Refinement and Implementation of Algorithmic Tools for Deciding the Error-detection Property[1]

ALIFASI DAKA and STAVROS KONSTANTINIDIS

Department of Mathematics and Computing Science, Saint Mary's University
Halifax, Nova Scotia, B3H 3C3 Canada
alifasi@hotmail.com, s.konstantinidis@smu.ca

Abstract We consider new and refine existing algorithmic tools for deciding whether a given regular language is error-detecting for a given rational channel, with the aim of providing tractable implementations of these concepts. The algorithmic tools involve product constructions between automata and transducers, a decision procedure for functionality of transducers, and the new concept of pseudo-sequential transducer. Our implementation is written in C++ using Grail-style libraries. To our knowledge this is the first implementation deciding the error-detection property at this level of generality, which brings us one step closer to the development of practical software for investigating the error-detecting capabilities of communication languages.

Keywords: algorithms, automata, error detection, implementation, languages, transducer functionality, transducers

1 Introduction

The abstract above serves very well as a summary of the theme and the contributions of this paper, which is structured as follows. In the next section, we present the basic notation and terminology about languages, channels, error-detection, as well as representations of these concepts via finite automata and transducers. In Section 3, we consider a special class of real-time transducers, called nondeterministic sequential transducers, and we argue that these machines are adequate models for combinatorial channels. In Section 4, we introduce pseudo-sequential transducers as a technical intermediate tool to answer questions about the error-detecting capabilities of languages with respect to channels that are modeled via sequential transducers. In Section 5, we present concrete algorithmic tools, involving sequential and pseudo-sequential transducers, that lead to a concrete algorithm for deciding the property of error-detection. The algorithm is termed concrete as it has led to an open implementation in C++, which is discussed in Section 6.

2 Basic notions and notation

We begin this section with notation and concepts on words, languages and binary relations, and then on finite automata and transducers. We use [10] as a general reference, as well as [2, 12, 11] for automata and transducers.

We denote with $|S|$ the cardinality of a set S. We use iff as a shorthand for "if and only if."

2.1 Word, Language, Binary relation, Channel, Error-detection

We use the standard language theory concepts of alphabet Σ, word (or string), language, etc. The empty word is denoted with λ and the length of a word w is denoted with $|w|$. The set of all words is denoted with Σ^*. A binary relation ρ (over the alphabet Σ) is a subset of $\Sigma^* \times \Sigma^*$. The

[1]Research supported by NSERC.

domain, $\mathrm{dom}(\rho)$, of ρ is the set of all words x with $(x, y) \in \rho$, for some word y. The relation is called functional if $(x, y_1), (x, y_2) \in \rho$ implies that $y_1 = y_2$. A (combinatorial) channel γ is a binary relation that is domain preserving, that is, $(x, x) \in \gamma$, for all $x \in \mathrm{dom}(\gamma)$. The meaning of $(x, y) \in \gamma$ is that the message x can be received as y via the channel. Moreover, if $y \neq x$, we say that x is received with errors, or that y contains errors. A language L is error-detecting for a channel γ if no word $x \in L_\lambda$ can be received with errors as $y \in L_\lambda$ via the channel γ; that is, $(x, y) \in \gamma$ and $x, y \in L_\lambda$ imply that $x = y$. We have used the notation $L_\lambda = L \cup \{\lambda\}$.

2.2 Automaton and transducer concepts

A nondeterministic finite automaton with empty transitions, λ-NFA for short, is a quintuple $\hat{\mathbf{a}} = (Q, \Sigma, T, s, F)$ such that Q is the set of states, Σ is the alphabet, $s \in Q$ is the start (or initial) state, $F \subseteq Q$ is the set of final states, and $T \subseteq Q \times (\Sigma \cup \{\lambda\}) \times Q$ is the finite set of transitions. Let (p, x, q) be a transition of $\hat{\mathbf{a}}$. Then x is called the label of the transition, and we say that p has an outgoing transition (with label x), and q has an incoming transition (with label x). A path of $\hat{\mathbf{a}}$ is a finite sequence $(p_0, x_1, p_1, \ldots, x_\ell, p_\ell)$, for some nonnegative integer ℓ, such that each triple (p_{i-1}, x_i, p_i) is a transition of $\hat{\mathbf{a}}$. The word $x_1 \cdots x_\ell$ is called the label of the path. The path is called accepting if p_0 is the start state and p_ℓ is a final state. The language accepted by $\hat{\mathbf{a}}$, denoted as $L(\hat{\mathbf{a}})$, is the set of labels of all the accepting paths of $\hat{\mathbf{a}}$. The λ-NFA $\hat{\mathbf{a}}$ is called trim, if every state appears in some accepting path of $\hat{\mathbf{a}}$. The λ-NFA $\hat{\mathbf{a}}$ is called an NFA, if no transition label is empty, that is, $T \subseteq Q \times \Sigma \times Q$. A deterministic finite automaton, DFA for short, is a special type of NFA where there is no state p having two outgoing transitions with different labels. Given any λ-NFA $\hat{\mathbf{a}}$, we denote by $\hat{\mathbf{a}}^\lambda$ the λ-NFA that is obtained from $\hat{\mathbf{a}}$ if we add the transitions (p, λ, p), for all states p. Obviously,

$$L(\hat{\mathbf{a}}^\lambda) = L(\hat{\mathbf{a}}).$$

A (finite) transducer is a sextuple $\hat{\mathbf{t}} = (Q, \Sigma, \Delta, T, s, F)$ such that Q, s, F are exactly the same as those in λ-NFAs, Σ is now called the input alphabet, Δ is the output alphabet, and $T \subseteq Q \times \Sigma^* \times \Delta^* \times Q$ is the finite set of transitions. We write $(p, x/y, q)$ for a transition – the label here is (x/y), with x being the input and y being the output label. The concepts of path, accepting path, and trim transducer are similar to those in λ-NFAs. In particular the label of a path $(p_0, x_1/y_1, p_1, \ldots, x_\ell/y_\ell, p_\ell)$ is the pair $(x_1 \cdots x_\ell, y_1 \cdots y_\ell)$ consisting of the input and output labels in the path. The relation realized by the transducer $\hat{\mathbf{t}}$, denoted as $R(\hat{\mathbf{t}})$, is the set of labels in all the accepting paths of $\hat{\mathbf{t}}$. The transducer $\hat{\mathbf{t}}$ is said to be in standard form, if each transition $(p, x/y, q)$ is such that $x \in (\Sigma \cup \{\lambda\})$ and $y \in (\Delta \cup \{\lambda\})$. We note that every transducer is effectively equivalent (realizing the same relation, that is) to one in standard form. A transition of the form $(p, \lambda/y, q)$ is called a λ-input transition. We write $\hat{\mathbf{t}}(x)$ for the set of possible outputs of $\hat{\mathbf{t}}$ on input x, that is, $y \in \hat{\mathbf{t}}(x)$ iff $(x, y) \in R(\hat{\mathbf{t}})$. A functional transducer is a transducer realizing a functional relation, and a channel transducer $\hat{\mathbf{c}}$ is a transducer realizing a domain preserving relation.

Size of machines. For any finite state machine $\hat{\mathbf{m}}$ we assume that the set of states is of the form $\{0, 1, \ldots, n-1\}$, for some integer $n > 0$. The alphabet is also seen as a set of this form. For all practical purposes, any such object (state or alphabet symbol) can be stored in a register of a modern computer and can be processed with unit cost. This implies that we can construct arrays, indexed by states, whose entries maintain information about the states and, therefore, accessing the entry of a randomly chosen state takes constant time. The size $|\overrightarrow{t}|$ of a transition $\overrightarrow{t} = (p, \beta, q)$ is $1 + |\beta|$ if the machine is a λ-NFA, or $1 + |x| + |y|$ if it is a transducer and $\beta = x/y$. The size $|\hat{\mathbf{m}}|$

of a machine $\hat{\mathbf{m}}$ is the number of states plus the total size of the transitions:

$$|\hat{\mathbf{m}}| = |Q_{\hat{\mathbf{m}}}| + \sum_{\overrightarrow{t} \in T_{\hat{\mathbf{m}}}} |\overrightarrow{t}|.$$

3 Nondeterministic sequential transducers realizing channels

Arbitrary transducers in standard form are quite useful in realizing combinatorial channels—see, for example, Fig. 1. When it comes to deciding properties of languages with respect to channels,

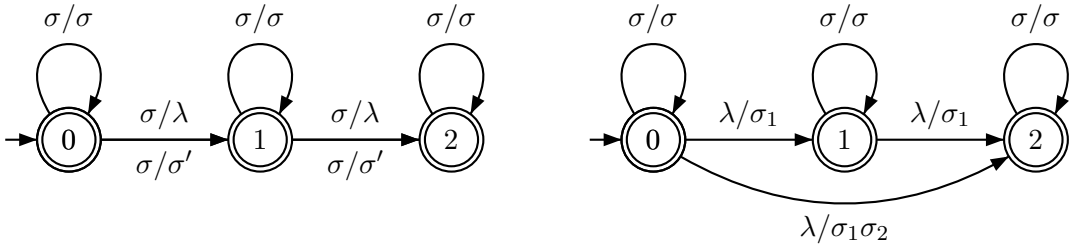

Figure 1: The first channel transducer allows at most two symbol substitutions/deletions in any input word. The second channel transducer allows at most two insertions of symbols in any input word. An arrow with label 'σ/λ' indicates multiple transitions between the states involved, one for each letter σ of the alphabet. A label 'σ/σ'' indicates multiple transitions, for all *different* symbols σ and σ'. Labels with σ_1, σ_2 again indicate multiple transitions, for all symbols σ_1, σ_2 (possibly equal). These notational conventions also apply to the rest of the figures.

the problem of testing the functionality of a transducer is instrumental. However, this test works efficiently when the transducer in question is real-time [11] and in particular when each transition $(p, \sigma/e, q)$ is such that e is a single word. For this reason we provide the following definition.

Definition 1 *A transducer is called (nondeterministic) sequential if every transition label is of the form σ/w, where σ is a single symbol and w is a word.*

By definition, if the input to a sequential transducer $\hat{\mathbf{s}}$ is λ, then there is an output iff the start state is also a final sate and the output is exactly λ. Observe that the first channel transducer in Fig. 1 is sequential, but the second one is not. However, it turns out that the second one is equivalent to the first sequential transducer in Fig. 2, except for any pairs (λ, y), where y is nonempty. In fact, in [3] it is shown that many types of channels, including all ordinary SID channels [7, 9], can be realized using sequential transducers, again except for the pairs (λ, y) mentioned before. This limitation is negligible for most practical purposes.

4 Pseudo-sequential transducers

While sequential transducers are convenient for certain needs (as discussed in the previous section), they are not in standard form, as they could include transitions having labels σ/w with $|w| \geq 2$. This creates a complication when we need to (a) intersect the output language of a sequential transducer $\hat{\mathbf{s}}$ with a regular language, given via an automaton $\hat{\mathbf{a}}$, and then (b) test the functionality

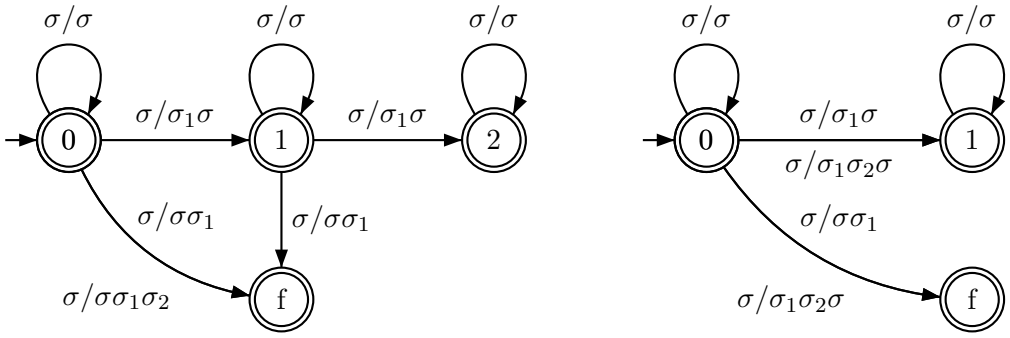

Figure 2: The first transducer shown here is sequential and equivalent to the second transducer in Fig. 1 minus any channel pairs where the input word is empty and the output word is nonempty. State f has no outgoing transitions, which is necessary to realize insertion errors at the end of the input word. The second transducer shown here is again sequential and realizes the channel that allows a burst of up to two consecutive insertions in any input word.

of the resulting transducer. This sequence of tasks can be accomplished if we somehow convert $\hat{\mathbf{s}}$ to a certain type of transducer $\hat{\mathbf{p}}$ in standard form, so as to intersect its outputs with $\hat{\mathbf{a}}$, and then convert the resulting transducer back to sequential form in order to test it for functionality.

Definition 2 *A (nondeterministic) pseudo-sequential transducer is a trim transducer in standard form with the following restrictions on the λ-input transitions.*

- *The start and final states have no outgoing λ-input transitions.*
- *If a state has an outgoing λ-input transition then that state has no other outgoing transitions.*

Lemma 1 *Let $\hat{\mathbf{p}}$ be a pseudo-sequential transducer.*

1. *$\hat{\mathbf{p}}$ contains no cycle consisting entirely of λ-input transitions.*
2. *Let $(p, \lambda/y, q)$ be a λ-input transition in $\hat{\mathbf{p}}$. There exists a unique state r and a unique path of λ-input transitions from q to r such that r has no outgoing λ-transitions—obviously, the unique path is empty if already state q has no λ-input transitions.*

Proof. For the first statement, assume that $\hat{\mathbf{p}}$ contains a cycle of λ-input transitions. By the second condition in Definition 2, these transitions are the only outgoing transitions of the states involved in the cycle and, as $\hat{\mathbf{p}}$ is trim, one of the states in the cycle must be final, which contradicts the fact that the final states of $\hat{\mathbf{p}}$ have no outgoing λ-input transitions. The second statement follows from the facts that state p must reach a final state (as $\hat{\mathbf{p}}$ is trim), and no final state of $\hat{\mathbf{p}}$ has any outgoing λ-input transitions. ■

The next result is important because the resulting machine $\hat{\mathbf{p}}$ is in standard form and can be intersected on the output with an NFA—see Section 5.

Lemma 2 *There is a linear time algorithm that converts any sequential transducer $\hat{\mathbf{s}}$ to an equivalent pseudo-sequential transducer $\hat{\mathbf{p}}$ such that $|\hat{\mathbf{p}}| = O(|\hat{\mathbf{s}}|)$.*

Proof. The required transducer $\hat{\mathbf{p}}$ is obtained if we first make $\hat{\mathbf{s}}$ trim and then, for each transition $(p, \sigma/\sigma_1 \cdots \sigma_\ell, q)$ of $\hat{\mathbf{s}}$, with each $\sigma_i \in \Sigma$, we either add that transition to the set of transitions of $\hat{\mathbf{p}}$ if $\ell < 2$, or we add to $\hat{\mathbf{p}}$ the transitions $(p, \sigma/\sigma_1, q_1), (q_1, \lambda/\sigma_2, q_2), \ldots, (q_{\ell-1}, \lambda/\sigma_\ell, q_\ell)$ if $\ell \geq 2$, where $q_\ell = q$ and the other q_i's are new states. This transition expansion is a folklore construction used to convert a transducer to one in standard form—see, e.g., [2, 12]. The start and final states of $\hat{\mathbf{p}}$ are exactly those of $\hat{\mathbf{s}}$. ■

Now we turn to the converse of the above lemma. Consider a pseudo-sequential transducer $\hat{\mathbf{p}}$ with some set of states $Q_{\hat{\mathbf{p}}}$, and let

$$Q_{\hat{\mathbf{p}}}^{\text{root}} = \{r \in Q_{\hat{\mathbf{p}}} \mid r \text{ has incoming } \lambda\text{-input transitions, but no outgoing } \lambda\text{-input transitions}\}$$

By Lemma 1, for each transition $(p, \lambda/y, q)$, there is a unique λ-input path from p to a unique $r \in Q_{\hat{\mathbf{p}}}^{\text{root}}$. If we create the reverses of all the λ-input transitions heading to r, we get a labeled tree $\hat{\mathbf{e}}_r$ whose root is r. Moreover, for any other $r' \in Q_{\hat{\mathbf{p}}}^{\text{root}} - \{r\}$, the trees $\hat{\mathbf{e}}_r$ and $\hat{\mathbf{e}}_{r'}$ share no vertices (as $\hat{\mathbf{p}}$ is pseudo-sequential)—see the example in the next figure.

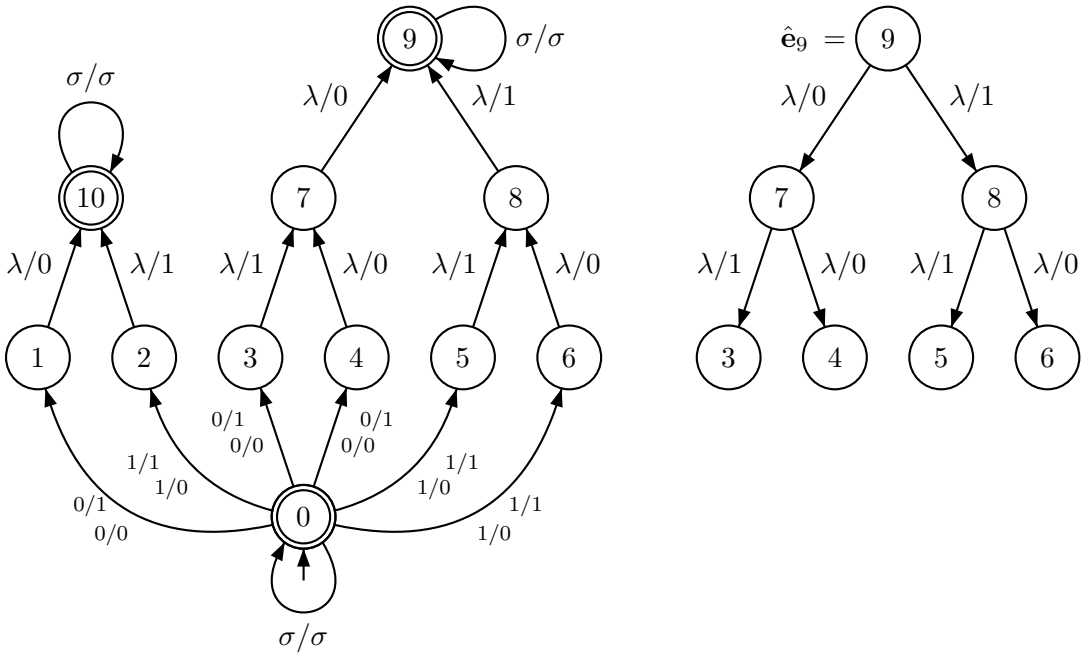

Figure 3: A pseudo-sequential transducer $\hat{\mathbf{p}}$, with $Q_{\hat{\mathbf{p}}}^{\text{root}} = \{9, 10\}$, and the tree $\hat{\mathbf{e}}_9$. The input and output alphabets are $\{0, 1\}$.

A simple algorithm converting $\hat{\mathbf{p}}$ to an equivalent sequential transducer $\hat{\mathbf{s}}$ is the following

(1) Let $\hat{\mathbf{s}}$ be a copy of $\hat{\mathbf{p}}$.

(2) Let X be the set of all transitions $(p, x/y, q)$ in $\hat{\mathbf{s}}$ such that $x \neq \lambda$ and there is a λ-input transition $(q, \lambda/z_1, r_1)$.

(3) Each transition $(p, x/y, q)$ in X is replaced with $(p, x/yz_1 \cdots z_k, r)$ such that r is the unique state in $Q_{\hat{\mathbf{p}}}^{\text{root}}$ that is reachable from q, and $z_1, \ldots, z_k$ are the words appearing in the unique λ-input path from q to r—see Fig. 4.

(4) Delete all the λ-input transitions.

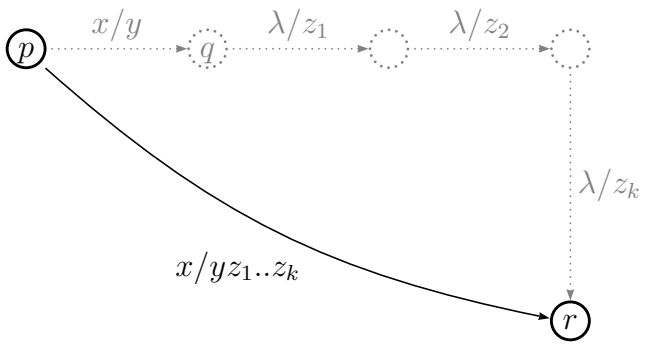

Figure 4: Converting from pseudo-sequential to sequential via the simple algorithm.

Step (3) of the above algorithm can be improved if we make use of the trees $\hat{\mathbf{e}}_r$, so that λ-input transitions are not visited multiple times. The improved algorithm is presented in the next lemma.

Lemma 3 *There is a linear time algorithm that converts any pseudo-sequential transducer* $\hat{\mathbf{p}}$ *to an equivalent sequential transducer* $\hat{\mathbf{s}}$ *such that* $|\hat{\mathbf{s}}| = O(|\hat{\mathbf{p}}|)$.

Proof. Initially $\hat{\mathbf{s}}$ is equal to an exact copy of $\hat{\mathbf{p}}$ having the same start and final states. Then, the algorithm computes in linear time the set $Q_{\hat{\mathbf{s}}}^{\text{root}}$ by scanning each transition of $\hat{\mathbf{s}}$ and recording, for each encountered state q, its outgoing and incoming transitions. Then, using the recorded data, the algorithm creates each tree $\hat{\mathbf{e}}_r$, with $r \in Q_{\hat{\mathbf{s}}}^{\text{root}}$, and iterates for each level i (from level 1 to the leaves) of $\hat{\mathbf{e}}_r$, where it deletes the vertices of level i and makes new edges from the root r to every vertex p of level $i+1$ such that the label of a new edge results by concatenating the labels of the two edges leading from r to p. When a vertex of $\hat{\mathbf{e}}_r$ is deleted, then also the corresponding state of $\hat{\mathbf{s}}$ is deleted, and when a new edge is made, then also the reverse of that edge is added as a transition of $\hat{\mathbf{s}}$. More details are given below—see also the next figure.

For each $r \in Q_{\hat{\mathbf{s}}}^{\text{root}}$:

(1) Compute the tree $\hat{\mathbf{e}}_r$

(2) Let R be the set of children of r; if R is empty, then exit

(3) For each edge $(r, \lambda/z, q)$ of $\hat{\mathbf{e}}_r$, with $q \in R$, and for each transition $(p, x/y, q)$ of $\hat{\mathbf{s}}$

  (4) delete q from $\hat{\mathbf{e}}_r$ and $\hat{\mathbf{s}}$, and add in $\hat{\mathbf{s}}$ the transition $(p, x/yz, r)$

  (5) if $x = \lambda$, then add in $\hat{\mathbf{e}}_r$ the edge $(r, \lambda/yz, p)$

(6) goto (2)

Note that, in Step (3), if q is a leaf node then there is no edge of the form $(q, \lambda/y, p)$ in $\hat{\mathbf{e}}_r$. On the other hand, there is always a transition of $\hat{\mathbf{s}}$ going into q, as $(q, \lambda/z, r)$ is in $\hat{\mathbf{s}}$ and q is reachable from the start state of $\hat{\mathbf{s}}$.

For each tree, the algorithm operates on the tree and the corresponding subgraph of $\hat{\mathbf{s}}$ in linear time. Moreover, as the trees are disjoint, the overall complexity of the algorithm is linear with respect to the original input $\hat{\mathbf{p}}$. ■

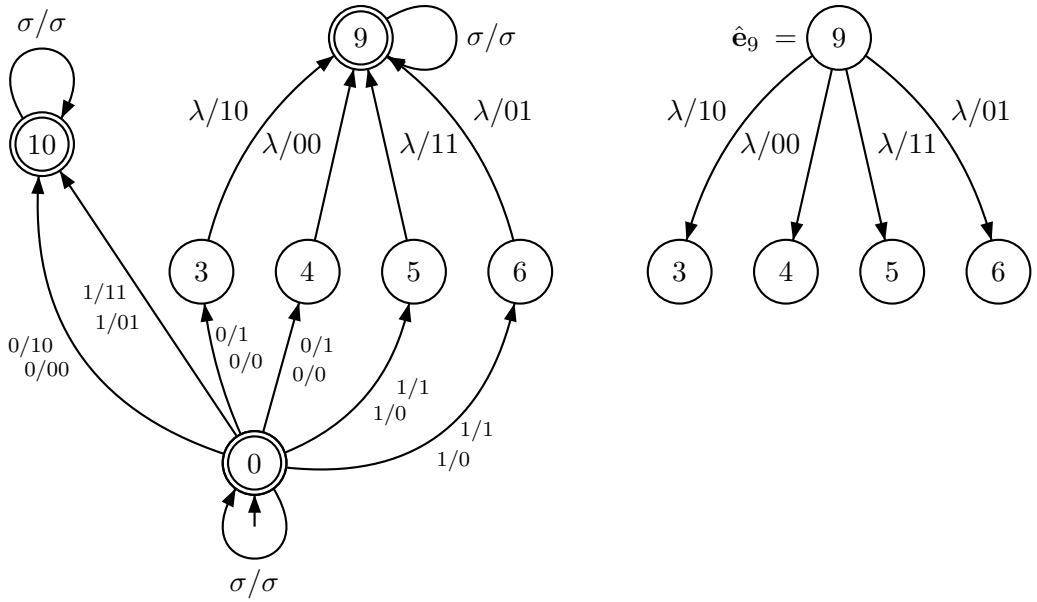

Figure 5: The transducer and the tree $\hat{\mathbf{e}}_9$ of Fig. 3 after eliminating one level of nodes as described in the algorithm in Lemma 3.

5 Error-detection: concrete algorithmic tools

In [8], the author characterizes error-detecting languages in terms of the operations $\downarrow$ and $\uparrow$, which are defined between a binary relation ρ and a language K as follows:

$$\rho \downarrow K = \rho \cap (K \times \Sigma^*), \quad \rho \uparrow K = \rho \cap (\Sigma^* \times K). \tag{1}$$

The characterization is as follows.

Theorem 1 [8] *A language L is error-detecting for a channel γ if and only if the binary relation $(\gamma \downarrow L_\lambda) \uparrow L_\lambda$ is functional, where $L_\lambda = L \cup \{\lambda\}$*

In [8] it is also shown via simple product constructions that, when ρ and K are effectively given via a transducer $\hat{\mathbf{c}}$ *in standard form* and a λ-NFA $\hat{\mathbf{a}}$, respectively, then we can construct trim transducers $(\hat{\mathbf{c}} \downarrow \hat{\mathbf{a}})$ and $(\hat{\mathbf{c}} \uparrow \hat{\mathbf{a}})$ realizing the relations in (1) such that the constructions operate in time $O(|\hat{\mathbf{c}}||\hat{\mathbf{a}}|)$ and the constructed transducers are of size $O(|\hat{\mathbf{c}}||\hat{\mathbf{a}}|)$. When $\hat{\mathbf{c}}$ is a channel transducer then we can decide whether $L(\hat{\mathbf{a}})$ is error-detecting for the channel $R(\hat{\mathbf{c}})$ by testing whether the transducer $(\hat{\mathbf{c}} \downarrow \hat{\mathbf{a}}_0) \uparrow \hat{\mathbf{a}}_0$ is functional, where $\hat{\mathbf{a}}_0$ is a λ-NFA accepting $L(\hat{\mathbf{a}}) \cup \{\lambda\}$. Transducer functionality is shown to be decidable in [6, 1, 11]. In particular, [1, 11] provide an algorithm that works for real-time transducers that are essentially nondeterministic sequential transducers[2]. As stated in [11], for any such transducer $\hat{\mathbf{s}}$, the algorithm works in time $O(|\hat{\mathbf{s}}|^2)$, assuming $O(1)$ cost in processing words independently of their lengths. With these tools, the tools about pseudo-sequential transducers, and the following lemma, we can state and prove the main result of this section—see Theorem 2 further below. This result is a refinement of Corollary 3 in [8], and gives a

[2] In [1], it is stated that the real-time transducer used as input to the algorithm has transitions $(p, \sigma/e, q)$, where e is a single word.

concrete algorithm for deciding the property of error-detection, which leads to a publicly available implementation—see Section 6. To our knowledge, this is the first open implementation of deciding the error-detection property at the level of generality considered in this paper.

A note on $(\hat{\mathbf{c}} \downarrow \hat{\mathbf{a}})$ **and** $(\hat{\mathbf{c}} \uparrow \hat{\mathbf{a}})$. These transducers are trim and in standard form. They have (q_0, q_0') as start state, where q_0 and q_0' are the start states of $\hat{\mathbf{c}}$ and $\hat{\mathbf{a}}^\lambda$, respectively. They have as final states all pairs (f, f') with f and f' being final states of $\hat{\mathbf{c}}$ and $\hat{\mathbf{a}}^\lambda$, respectively. A tuple $((p_1, p_2), x/y, (q_1, q_2))$ is a transition of $\hat{\mathbf{c}} \downarrow \hat{\mathbf{a}}$ (resp. $\hat{\mathbf{c}} \uparrow \hat{\mathbf{a}}$) iff $(p_1, x/y, p_2)$ is a transition of $\hat{\mathbf{c}}$ and (p_2, x, q_2) (resp. (p_2, y, q_2)) is a transition of $\hat{\mathbf{a}}^\lambda$.

Lemma 4 *If* $\hat{\mathbf{p}}$ *is a pseudo-sequential transducer and* $\hat{\mathbf{a}}$ *is a DFA, then the transducer* $(\hat{\mathbf{p}} \uparrow \hat{\mathbf{a}})$ *is pseudo-sequential.*

Proof. By definition of the operation '$\uparrow$', the transducer $(\hat{\mathbf{p}} \uparrow \hat{\mathbf{a}})$ is trim and in standard form. Now consider any λ-input transition $\overrightarrow{t} = ((p_1, p_2), \lambda/\sigma, (q_1, q_2))$ of $(\hat{\mathbf{p}} \uparrow \hat{\mathbf{a}})$. We have to show that (p_1, p_2) is neither the start state, nor a final state, and that it has no other outgoing transitions. By definition of $(\hat{\mathbf{p}} \uparrow \hat{\mathbf{a}})$, $(p_1, \lambda/\sigma, q_1)$ is a transition of $\hat{\mathbf{p}}$ and (p_2, σ, q_2) is a transition of $\hat{\mathbf{a}}$. Then (p_1, p_2) is neither the start state nor a final state, as otherwise the same would hold for the state p_1 of $\hat{\mathbf{p}}$. By Lemma 1, $p_1 \neq q_1$, as otherwise the transition $(p_1, \lambda/\sigma, q_1)$ itself would form a cycle. Now let $((p_1, p_2), x/y, (q_1', q_2'))$ be any transition in $(\hat{\mathbf{p}} \uparrow \hat{\mathbf{a}})$ going out of state (p_1, p_2). Again, $(p_1, x/y, q_1')$ is a transition of $\hat{\mathbf{p}}$ and (p_2, y, q_2') is a transition of $\hat{\mathbf{a}}^\lambda$. As $\hat{\mathbf{p}}$ is pseudo-sequential and $\hat{\mathbf{a}}$ is deterministic, it follows that $x = \lambda, y = \sigma, q_1' = q_1, q_2' = q_2$. Thus, $\overrightarrow{t}$ is the only transition going out of (p_1, p_2). ■

Theorem 2 *The following problem is decidable in time* $O(|\hat{\mathbf{c}}|^2 |\hat{\mathbf{a}}|^4)$.

Input: sequential channel transducer $\hat{\mathbf{c}}$ *and DFA* $\hat{\mathbf{a}}$.

Return: whether $L(\hat{\mathbf{a}})$ *is error-detecting for the channel* $R(\hat{\mathbf{c}})$.

Proof. We first note that the operation $\hat{\mathbf{c}} \downarrow \hat{\mathbf{a}}$ can be extended easily to the case where $\hat{\mathbf{c}}$ is not necessarily in standard form, but in input-standard form; that is, every transition label is of the form x/w where $x \in \Sigma \cup \{\lambda\}$ and $w \in \Sigma^*$. Indeed in this case, a tuple $((p_1, p_2), x/w, (q_1, q_2))$ is a transition of $\hat{\mathbf{c}} \downarrow \hat{\mathbf{a}}$ iff there are two corresponding transitions: $(p_1, x/w, q_1)$ of $\hat{\mathbf{c}}$ and (p_2, x, q_2) of $\hat{\mathbf{a}}^\lambda$. Let $\hat{\mathbf{a}}_0$ be a DFA accepting the language $L(\hat{\mathbf{a}}) \cup \{\lambda\}$. If the start state of $\hat{\mathbf{a}}$ is a final state, then $\hat{\mathbf{a}}_0 = \hat{\mathbf{a}}$. Otherwise, $\hat{\mathbf{a}}_0$ results from $\hat{\mathbf{a}}$ if we add a new state s', which is the start state of $\hat{\mathbf{a}}_0$ and also one of its final states, and the transitions (s', σ, p), for all transitions (s, σ, p) of $\hat{\mathbf{a}}$ in which s is the start state of $\hat{\mathbf{a}}$. Then, the algorithm is as follows.

1. Construct $\hat{\mathbf{p}} \leftarrow (\hat{\mathbf{c}} \downarrow \hat{\mathbf{a}}_0)$
2. Convert $\hat{\mathbf{p}}$ to pseudo-sequential
3. Construct $\hat{\mathbf{s}} \leftarrow (\hat{\mathbf{p}} \uparrow \hat{\mathbf{a}}_0)$
4. Convert $\hat{\mathbf{s}}$ to sequential
5. if ($\hat{\mathbf{s}}$ is functional) return YES; else return NO

Note that in the first step, transducer $\hat{\mathbf{p}}$ is sequential because $\hat{\mathbf{c}}$ is sequential and each transition label of $\hat{\mathbf{p}}$ is also a transition label of $\hat{\mathbf{c}}$. The second step is necessary so that $\hat{\mathbf{p}}$ is converted in standard form before performing $\hat{\mathbf{p}} \uparrow \hat{\mathbf{a}}_0$.

The correctness and complexity of the algorithm follow from the statements in the previous paragraph, the discussion at the beginning of this section, and Lemmata 4, 2, 3. ■

6 Open implementation

We have implemented a version of the algorithm presented in Theorem 2 and made the implementation accessible via the website

`http://research1.cs.smu.ca/~a_daka/index.php`

which is also accessible via

`http://cs.smu.ca/~stavros/#research`

Implementation details are given in that website as well as in the thesis [3]. The implementation is written in C++ and uses an old version of Grail [5] libraries. We have added classes for transducer objects and methods implementing the various algorithmic tools required for deciding error-detection.

At the time of writing this report, the current implementation converting a pseudo-sequential transducer to an equivalent sequential one does not use the algorithm in Lemma 3, but the simpler algorithm presented prior to that lemma.

The website allows users to enter automata in Grail format and also transducers in Grail-like format. The empty word is represented using the character '~'. Below we show the Grail-like description of the first transducer in Fig 1, where the input and output alphabets are equal to $\{a, b\}$, and the Grail description of a DFA accepting the language of all words of length 3 over $\{a, b\}$ with an even number of b's.

```
(START) |- 0
0 (a,a) 0
0 (b,b) 0
0 (a,~) 1
0 (b,~) 1
0 (a,b) 1
0 (b,a) 1
1 (a,a) 1
1 (b,b) 1
1 (a,~) 2
1 (b,~) 2
1 (a,b) 2
1 (b,a) 2
2 (a,a) 2
2 (b,b) 2
0 -| (FINAL)
1 -| (FINAL)
2 -| (FINAL)
```

```
(START) |- 0
0 a 2
0 b 1
1 a 3
1 b 4
2 a 4
2 b 3
3 b 5
4 a 5
5 -| (FINAL)
```

Acknowledgement

The authors have used the LaTeX package [4] for drawing automata.

References

[1] M.P. Béal, O. Carton, C. Prieur and J. Sakarovitch. Squaring transducers: An efficient procedure for deciding functionality and sequentiality. *Theoret. Computer Science* **292**:1 (2003), 45–63.

[2] J. Berstel. *Transductions and Context-Free Languages*. B.G. Teubner, Stuttgart, 1979.

[3] A. Daka. *Computing Error-detecting Capabilities of Reguar Languages*. MSc Thesis, Mathematics and Computing Science, Saint Mary's University, Canada, 2011

[4] GasTex. `http://www.lsv.ens-cachan.fr/~gastin/gastex/` Accessed on Oct. 25, 2011

[5] Grail+. `http://www.csd.uwo.ca/Research/grail/` Accessed on Sep. 10, 2011

[6] E.M. Gurari and O.H. Ibarra. Finite-valued and finitely ambiguous transducers. *Math. Systems Theory* **16** (1983), 61–66.

[7] L. Kari and S. Konstantinidis. Descriptional complexity of error/edit systems. *Journal of Automata, Languages and Combinatorics* **9**:2–3 (2004), 293–309.

[8] S. Konstantinidis. Transducers and the properties of error-detection, error-correction, and finite-delay decodability. *J. Universal Computer Science* **8**:2 (2002), 278–291.

[9] S. Konstantinidis and P.V. Silva. Computing maximal error-detecting capabilities and distances of regular languages. *Fundamenta Informaticae* **101**:4 (2010), 257–270.

[10] G. Rozenberg and A. Salomaa (eds). *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

[11] J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, Berlin, 2009.

[12] S. Yu. Regular Languages. In [10], Vol. 1, pp 41–110.