outputs. In this paper, we present an automata-theoretic technique for verifying that a given functional transducer is robust in this sense.

Our definition of robustness of (functional) transducers is inspired by the analytic notion of *Lipschitz continuity*. Recall that a function f over a metric space (let us say with distance metric d) is K-Lipschitz if for all x, y, we have $d(f(x), f(y)) \leq Kd(x, y)$. Intuitively, a Lipschitz function responds proportionally, and hence robustly, to changes in the input. In our model, a transducer is robust if the function encoded by the transducer satisfies a property very similar to Lipschitz-continuity. The one difference between the Lipschitz criterion and ours is that the output of a Lipschitz-continuous function changes proportionally to *every* change to the input, however large. From the modeling point of view, this requirement seems too strong: if the input is noisy beyond a certain point, it makes little sense to constrain the behavior of the output. Accordingly, we define robustness of a transducer $\mathcal{T}$ with respect to a certain threshold B on the amount of input perturbation—given constants B, K and a distance metric d over strings, $\mathcal{T}$ is (B, K)-robust if for all x, y: $d(x, y) \leq B \Rightarrow d(\mathcal{T}(x), \mathcal{T}(y)) \leq Kd(x, y)$.

Our main technical contribution is a set of decision procedures based on reducing the problem of verifying (B, K)-robustness of a transducer to the problem of checking the emptiness of a reversal-bounded counter machine. Naturally, whether a transducer is robust or not depends on the distance metric used. We present decision procedures to verify robustness under two distance metrics that are weighted generalizations of the well-known Manhattan and Levenshtein distances over strings. Our decision procedures under these metrics are in PSPACE and EXPSPACE, respectively. When the transducer in question is restricted to be a Mealy machine, we present simpler decision procedures under these metrics that are in NLOGSPACE and PSPACE, respectively.

The rest of the paper is organized as follows. In Sec. 2, we present our formal models and definitions. In Sec. 3, we present a class of *distance-tracking automata* that are central to our decision procedures, presented in Sec. 4. We conclude with a discussion of related work in Sec. 5.

2 Preliminaries

In this section, we define our transducer models, distance metrics and our notion of robustness. In what follows, we use the following notation. Input strings are typically denoted by lowercase letters s, t etc. and output strings by s', t' etc. We denote the concatenation of strings s and t by $s.t$, the i^{th} character of string s by $s[i]$, the substring $s[i].s[i+1].\ldots.s[j]$ by $s[i,j]$, the length of the string s by $|s|$, and the empty string and empty symbol by ϵ.

Functional Transducers. The transducers considered in this paper may be nondeterministic, but must define functions between regular sets of strings. Formally, a *transduction* R from a finite alphabet Σ to a finite alphabet Γ is an arbitrary subset of $\Sigma^\star \times \Gamma^\star$. We use $R(s)$ to denote the set $\{t \mid (s,t) \in R\}$. We say that a transduction is *functional* if $\forall s \in \Sigma^\star$, $|R(s)| \leq 1$.

A *finite transducer* (FT) is a finite-state device with two tapes: a read-only input tape and a write-only output tape. It scans the input tape from left to right; in each step, it reads an input symbol, nondeterministically chooses its next state, writes a corresponding finite string to the output tape, and advances its reading head by one position to the right. The output of an FT is the string on the output tape if the FT finishes scanning the input tape in some designated final state. Formally, a finite transducer $\mathcal{T}$ is a tuple $(Q, \Sigma, \Gamma, q_0, E, F)$ where Q is a finite nonempty set of states, q_0 is the initial state, $E \subseteq Q \times \Sigma \times \Gamma^\star \times Q$ is a set of transitions, and F is a set of final states[4].

A run of $\mathcal{T}$ on a string $s = s[0]s[1]\ldots s[n]$ is defined in terms of the sequence: (q_0, w'_0), $(q_1, w'_1), \ldots, (q_n, w'_n)$, (q_{n+1}, ϵ) where for each $i \in [0,n]$, $(q_i, s[i], w'_i, q_{i+1})$ is a transition in E. A run is called accepting if $q_{n+1} \in F$. The output of $\mathcal{T}$ along a run is the string $w'_0.w'_1.\ldots.w'_n$ if the run is accepting, and is undefined otherwise. The transduction computed by an FT $\mathcal{T}$ is the relation $[\![\mathcal{T}]\!] \subseteq \Sigma^\star \times \Gamma^\star$, where $(s,s') \in [\![\mathcal{T}]\!]$ iff there is an accepting run of $\mathcal{T}$ on s with s' as the output along that run. $\mathcal{T}$ is called *single-valued* or *functional* if $[\![\mathcal{T}]\!]$ is functional. Checking if an arbitrary FT is functional can be done in polynomial time [10]. The input language, L, of a functional transducer $\mathcal{T}$ is the set $\{s \mid [\![\mathcal{T}]\!](s) \text{ is defined}\}$. When viewed as a relation over $\Sigma^\star \times \Gamma^\star$, $[\![\mathcal{T}]\!]$ defines a partial function; however, when viewed as a relation over $L \times \Gamma^\star$, $[\![\mathcal{T}]\!]$ is a total function.

Mealy Machines. These are deterministic, symbol-to-symbol, functional transducers. The notion of determinism is the standard one, and a symbol-to-symbol transduction means that for every transition of the form (q, a, w', q'), $|w'| = 1$. The input language L of a Mealy machine $\mathcal{T}$ is the set $\Sigma^\star$ (i.e., every state is accepting). Thus, the transduction $[\![\mathcal{T}]\!] : \Sigma^\star \to \Gamma^\star$ is a total function.

In what follows, we use the term finite transducers, or simply transducers, to refer to both functional transducers and Mealy machines, and distinguish between them as necessary. As a technicality that simplifies our proofs, we assume that for all $i > |s|$, $s[i] = \texttt{\#}$, where # is a special end-of-string symbol not in Σ or Γ.

Distance Metrics. A *metric space* is an ordered pair (M, d), where M is a set and $d : M \times M \to \mathbb{R}$, the distance metric, is a function with the properties:

[4] Some authors prefer to call this model a *generalized sequential machine*, and define transducers to allow ϵ-transitions, i.e., the transducer can change state without moving the reading head. Note that we disallow ϵ-transitions.

(1) $d(x,y) \geq 0$, (2) $d(x,y) = 0$ iff $x = y$, (3) $\forall x,y : d(x,y) = d(y,x)$, and (4) $\forall x,y,z : d(x,z) \leq d(x,y) + d(y,z)$.

The Hamming distance and Levenshtein distance metrics are often used to measure distances (or similarity) between strings. The Hamming distance, defined for two equal length strings, is the minimum number of symbol substitutions required to transform one string into the other. For strings of unequal length, the Hamming distance is replaced by the Manhattan distance or the L_1-norm that also accounts for the difference in the lengths. The Levenshtein distance between two strings is the minimum number of symbol insertions, deletions and substitutions required to transform one string into the other.

The Hamming/Manhattan and Levenshtein distances only track the number of symbol mismatches, and not the degree of mismatch. For some applications, these distance metrics can be too coarse. Hence, we use distance metrics equipped with integer penalties - *pairwise symbol mismatch penalties* for substitutions and a *gap penalty* for insertions/deletions. We denote by $\texttt{diff}(a,b)$ the mismatch penalty for substituting symbols a and b, with $\texttt{diff}(a,b) = 0$ if $a = b$. We require $\texttt{diff}(a,b)$ to be well-defined when either a or b is #. We denote by α the fixed, non-zero gap penalty for insertion or deletion of a symbol. We now define the weighted extensions of the Manhattan and Levenshtein distances formally.

The generalized Manhattan distance is defined by the following recurrence relations, for $i,j \geq 1$, and $s[0] = t[0] = \epsilon$:

$$d_M(s[0],t[0]) = 0 \quad d_M(s[0,j],t[0,j]) = d_M(s[0,j\text{-}1],t[0,j\text{-}1]) + \texttt{diff}(s[j],t[j]). \tag{1}$$

The generalized Levenshtein distance is defined by the following recurrence relations, for $i,j \geq 1$, and $s[0] = t[0] = \epsilon$:

$$\begin{array}{lll} d_L(s[0],t[0]) = 0, & d_L(s[0,i],t[0]) = i\alpha, & d_L(s[0],t[0,j]) = j\alpha \\ d_L(s[0,i],\ t[0,j]) = \min(& d_L\ (s[0,i\text{-}1],\ t[0,j\text{-}1]) + \texttt{diff}(s[i],\ t[j]), & \\ & d_L\ (s[0,i\text{-}1],\ t[0,j]) + \alpha, & \\ & d_L\ (s[0,i],\ t[0,j\text{-}1]) + \alpha). & \end{array} \tag{2}$$

The first three relations in (2), that involve empty strings, are obvious. The generalized Levenshtein distance between the nonempty prefixes, $s[0,i]$ and $t[0,j]$, is the minimum over the distances corresponding to three possible transformations: (1) optimal (generalized Levenshtein) transformation of $s[0,i\text{-}1]$ into $t[0,j\text{-}1]$ and substitution of $s[i]$ with $t[j]$, with a mismatch penalty of $\texttt{diff}(s[i],t[j])$, (2) optimal transformation of $s[0,i\text{-}1]$ into $t[0,j]$ and deletion of $s[i]$, with a gap penalty of α, and, (3) optimal transformation of $s[0,i]$ into $t[0,j\text{-}1]$ and insertion of $t[j]$ with a gap penalty of α.

Observe that if $\texttt{diff}(a,b)$ is defined to be 1 for $a \neq b$ and 0 otherwise, the above definitions correspond to the usual Manhattan and Levenshtein distances, respectively. In our work, $\texttt{diff}(a,b)$ and α are external parameters provided to the algorithm by the user, and we require that the resulting generalized Manhattan and Levenshtein distances are distance metrics.

Robustness. As explained in Sec. 1, our notion of robustness for finite transducers is an adaptation of the analytic notion of Lipschitz continuity, and is defined with respect to a fixed bound on the amount of input perturbation.

Definition 2.1 (Robust String Transducers). *Given an upper bound B on the input perturbation, a constant K and a distance metric $d : \Sigma^* \times \Sigma^* \cup \Gamma^\star \times \Gamma^\star \rightarrow \mathbb{N}$, a transducer $\mathcal{T}$ defined over a regular language $L \subseteq \Sigma^*$, with $[\![\mathcal{T}]\!] : L \rightarrow \Gamma^\star$, is called (B, K)-robust if:*

$$\forall \delta \leq B, \forall s, t \in L : \ d(s,t) = \delta \ \Rightarrow \ d([\![\mathcal{T}]\!](s), [\![\mathcal{T}]\!](t)) \leq K\delta.$$

3 Distance Tracking Automata

In Sec. 4, we show how to reduce the problem of verifying robustness of finite transducers to the problem of checking emptiness of carefully constructed composite machines. A key component of these constructions are machines that can track the generalized Manhattan or Levenshtein distance between two strings. Our earlier work [18] presents automata constructions for tracking the usual Manhattan and Levenshtein distances. In this section, we first briefly review reversal-bounded counter machines and then adapt our distance tracking automata constructions for the generalized versions of the distance metrics.

3.1 Review: Reversal-bounded Counter Machines [13, 14]

A (one-way, nondeterministic) h-counter machine $\mathcal{A}$ is a (one-way, nondeterministic) finite automaton, augmented with h integer counters. Let G be a finite set of integer constants (including 0). In each step, $\mathcal{A}$ may read an input symbol, perform a test on the counter values, change state, and increment each counter by some constant $g \in G$. A test on a set of integer counters $Z = \{z_1, \ldots, z_h\}$ is a Boolean combination of tests of the form $z\theta g$, where $z \in Z$, $\theta \in \{\leq, \geq, =, <, >\}$ and $g \in G$. Let T_Z be the set of all such tests on counters in Z.

Formally, $\mathcal{A}$ is defined as a tuple $(\Sigma, X, x_0, Z, G, E, F)$ where Σ, X, x_o, F, are the input alphabet, set of states, initial state, and final states respectively. Z is a set of h integer counters, and $E \subseteq X \times (\Sigma \cup \epsilon) \times T_Z \times X \times G^h$ is the transition relation. Each transition $(x, \sigma, t, x', g_1, \ldots, g_h)$ denotes a change of state from x to x' on symbol $\sigma \in \Sigma \cup \epsilon$, with $t \in T_Z$ being the enabling test on the counter values, and $g_k \in G$ being the amount by which the k^{th} counter is incremented.

A configuration μ of a one-way multi-counter machine is defined as the tuple $(x, \sigma, z_1, \ldots, z_h)$, where x is the state of the automaton, σ is a symbol of the input string being read by the automaton and $z_1, \ldots, z_h$ are the values of the counters. We define a move relation $\rightarrow_{\mathcal{A}}$ on the configurations: $(x, \sigma, z_1, \ldots, z_h) \rightarrow_{\mathcal{A}} (x', \sigma', z'_1, \ldots, z'_h)$ iff $(x, \sigma, t(z_1, \ldots, z_h), x', g_1, \ldots, g_h) \in E$, where, $t(z_1, \ldots, z_h)$ is *true*, $\forall k$: $z'_k = z_k + g_k$, and σ' is the next symbol in the input string being read. A path is an element of $\rightarrow^\star_{\mathcal{A}}$, i.e., a path is a finite sequence of configurations $\mu_1, \ldots, \mu_m$ where for all $j : \mu_j \rightarrow_{\mathcal{A}} \mu_{j+1}$. A string $s \in \Sigma^\star$ is accepted by $\mathcal{A}$ if $(x_0, s[0], 0, \ldots 0) \rightarrow^\star_{\mathcal{A}} (x, s[j], z_1, \ldots, z_h)$, for some $x \in F$ and $j \leq |s|$ (we make no assumptions about $z_1, \ldots, z_h$ in the accepting configuration). The set of strings (language) accepted by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$.

In general, multi-counter machines can simulate actions of Turing machines (even with just 2 counters). In [13], the author presents a class of counter machines - *reversal-bounded* counter machines - with efficiently decidable properties. A counter is said to be in the increasing mode between two successive configurations if the counter value is the same or increasing, and in the decreasing mode if the counter value is strictly decreasing. We say that a counter is *r-reversal bounded* if the maximum number of times it changes mode (from increasing to decreasing and vice versa) along *any* path is r. We say that a multi-counter machine $\mathcal{A}$ is r-reversal bounded if each of its counters is at most r-reversal bounded. We denote the class of h-counter, r-reversal-bounded machines by NCM(h, r).

Lemma 3.1. *[11] The nonemptiness problem for $\mathcal{A}$ in class NCM(h, r) can be solved in* NLogspace *in the size of $\mathcal{A}$.*

Recall that for all $i > |s|$, $s[i] = \texttt{#}$. In what follows, let $\Sigma^{\texttt{#}} = \Sigma \cup \{\texttt{#}\}$.

3.2 Automaton for Tracking Generalized Manhattan Distance

We now define automata $\mathcal{D}_M^{=\delta}$, $\mathcal{D}_M^{>\delta}$ that accept pairs of strings (s, t) such that $d_M(s, t) = \delta$, $d_M(s, t) > \delta$, respectively, where $d_M(s, t)$ is the Manhattan distance between s and t. The automata $\mathcal{D}_M^{=\delta}$, $\mathcal{D}_M^{>\delta}$ are 1-reversal-bounded 1-counter machines (i.e., in NCM(1,1)), and are each defined as a tuple $(\Sigma^{\texttt{#}} \times \Sigma^{\texttt{#}}, X, x_0, Z, G, E, \{acc\})$, where $(\Sigma^{\texttt{#}} \times \Sigma^{\texttt{#}})$ is the input alphabet, $X = \{x_0, x, acc\}$, is a set of three states, x_0 is the initial state, $Z = \{z\}$ is a single 1-reversal-bounded counter, $G = \{\delta, 0\} \cup \cup_{a,b \in \Sigma^{\texttt{#}}} \{\texttt{diff}(a, b)\}$ is a set of integers, and $\{acc\}$ is the singleton set of final states. The transition relations of $\mathcal{D}_M^{=\delta}$, $\mathcal{D}_M^{>\delta}$ both include the following transitions:

1. An *initialization transition* $(x_0, (\epsilon, \epsilon), true, x, \delta)$ that sets the counter z to δ.
2. Transitions of the form $(x, (a, a), z \geq 0, x, 0)$, for $a \neq \texttt{#}$, that read a pair of identical, non-# symbols, and leave the state and counter unchanged.
3. Transitions of the form $(x, (a, b), z \geq 0, x, -\texttt{diff}(a, b))$, for $a \neq b$, which read a pair (a, b) of distinct symbols, and decrement the counter z by the corresponding mismatch penalty $\texttt{diff}(a, b)$.
4. Transitions of the form $(acc, (*, *), *, acc, 0)$, which ensure that the machine stays in its final state upon reaching it.

The only difference in the transition relations of $\mathcal{D}_M^{=\delta}$, $\mathcal{D}_M^{>\delta}$ is in their transitions into accepting states. The *accepting transitions* of $\mathcal{D}_M^{=\delta}$ are of the form $(x, (\texttt{#}, \texttt{#}), z = 0, acc, 0)$, and move $\mathcal{D}_M^{=\delta}$ to an accepting state upon reading a $(\texttt{#}, \texttt{#})$ pair when the counter value is zero, i.e., when the Manhattan distance between the strings being read is exactly equal to δ. The accepting transitions of $\mathcal{D}_M^{>\delta}$ are of the form $(x, (*, *), z < 0, acc, 0)$, and move $\mathcal{D}_M^{>\delta}$ to an accepting state whenever the counter value goes below zero, i.e., when the Manhattan distance between the strings being read is greater than δ.

Lemma 3.2. *$\mathcal{D}_M^{=\delta}$, $\mathcal{D}_M^{>\delta}$ accept a pair of strings (s, t) iff $d_M(s, t) = \delta$, $d_M(s, t) > \delta$, respectively.*

Note: The size of $\mathcal{D}_M^{=\delta}$ or $\mathcal{D}_M^{>\delta}$ is $O(\delta + |\Sigma|^2 \texttt{MAX}_{\texttt{diff}_\Sigma})$, where $\texttt{MAX}_{\texttt{diff}_\Sigma}$ is the maximum mismatch penalty over Σ.

3.3 Automaton for Tracking Generalized Levenshtein Distance

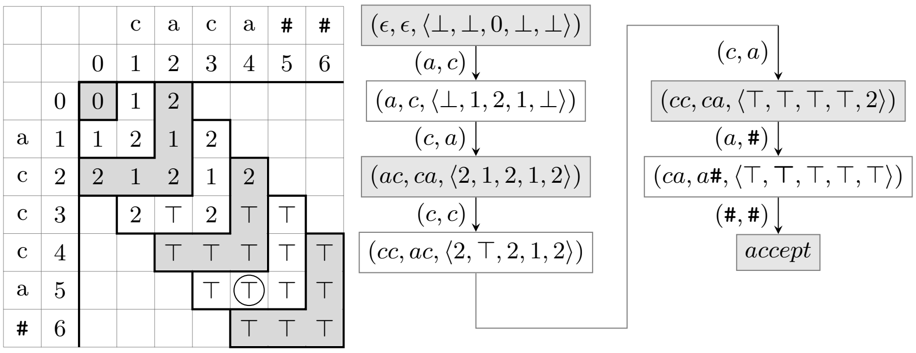
The standard dynamic programming-based algorithm for computing the Levenshtein distance $d_L(s,t)$ can be extended naturally to compute the generalized Levenshtein distance using the recurrence relations in (2). This algorithm organizes the bottom-up computation of the generalized Levenshtein distance with the help of a table $\mathbf{t}$ of height $|s|$ and width $|t|$. The 0^{th} row and column of $\mathbf{t}$ account for the base case of the recursion. The $\mathbf{t}(i,j)$ entry stores the generalized Levenshtein distance of the strings $s[0,i]$ and $t[0,j]$. In general, the entire table has to be populated in order to compute $d_L(s,t)$. However, when one is only interested in some bounded distance δ, then for every i, it suffices to compute values for the cells from $\mathbf{t}(i, i-\delta)$ to $\mathbf{t}(i, i+\delta)$ [12]. We call this region the δ-*diagonal* of $\mathbf{t}$, and use this observation to construct DFA's $\mathcal{D}_L^{=\delta}$, $\mathcal{D}_L^{>\delta}$ that accept pairs of strings (s,t) such that $d_L(s,t) = \delta$, $d_L(s,t) > \delta$, respectively[5].

In each step, $\mathcal{D}_L^{=\delta}$, $\mathcal{D}_L^{>\delta}$ read a pair of input symbols and change state to mimic the bottom-up edit distance computation by the dynamic programming algorithm. As in the case of Manhattan distance, $\mathcal{D}_L^{=\delta}$, $\mathcal{D}_L^{>\delta}$ are identical, except for their accepting transitions. Formally, $\mathcal{D}_L^{=\delta}$, $\mathcal{D}_L^{>\delta}$ are each defined as a tuple $(\Sigma^{\#} \times \Sigma^{\#}, Q, q_0, \Delta, \{acc\})$, where $(\Sigma^{\#} \times \Sigma^{\#})$, Q, q_0, Δ, $\{acc\}$ are the input alphabet, the set of states, the initial state, the transition function and the singleton set of final states respectively. A state is defined as the tuple $(x, y, \mathbf{e})$, where x and y are strings of length at most δ and $\mathbf{e}$ is a vector containing $2\delta + 1$ entries, with values from the set $\{0, 1, \ldots, \delta, \bot, \top\}$). A state of $\mathcal{D}_L^{=\delta}$, $\mathcal{D}_L^{>\delta}$ maintains the invariant that if i symbol pairs have been read, then x, y store the last δ symbols of s, t (i.e., $x = s[i\text{-}\delta\text{+}1, i]$, $y = t[i\text{-}\delta\text{+}1, i]$), and the entries in $\mathbf{e}$ correspond to the values stored in $\mathbf{t}(i,i)$ and the cells within the δ-diagonal, above and to the left of $\mathbf{t}(i,i)$. The values in these cells greater than δ are replaced by $\top$. The initial state is $q_0 = (\epsilon, \epsilon, \langle \bot, \ldots, \bot, 0, \bot, \ldots, \bot \rangle)$, where ϵ denotes the empty string, $\bot$ is a special symbol denoting an undefined value, and the value 0 corresponds to entry $\mathbf{t}(0,0)$. Upon reading the i^{th} input symbol pair, say (a,b), $\mathcal{D}_L^{=\delta}$, $\mathcal{D}_L^{>\delta}$ transition from state $q_{i\text{-}1} = (x_{i\text{-}1}, y_{i\text{-}1}, \mathbf{e}_{i\text{-}1})$ to a state $q_i = (x_i, y_i, \mathbf{e}_i)$ such that x_i, y_i are the δ-length suffices of $x_{i\text{-}1}.a$, $y_{i\text{-}1}.b$, respectively, and $\mathbf{e}_i$ is the appropriate set of entries in the δ-diagonal of $\mathbf{t}$ computed from $x_{i\text{-}1}$, $y_{i\text{-}1}$, $\mathbf{e}_{i\text{-}1}$, the input symbol pair and the relevant mismatch/gap penalties (for more details, see [18]).

Finally, upon reading the symbol $(\#, \#)$ in state $(x, y, \mathbf{e})$, we add transitions to the single accepting state *acc* in $\mathcal{D}_L^{=\delta}$ (and in $\mathcal{D}_L^{>\delta}$) iff:

- $|s| = |t|$, i.e., x and y do not contain #, and the $(\delta+1)^{th}$ entry in $\mathbf{e}$ is δ ($\top$ in the case of $\mathcal{D}_L^{>\delta}$), or,

[5] The fact that there exists a DFA that accepts string pairs within bounded (generalized) Levenshtein distance from each other follows from results in [8, 9]. However, these theorems do not provide a constructive procedure for such an automaton.



			c	a	c	a	#	#
		0	1	2	3	4	5	6
	0	0	1	2				
a	1	1	2	1	2			
c	2	2	1	2	1	2		
c	3		2	⊤	2	⊤	⊤	
c	4			⊤	⊤	⊤	⊤	⊤
a	5				⊤	⊤	⊤	⊤
#	6					⊤	⊤	⊤

$(\epsilon, \epsilon, \langle \bot, \bot, 0, \bot, \bot \rangle)$
(a, c)
$(a, c, \langle \bot, 1, 2, 1, \bot \rangle)$
(c, a)
$(ac, ca, \langle 2, 1, 2, 1, 2 \rangle)$
(c, c)
$(cc, ac, \langle 2, \top, 2, 1, 2 \rangle)$
(c, a)
$(cc, ca, \langle \top, \top, \top, \top, 2 \rangle)$
$(a, \texttt{\#})$
$(ca, a\texttt{\#}, \langle \top, \top, \top, \top, \top \rangle)$
$(\texttt{\#}, \texttt{\#})$
accept

Fig. 3.1: Dynamic programming table emulated by $\mathcal{D}_L^{>2}$. The table $\mathbf{t}$ filled by the dynamic programming algorithm for $\delta = 2$ is shown to the left, and a computation of $\mathcal{D}_L^{>2}$ on the strings $s = accca$, $t = caca$ is shown to the right. Here, $\Sigma = \{a, b, c\}$, $\texttt{diff}(a, b) = \texttt{diff}(b, c) = \texttt{diff}(a, \texttt{\#}) = 1$, $\texttt{diff}(a, c) = \texttt{diff}(b, \texttt{\#}) = 2$, $\texttt{diff}(c, \texttt{\#}) = 3$ and $\alpha = 1$.

- $|s| = |t| + \ell$, i.e., y contains ℓ #'s, x contains no #, and the $(\delta + 1 - \ell)^{th}$ entry in $\mathbf{e}$ is δ ($\top$ in the case of $\mathcal{D}_L^{>\delta}$), or,
- $|t| = |s| + \ell$, i.e., x contains ℓ #'s, y contains no #, and the $(\delta + 1 + \ell)^{th}$ entry in $\mathbf{e}$ is δ ($\top$ in the case of $\mathcal{D}_L^{>\delta}$).

Upon reaching *acc*, $\mathcal{D}_L^{=\delta}$, $\mathcal{D}_L^{>\delta}$ remains in it.

Example Run. A run of $\mathcal{D}_L^{>2}$ on the string pair $s = accca$, $t = caca$ that checks if $d_L(s, t) > 2$, is shown in Fig. 3.1. The mismatch and gap penalties are as enumerated in the caption.

The following lemma states the correctness of these constructions. The proof follows from the state-invariants maintained by $\mathcal{D}_L^{=\delta}$, $\mathcal{D}_L^{>\delta}$ and their accepting transitions.

Lemma 3.3. *$\mathcal{D}_L^{=\delta}$, $\mathcal{D}_L^{>\delta}$ accept a pair of strings (s, t) iff $d_L(s, t) = \delta$, $d_L(s, t) > \delta$, respectively.*

Note: The size of $\mathcal{D}_L^{=\delta}$ or $\mathcal{D}_L^{>\delta}$ is $O((\delta|\Sigma|)^{4\delta})$.

4 Robustness analysis

From Definition 2.1, it follows that checking (B, K)-robustness of a transducer $\mathcal{T}$ is equivalent to checking if *for each* $\delta \leq B$, $\forall s, t \in L : d(s, t) = \delta \implies d(\llbracket \mathcal{T} \rrbracket(s), \llbracket \mathcal{T} \rrbracket(t)) \leq K\delta$. Thus, we focus on the problem of checking robustness of a transducer for some *fixed input perturbation* δ. We reduce this problem to checking language emptiness of a product machine $\mathcal{A}^\delta$ constructed from (1) an input automaton $\mathcal{A}_I^\delta$ that accepts a pair of strings (s, t) iff $d(s, t) = \delta$, (2) a pair-transducer $\mathcal{P}$ that transforms input string pairs (s, t) to output string pairs

(s',t') according to $\mathcal{T}$, and (3) an output automaton $\mathcal{A}_O^\delta$ that accepts (s',t') iff $d(s',t') > K\delta$. We construct $\mathcal{A}^\delta$ such that $\mathcal{T}$ is robust iff for all $\delta \leq B$, the language of $\mathcal{A}^\delta$ is empty.

Later in this section, we present specialized constructions for $\mathcal{A}_I^\delta$, $\mathcal{A}_O^\delta$ for checking robustness of Mealy machines and functional transducers, with respect to the generalized Manhattan and Levenshtein distances. The definition of the pair-transducer $\mathcal{P}$ is standard in all these scenarios, and hence we present it first. We next define the product machine $\mathcal{A}^\delta$ for two relevant scenarios. Scenario 1 is when $\mathcal{A}_I^\delta$ and $\mathcal{A}_O^\delta$ are both DFAs - as we will see, this scenario presents itself while checking robustness of either type of transducer with respect to the generalized Levenshtein distance. Scenario 2 is when $\mathcal{A}_I^\delta$ and $\mathcal{A}_O^\delta$ are both 1-reversal-bounded counter machines - this scenario presents itself while checking robustness of either type of transducer with respect to the generalized Manhattan distance.

Recall that $\Sigma^\# = \Sigma \cup \{\#\}$. Let $\Gamma^\# = \Gamma \cup \{\#\}$, $\Gamma^{\epsilon,\#} = \Gamma \cup \{\epsilon, \#\}$, $\widetilde{\Sigma} = \Sigma^\# \times \Sigma^\#$ and $\widetilde{\Gamma} = \Gamma^{\epsilon,\#} \times \Gamma^{\epsilon,\#}$.

Pair-transducer, $\mathcal{P}$. Given a transducer $\mathcal{T}$, the pair-transducer $\mathcal{P}$ reads an input string pair and produces an output string pair according to $\mathcal{T}$. Formally, given $\mathcal{T} = (Q, \Sigma, \Gamma, q_0, E, F)$, $\mathcal{P}$ is defined as the tuple $(Q_\mathcal{P}, \widetilde{\Sigma}, \widetilde{\Gamma}, q_{0_\mathcal{P}}, E_\mathcal{P}, F_\mathcal{P})$ where $Q_\mathcal{P} = Q \times Q$, $q_{0_\mathcal{P}} = (q_0, q_0)$, $F_\mathcal{P} = F \times F$, and, $E_\mathcal{P}$ is the set of all transitions of the form $((q_1,q_2),(a,b),(w',v'),(q_1',q_2'))$ such that $(q_1, a,\ w', q_1') \in E$ and $(q_2, b, v', q_2') \in E$. While for Mealy machines, in all transitions in $E_\mathcal{P}$, w', v' are symbols in $\Gamma \cup \{\#\}$, for arbitrary functional transducers, w', v' may be strings of different lengths, and either or both could be ϵ. We define the function $[\![\mathcal{P}]\!]$ such that $[\![\mathcal{P}]\!](s,t) = (s',t')$ if $[\![\mathcal{T}]\!](s) = s'$ and $[\![\mathcal{T}]\!](t) = t'$.

Product machine, $\mathcal{A}^\delta$. Given input automaton $\mathcal{A}_I^\delta$, pair transducer $\mathcal{P}$ and output automaton $\mathcal{A}_O^\delta$, the product machine $\mathcal{A}^\delta$ is constructed to accept all string pairs (s,t) such that (s,t) is accepted by $\mathcal{A}_I^\delta$ and there exists a string pair (s',t') accepted by $\mathcal{A}_O^\delta$ with $(s',t') = [\![\mathcal{P}]\!](s,t)$. Notice that while in each of its transitions, $\mathcal{A}_O^\delta$ can only read a pair of *symbols* at a time, each transition of $\mathcal{P}$ potentially generates a pair of (possibly unequal length) output *strings*. Hence, $\mathcal{A}^\delta$ cannot be constructed as a simple synchronized product.

Scenario 1. Given a DFA input automaton $\mathcal{A}_I^\delta = (Q_I, \widetilde{\Sigma}, q_{0_I}, \Delta_I, F_I)$, pair transducer $\mathcal{P} = (Q_\mathcal{P}, \widetilde{\Sigma}, \widetilde{\Gamma}, q_{0_\mathcal{P}}, E_\mathcal{P}, F_\mathcal{P})$ and a DFA output automaton $\mathcal{A}_O^\delta = (Q_O, \widetilde{\Gamma}, q_{0_O}, \Delta_O, F_O)$, $\mathcal{A}^\delta$ is a DFA given by the tuple $(Q, \widetilde{\Sigma}, q_0, \Delta, F)$, where $Q \subseteq Q_I \times Q_\mathcal{P} \times Q_O$, $q_0 = (q_{0_I}, q_{0_\mathcal{P}}, q_{0_O})$, $F = F_I \times F_\mathcal{P} \times F_O$, and E is defined as follows:
$\Delta((q_I, q_\mathcal{P}, q_O), (a,b)) = (q_I', q_\mathcal{P}', q_O')$ iff

1. $\Delta_I(q_I, (a,b)) = q_I'$, and
2. there exist w', v' such that
   (a) $(q_\mathcal{P}, (a,b), (w',v'), q_\mathcal{P}') \in E_\mathcal{P}$, and
   (b) $\Delta_O^*(q_O, (w',v')) = q_O'$.

Scenario 2. For counter machines, one also needs to keep track of the counters. Given input automaton $\mathcal{A}_I^\delta$ in NCM(h_I,1), of the form $(\widetilde{\Sigma}, X_I, x_{0_I}, Z_I, G_I, E_I, F_I)$, pair transducer $\mathcal{P} = (Q_\mathcal{P}, \widetilde{\Sigma}, \widetilde{\Gamma}, q_{0_\mathcal{P}}, E_\mathcal{P}, F_\mathcal{P})$ and output automaton $\mathcal{A}_O^\delta$ in class NCM(h_O,1), of the form $(\widetilde{\Gamma}, X_O, x_{0_O}, Z_O, G_O, E_O, F_O)$, $\mathcal{A}^\delta$ is in NCM(h,1), with

$h = h_I + h_O$, and is given by the tuple $(\widetilde{\Sigma}, X, x_0, Z, G, E, F)$, where $X \subseteq X_I \times Q_{\mathcal{P}} \times X_O$, $x_0 = (x_{0_I}, q_{0_{\mathcal{P}}}, x_{0_O})$, $Z = Z_I \cup Z_O$, $G = G_I \cup G_O$, $F = F_I \times F_{\mathcal{P}} \times F_O$, and E is defined as follows:
$((x_I, q_{\mathcal{P}}, x_O), (a, b), t, (x'_I, q'_{\mathcal{P}}, x'_O), g_{I1}, \ldots, g_{Ih_I}, g_{O1}, \ldots, g_{Oh_O}) \in E$ iff

1. $(x_I, (a, b), t_I, x'_I, g_{I1}, \ldots, g_{Ih_I}) \in E_I$ with $t \Rightarrow t_I$, and
2. there exist w', v' such that
   (a) $(q_{\mathcal{P}}, (a, b), (w', v'), q'_{\mathcal{P}}) \in E_{\mathcal{P}}$, and
   (b) $(x_O, (w'[0], v'[0]), z_{O1}, \ldots, z_{Oh_O}) \rightarrow^{\star}_{\mathcal{A}^{\delta}_{O}} (x'_O, (w'[j], v'[\ell]), z'_{O1}, \ldots, z'_{Oh_O})$, with $j = |w'| - 1$, $\ell = |v'| - 1$, $t \Rightarrow t_O$ where t_O is the enabling test corresponding to the first move along $\rightarrow^{\star}_{\mathcal{A}^{\delta}_{O}}$ and $\forall k$: $z'_{Ok} = z_{Ok} + g_{Ok}$.

4.1 Mealy Machines

Generalized Manhattan Distance. For a Mealy machine $\mathcal{T}$, it is easy to see from the descriptions of $\mathcal{A}^{\delta}_{I}$, $\mathcal{A}^{\delta}_{O}$ and from the constructions in Sec. 3.2 that $\mathcal{A}^{\delta}_{I}$ is the same as $\mathcal{D}^{=\delta}_{M}$ and $\mathcal{A}^{\delta}_{O}$ is essentially the same as $\mathcal{D}^{>K\delta}_{M}$, with the alphabet being $\widetilde{\Gamma}$. Thus, $\mathcal{A}^{\delta}_{I}$ and $\mathcal{A}^{\delta}_{O}$ are both in NCM(1,1). Let $\mathcal{A}^{\delta}$ be the product machine, as defined in *Scenario 2* using $\mathcal{A}^{\delta}_{I}$, $\mathcal{P}$ and $\mathcal{A}^{\delta}_{O}$. From Lemma 3.2 and the definition of $\mathcal{A}^{\delta}$, it follows that $\mathcal{A}^{\delta}$ accepts all input strings (s, t) such that $d_M(s, t) = \delta$, and there exists $(s', t') = [\![\mathcal{P}]\!](s, t)$ with $d_M(s', t') > K\delta$. Thus, any pair of input strings accepted by $\mathcal{A}^{\delta}$ is a witness to the non-robustness of $\mathcal{T}$; equivalently $\mathcal{T}$ is robust iff $\mathcal{A}^{\delta}$ is empty for all $\delta \leq B$.

The product machine $\mathcal{A}^{\delta}$ is in NCM(2, 1) and its size is polynomial in $size(\mathcal{T})$, δ, K, $|\Sigma|$, $|\Gamma|$ and $\texttt{MAX}_{\texttt{diff}}$, where $\texttt{MAX}_{\texttt{diff}}$ is the maximum mismatch penalty over Σ and Γ. Since, we need to check nonemptiness of $\mathcal{A}^{\delta}$ for all $\delta \leq B$, we have the following theorem using Lemma 3.1.

Theorem 4.1. *Robustness verification of a Mealy machine $\mathcal{T}$ with respect to the generalized Manhattan distance can be accomplished in* NLogspace *in $size(\mathcal{T})$, B, K, $|\Sigma|$, $|\Gamma|$ and* $\texttt{MAX}_{\texttt{diff}}$ *(maximum mismatch penalty).*

Generalized Levenshtein Distance. For a Mealy machine $\mathcal{T}$, $\mathcal{A}^{\delta}_{I}$ is the same as $\mathcal{D}^{=\delta}_{L}$ and $\mathcal{A}^{\delta}_{O}$ is the same as $\mathcal{D}^{>K\delta}_{L}$, with alphabet $\widetilde{\Gamma}$. Thus, $\mathcal{A}^{\delta}_{I}$ and $\mathcal{A}^{\delta}_{O}$ are both dfas. Let $\mathcal{A}^{\delta}$ be a product machine, as defined in *Scenario 1* using $\mathcal{A}^{\delta}_{I}$, $\mathcal{P}$ and $\mathcal{A}^{\delta}_{O}$. As before, from Lemma 3.3 and the definition of $\mathcal{A}^{\delta}$, it follows that $\mathcal{T}$ is robust iff $\mathcal{A}^{\delta}$ is empty for all $\delta \leq B$.

The size of $\mathcal{A}^{\delta}$ is $O(size^2(\mathcal{T})|\Sigma|^{4\delta}(|\Gamma|K)^{4K\delta}\delta^{4\delta(1+K)})$. Since the emptiness of the dfa $\mathcal{A}^{\delta}$ can be checked in NLogspace in the size of $\mathcal{A}^{\delta}$, and we need to repeat this for all $\delta \leq B$, we have the following theorem.

Theorem 4.2. *Robustness verification of a Mealy machine $\mathcal{T}$ with respect to the generalized Levenshtein distance can be accomplished in* PSpace *in B and K.*

4.2 Functional Transducers

Checking robustness of functional transducers is more involved than checking robustness of Mealy machines. The main reason is that $\mathcal{T}$ may produce output symbols for two strings in an unsynchronized fashion, i.e., the symbols read by $\mathcal{A}_O^\delta$ may be of the form (a, ϵ) or (ϵ, a). While this does not affect the input automata constructions, the output automata for functional transducers differ from the ones for Mealy machines.

Generalized Manhattan Distance. As stated above, $\mathcal{A}_I^\delta$ is the same as $\mathcal{D}_M^{=\delta}$. The construction of $\mathcal{A}_O^\delta$ is based on the observation that if s', t' are mismatched in $1 + K\delta$ positions, $d_M(s', t')$ is guaranteed to be greater than $K\delta$. Let $\eta = 1 + K\delta$. We define $\mathcal{A}_O^\delta$ to be in class NCM($1 + 2\eta$, 1) with a *distance counter* z and two sets of *position counters* $c_1, \ldots, c_\eta$ and $d_1, \ldots, d_\eta$. The counter z is initialized to $K\delta$ and for all j, position counters c_j, d_j are initialized to hold guesses for η *mismatch positions* in s', t', respectively. In particular, the position counters are initialized such that for all j, $c_j = d_j$, $c_j \geq 0$, and $c_j < c_{j+1}$, thereby ensuring that the counter pairs store η *distinct* position guesses [6]. For notational convenience, we denote the initial position guess stored in the position counter c_j (or d_j) by p_j.

Intuitively, for each j, $\mathcal{A}_O^\delta$ uses its position counters to compare the symbols at the p_j^{th} position of each string. For all j, $\mathcal{A}_O^\delta$ decrements c_j, d_j upon reading a nonempty symbol of s', t', respectively. Thus, $\mathcal{A}_O^\delta$ reads the p_j^{th} symbol of s', t' when $c_j = 0$, $d_j = 0$, respectively. If the p_j^{th} symbols are mismatched symbols a, b, then $\mathcal{A}_O^\delta$ decrements the distance counter z by $\mathtt{diff}(a, b)$. Now, recall that the symbol at the p_j^{th} position for one string may arrive before that for the other string. Thus, for instance, c_j may be 0, while d_j is still positive. In this case, $\mathcal{A}_O^\delta$ needs to remember the *earlier* symbol in its state till the *delayed* symbol arrives. Fortunately, $\mathcal{A}_O^\delta$ has to remember at most η symbols corresponding to the η guessed positions. When the delayed symbol at position p_j of the trailing string arrives, i.e. d_j finally becomes 0, $\mathcal{A}_O^\delta$ compares it to the symbol stored in its state and decrements z as needed.

Formally, a state of $\mathcal{A}_O^\delta$ is a tuple of the form (pos, id, vec), where $pos \in [1, \eta]$ is a positive integer (initially 0) that keeps track of the earliest position for which $\mathcal{A}_O^\delta$ is waiting to read symbols of both strings, $id \in \{0, 1, 2\}$ is used to track which of the strings is leading the other, and vec is a η-length vector that stores the symbols of the leading string. Initially, all entries of vec are $\perp$. The invariant maintained by the state is as follows: if $pos = j$, (a) $id = 0$ iff $c_j > 0$, $d_j > 0$ and $vec_j = \perp$, (b) $id = 1$ iff $c_j \leq 0$, $d_j > 0$ and $vec_j = s'[p_j]$, and (c) $id = 2$ iff $c_j > 0$, $d_j \leq 0$ and $vec_j = t'[p_j]$. Thus, if c_j becomes zero while d_j is non-zero, id is set to 1, and vec_j is set to the symbol read, i.e., $s'[p_j]$; when d_j eventually

[6] Note that this can be done nondeterministically as follows. First all 2η counters are incremented by 1, and at some nondeterministically chosen point, the machine stops incrementing the c_1, d_1 counters, then at some further point stops incrementing the c_2, d_2 counters, and so on. This ensures that for each j, $c_j = d_j$, and the higher index counters have higher (distinct) values.

becomes zero due to the p_j^{th} symbol of t' being read, then vec_j is set to $\perp$, z is decremented by $\texttt{diff}(s'[p_j], t'[p_j])$ and pos is incremented. The case when the p_j^{th} symbol of t' is output before that of s' is handled symmetrically. $\mathcal{A}_O^\delta$ moves to an accepting state whenever the value in z goes below 0, i.e. $d_M(s', t') > K\delta$, and stays there. $\mathcal{A}_O^\delta$ moves to a special rejecting state if the value in z is nonnegative and either the string pairs or all position guesses are exhausted, i.e., if $\mathcal{A}_O^\delta$ reads a $(\texttt{\#}, \texttt{\#})$ symbol or if all position counters are negative.

In effect, the construction ensures that if $\mathcal{A}_O^\delta$ accepts a pair of strings (s', t'), then $d_M(s', t') > K\delta$. On the other hand, note that if $d_M(s', t') > K\delta$, then there exists a run of $\mathcal{A}_O^\delta$ in which it correctly guesses some mismatch positions (whose number is at most η) such that their cumulative mismatch penalty exceeds $K\delta$.

Lemma 4.1. *The above $\mathcal{A}_O^\delta$ accepts a pair of strings (s, t) iff $d_M(s, t) > K\delta$.*

Note that the size of $\mathcal{A}_O^\delta$ is $O(\Gamma^{2K\delta})$. Let $\mathcal{A}^\delta$ be a product machine, as defined in *Scenario 2* using $\mathcal{A}_I^\delta$, $\mathcal{P}$ and $\mathcal{A}_O^\delta$. From Lemma 3.2, Lemma 4.1 and the definition of $\mathcal{A}^\delta$, it follows that $\mathcal{T}$ is robust iff $\mathcal{A}^\delta$ is empty for all $\delta \leq B$. $\mathcal{A}^\delta$ is in class NCM($2+2\eta$, 1), and its size is $O(size^2(\mathcal{T})(\delta + |\Sigma|^2 \texttt{MAX}_{\texttt{diff}_\Sigma})\Gamma^{2K\delta})$, with $\texttt{MAX}_{\texttt{diff}}$ being the maximum mismatch penalty over Σ. Since we need to repeat this for all $\delta \leq B$, we have the following theorem using Lemma 3.1.

Theorem 4.3. *Robustness verification of a functional transducer $\mathcal{T}$ with respect to the generalized Manhattan distance can be accomplished in* PSpace *in B, K.*

Generalized Levenshtein distance. The input automaton $\mathcal{A}_I^\delta$ is the same as $\mathcal{D}_L^{=\delta}$. In order to track the generalized Levenshtein distance between the unsynchronized output strings generated by $\mathcal{P}$, $\mathcal{A}_O^\delta$ needs to remember substrings of the leading string in its state, and not simply the symbols at possible mismatch positions. A natural question to ask is whether there exists a bound on the length of the substrings that $\mathcal{A}_O^\delta$ needs to remember in its state. We first address this question before defining $\mathcal{A}_O^\delta$.

Consider $\mathcal{A}_I^\delta \otimes \mathcal{P}$, the synchronous product of the input automaton $\mathcal{A}_I^\delta$ and the pair transducer $\mathcal{P}$. Let $\mathcal{T}_{I\otimes\mathcal{P}} = (Q_{I\otimes\mathcal{P}}, \widetilde{\Sigma}, \widetilde{\Gamma}, q_{0_{I\otimes\mathcal{P}}}, E_{I\otimes\mathcal{P}}, F_{I\otimes\mathcal{P}})$ be obtained by *trimming* $\mathcal{A}_I^\delta \otimes \mathcal{P}$, i.e., by removing all states that are not reachable from the initial state or from which no final state is reachable. The set $E_{I\otimes\mathcal{P}}$ of transitions of $\mathcal{T}_{I\otimes\mathcal{P}}$ can be extended in a natural way to the set $E^*_{I\otimes\mathcal{P}}$ of paths of $\mathcal{T}_{I\otimes\mathcal{P}}$. Note that for any path $(q_{0_{I\otimes\mathcal{P}}}, (w, v), (w', v'), q_{f_{I\otimes\mathcal{P}}})$ from the initial state to some final state $q_{f_{I\otimes\mathcal{P}}} \in F_{I\otimes\mathcal{P}}$, $d_L(w, v) = \delta$ and $[\![\mathcal{P}]\!](w, v) = (w', v')$.

We define the *pairwise-delay* of a path π of $\mathcal{T}_{I\otimes\mathcal{P}}$, denoted $pd(\pi)$, as the difference in lengths of its output string labels: for $\pi = (q, (w, v), (w', v'), q')$, $pd(\pi) = \mathsf{abs}(|w'| - |v'|)$. $\mathcal{T}_{I\otimes\mathcal{P}}$ is said to have *bounded pairwise-delay* if the pairwise-delay of all its paths is bounded. For $\mathcal{T}_{I\otimes\mathcal{P}}$ with bounded pairwise-delay, we denote the maximum pairwise-delay over all paths of $\mathcal{T}_{I\otimes\mathcal{P}}$ by $\mathcal{D}(\mathcal{T}_{I\otimes\mathcal{P}})$. Let ℓ_{max} be the length of the longest output string in any transition of $\mathcal{T}$, i.e., $\ell_{max} = \max\{|w'| \mid (q, a, w', q') \in E\}$, and let Q_I, Q be the set of states of $\mathcal{A}_I^\delta$, $\mathcal{T}$.

Lemma 4.2. *$\mathcal{T}_{I\otimes\mathcal{P}}$ has bounded pairwise-delay, with $\mathcal{D}(\mathcal{T}_{I\otimes\mathcal{P}}) < |Q|^2.|Q_I|\ell_{max}$, iff the pairwise-delay of all cyclic paths in $\mathcal{T}_{I\otimes\mathcal{P}}$ is 0.*

Proof. If there is a cyclic path $c = (q, (w, v), (w', v'), q)$ in $\mathcal{T}_{I \otimes \mathcal{P}}$ with $pd(c) \neq 0$, then for n traversals through c, $pd(c^n) = n(pd(c))$, and hence $\mathcal{D}(\mathcal{T}_{I \otimes \mathcal{P}})$ is not bounded. If for all cycles c, $pd(c) = 0$, then for any path π, $pd(\pi) = pd(\pi_{acy})$, where π_{acy} is the acyclic path obtained from π by iteratively removing all cycles from π. Thus, $\mathcal{D}(\mathcal{T}_{I \otimes \mathcal{P}})$ is bounded by the maximum possible pairwise-delay along any acyclic path of $\mathcal{T}_{I \otimes \mathcal{P}}$. This maximum delay is $(|Q_{I \otimes \mathcal{P}}| - 1)\ell_{max}$ and is exhibited along an acyclic path of maximum length $|Q_{I \otimes \mathcal{P}}| - 1$, with the output string pair along each transition being ϵ and a string of length ℓ_{max}. By definition of $\mathcal{T}_{I \otimes \mathcal{P}}$, $|Q_{I \otimes \mathcal{P}}| \leq |Q|^2.|Q_I|$. The result follows. □

Corollary 1. *$\mathcal{T}_{I \otimes \mathcal{P}}$ has bounded pairwise-delay iff each simple cycle of $\mathcal{T}_{I \otimes \mathcal{P}}$ is labeled with equal length output strings.*

Lemma 4.3. *If $\mathcal{T}_{I \otimes \mathcal{P}}$ does not have bounded pairwise-delay, $\mathcal{T}$ is non-robust.*

Proof. We exhibit a witness for non-robustness of $\mathcal{T}$. If $\mathcal{T}_{I \otimes \mathcal{P}}$ does not have bounded pairwise-delay, then there is some simple cycle $c : (q, (w_c, v_c), (w'_c, v'_c), q)$ in $\mathcal{T}_{I \otimes \mathcal{P}}$ with $|w'_c| \neq |v'_c|$. Consider the paths $\pi_1 = (q_{0_{I \otimes \mathcal{P}}}, (w_1, v_1), (w'_1, v'_1), q)$ and $\pi_2 = (q, (w_2, v_2), (w'_2, v'_2), q_{f_{I \otimes \mathcal{P}}})$, with $q_{f_{I \otimes \mathcal{P}}} \in F_{I \otimes \mathcal{P}}$. Let us assume that $|w'_1| > |v'_1|$, $|w'_c| > |v'_c|$ and $|w'_2| > |v'_2|$ (the other cases can be handled similarly). Let $|w'_c| - |v'_c| = l_c$ and $|w'_1.w'_2| - |v'_1.v'_2| = l$.

Then, given δ, K, there exists $n \in \mathbb{N}$ such that $l + nl_c > K\delta$. The witness path π to non-robustness of $\mathcal{T}$ can now be constructed from π_1, followed by n-traversals of c, followed by π_2. By definition of $\mathcal{T}_{I \otimes \mathcal{P}}$, the generalized Levenshtein distance, $d_L(w_1.(w_c)^n.w_2, v_1.(v_c)^n.v_2)$, of the input string labels of π, equals δ, and by construction of π, the difference in the lengths, and hence the generalized Levenshtein distance, $d_L(w'_1.(w'_c)^n.w'_2, v'_1.(v'_c)^n.v'_2)$ of the output string labels of π exceeds $K\delta$. □

Lemma 4.2 is helpful in constructing an output automaton $\mathcal{A}_O^\delta$ that accepts a pair of output strings (s', t') iff $d_L(s', t') > K\delta$. The construction of $\mathcal{A}_O^\delta$ is very similar to that of $\mathcal{D}_L^{>K\delta}$, defined over alphabet $\widetilde{\Gamma}$, with one crucial difference. Having read the j^{th} symbol of s', in order to compute all entries in the j^{th} row of the $K\delta$-diagonal in the dynamic programming table, we need to have seen the $(j + K\delta)^{th}$ symbol of t'. However, the maximum delay between s' and t' could be as much as $\mathcal{D}(\mathcal{T}_{I \otimes \mathcal{P}})$ (by Lemma 4.2). Hence, unlike $\mathcal{D}_L^{>K\delta}$, which only needs to remember strings of length $K\delta$ in its state, $\mathcal{A}_O^\delta$ needs to remember strings of length $\mathcal{D}(\mathcal{T}_{I \otimes \mathcal{P}}) + K\delta$ in its state. Thus, a state of $\mathcal{A}_O^\delta$ is a tuple $(x, y, \mathbf{e})$, where x and y are strings of length at most $\mathcal{D}(\mathcal{T}_{I \otimes \mathcal{P}}) + K\delta$, and $\mathbf{e}$ is a vector of length $2K\delta + 1$.

Lemma 4.4. *If $\mathcal{T}_{I \otimes \mathcal{P}}$ has bounded pairwise-delay, $\mathcal{A}_O^\delta$ as described above accepts a pair of strings (s', t') iff $d_L(s', t') > K\delta$.*

Note that $\mathcal{A}_O^\delta$ is a DFA with size $O(|\Gamma|^{4(K\delta + \mathcal{D}(\mathcal{T}_{I \otimes \mathcal{P}}))})$, where $\mathcal{D}(\mathcal{T}_{I \otimes \mathcal{P}})$ is the maximum pairwise-delay of $\mathcal{T}$ and is $O(size^2(\mathcal{T})|\Sigma|^{4\delta}\delta^{4\delta}\ell_{max})$. Summarizing our robustness checking algorithm for a functional transducer $\mathcal{T}$, we first check if

$\mathcal{T}_{I\otimes\mathcal{P}}$ does not have bounded pairwise-delay. To do this, we check if there exists a simple cycle c in $\mathcal{T}_{I\otimes\mathcal{P}}$ for which $pd(c) \neq 0$. If yes, $\mathcal{T}$ is non-robust by Lemma 4.3. If not, we construct the product machine $\mathcal{A}^{\delta}$, as defined in *Scenario 1* using $\mathcal{A}_I^{\delta}$, $\mathcal{P}$ and $\mathcal{A}_O^{\delta}$. By Lemma 3.3, Lemma 4.4 and the definition of $\mathcal{A}^{\delta}$, it follows that $\mathcal{T}$, with bounded pairwise-delay, is robust iff $\mathcal{A}^{\delta}$ is empty for all $\delta \leq B$.

Checking if there exists a simple cycle c in $\mathcal{T}_{I\otimes\mathcal{P}}$ with $pd(c) \neq 0$ can be done in NLogspace in the size of $\mathcal{T}_{I\otimes\mathcal{P}}$[7], which is $O(size^2(\mathcal{T})|\Sigma|^{4\delta}\delta^{4\delta})$. Also, the nonemptiness of $\mathcal{A}^{\delta}$ can be checked in NLogspace in its size, as given by the product of $size(\mathcal{T}_{I\otimes\mathcal{P}})$ and $size(\mathcal{A}_O^{\delta})$. Repeating this for all $\delta \leq B$, we have the following theorem.

Theorem 4.4. *Robustness verification of a functional transducer $\mathcal{T}$ with respect to the Levenshtein distance can be accomplished in* ExpSpace *in B.*

5 Related Work

In prior work [15], [4–6] on continuity and robustness analysis, the focus is on checking if the function computed by a program has desirable properties such as Lipschitz continuity. While these papers reason about programs that manipulate numbers, we focus on robustness analysis of programs manipulating strings. As the underlying metric topologies are quite different, the results from prior work and our current approach are complementary.

More recent papers have aimed to develop a notion of robustness for reactive systems. In [19], the authors present polynomial-time algorithms for the analysis and synthesis of robust transducers. Their notion of robustness is one of input-output stability, that bounds the output deviation from disturbance-free behaviour under bounded disturbance, as well as the persistence of the effect of a sporadic disturbance. Also, unlike our distance metrics, their distances are measured using cost functions that map *each* string to a nonnegative integer. In [16, 3, 1], the authors develop different notions of robustness for reactive systems, with ω-regular specifications, interacting with uncertain environments. In [7], the authors present a polynomial-time algorithm to decide robustness of sequential circuits modeled as Mealy machines, w.r.t. a *common suffix distance* metric. Their notion of robustness also bounds the persistence of the effect of a sporadic disturbance.

In recent work in [18], we studied robustness of networked systems in the presence of channel perturbations. While the automata-theoretic framework employed in [18] is similar to the one proposed here, there are important differences in the system model, robustness definitions and the distance metrics. In [18], we tracked the deviation in the output of a synchronous network of Mealy machines, in the presence of channel perturbations, w.r.t. the (non-weighted) Manhattan and Levenshtein distances. As is evident in this paper, tracking distances and checking robustness for arbitrary functional transducers w.r.t. generalized distance metrics present a new set of challenges.

[7] This can be done using a technique similar to the one presented in [20] (Theorem 2.4) for checking nonemptiness of a Büchi automaton.

References

1. Bloem, R., Greimel, K., Henzinger, T., Jobstmann, B.: Synthesizing Robust Systems. In: Formal Methods in Computer Aided Design (FMCAD). pp. 85–92 (2009)
2. Bradley, R.K., Holmes, I.: Transducers: An Emerging Probabilistic Framework for Modeling Indels on Trees. Bioinformatics 23(23), 3258–3262 (2007)
3. Cerny, P., Henzinger, T., Radhakrishna, A.: Simulation Distances. In: Conference on Concurrency Theory (CONCUR). pp. 253–268 (2010)
4. Chaudhuri, S., Gulwani, S., Lublinerman, R.: Continuity Analysis of Programs. In: Principles of Programming Languages (POPL). pp. 57–70 (2010)
5. Chaudhuri, S., Gulwani, S., Lublinerman, R.: Continuity and Robustness of Programs. Communications of the ACM (2012)
6. Chaudhuri, S., Gulwani, S., Lublinerman, R., Navidpour, S.: Proving Programs Robust. In: Foundations of Software Engineering (FSE). pp. 102–112 (2011)
7. Doyen, L., Henzinger, T.A., Legay, A., Ničković, D.: Robustness of Sequential Circuits. In: Application of Concurrency to System Design (ACSD). pp. 77–84 (2010)
8. Eilenberg, S.: Automata, Languages, and Machines, vol. A. Academic Press New York (1974)
9. Frougny, C., Sakarovitch, J.: Rational Relations with Bounded Delay. In: Symposium on Theoretical Aspects of Computer Science (STACS). pp. 50–63 (1991)
10. Gurari, E., Ibarra, O.: A Note on Finite-valued and Finitely Ambiguous Transducers. Mathematical Systems Theory 16(1), 61–66 (1983)
11. Gurari, E.M., Ibarra, O.H.: The Complexity of Decision Problems for Finite-Turn Multicounter Machines. In: International Colloquium on Automata Languages and Programming (ICALP). pp. 495–505 (1981)
12. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press (1997)
13. Ibarra, O.H.: Reversal-Bounded Multicounter Machines and Their Decision Problems. Journal of the ACM 25(1), 116–133 (1978)
14. Ibarra, O.H., Su, J., Dang, Z., Bultan, T., Kemmerer, R.A.: Counter Machines: Decidable Properties and Applications to Verification Problems. In: Mathematical Foundations of Computer Science (MFCS). pp. 426–435 (2000)
15. Majumdar, R., Saha, I.: Symbolic Robustness Analysis. In: IEEE Real-Time Systems Symposium. pp. 355–363 (2009)
16. Majumdar, R., Render, E., Tabuada, P.: A Theory of Robust Software Synthesis. to appear in ACM Transactions on Embedded Computing Systems
17. Mohri, M.: Finite-state Transducers in Language and Speech Processing. Computational Linguistics 23(2), 269–311 (1997)
18. Samanta, R., Deshmukh, J.V., Chaudhuri, S.: Robustness Analysis of Networked Systems. In: Verification, Model Checking, and Abstract Interpretation (VMCAI). pp. 229–247 (2013)
19. Tabuada, P., Balkan, A., Caliskan, S.Y., Shoukry, Y., Majumdar, R.: Input-Output Robustness for Discrete Systems. In: International Conference on Embedded Software (EMSOFT) (2012)
20. Vardi, M.Y., Wolper, P.: Reasoning about Infinite Computations. Information and Computation 115(1), 1–37 (1994)
21. Veanes, M., Hooimeijer, P., Livshits, B., Molnar, D., Bjørner, N.: Symbolic Finite State Transducers: Algorithms and Applications. In: Principles of Programming Languages (POPL). pp. 137–150 (2012)