On Synthesis of Resynchronizers for Transducers

Sougata Bose
LaBRI, University of Bordeaux, France

Shankara Narayanan Krishna
Department of Computer Science & Engineering IIT Bombay, India

Anca Muscholl
LaBRI, University of Bordeaux, France

Vincent Penelle
LaBRI, University of Bordeaux, France

Gabriele Puppis
CNRS, LaBRI, University of Bordeaux, France

Abstract

We study two formalisms that allow to compare transducers over words under origin semantics: rational and regular resynchronizers, and show that the former are captured by the latter. We then consider some instances of the following synthesis problem: given transducers T_1, T_2, construct a rational (resp. regular) resynchronizer R, if it exists, such that T_1 is contained in $R(T_2)$ under the origin semantics. We show that synthesis of rational resynchronizers is decidable for functional, and even finite-valued, one-way transducers, and undecidable for relational one-way transducers. In the two-way setting, synthesis of regular resynchronizers is shown to be decidable for unambiguous two-way transducers. For larger classes of two-way transducers, the decidability status is open.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases String transducers, resynchronizers, synthesis

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.69

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.08688`.

Funding DeLTA project (ANR-16-CE40-0007)

1 Introduction

The notion of word transformation is pervasive in computer science, as computers typically process streams of data and transform them between different formats. The most basic form of word transformation is realized using finite memory. Such a model is called finite-state transducer and was studied from the early beginnings of automata theory. Differently from automata, the expressiveness of transducers is significantly affected by the presence of non-determinism (even when the associated transformation is a function), and by the capability of processing the input in both directions (one-way vs two-way transducers). Another difference is that many problems, notably, equivalence and containment, become undecidable when moving from automata to transducers [11, 14].

An alternative semantics for transducers, called *origin semantics*, was introduced in [4] in order to obtain canonical two-way word transducers. In the origin semantics, the output is tagged with positions of the input, called origins, that describe where each output element was produced. According to this semantics, two transducers may be non-equivalent even when they compute the same relation in the classical semantics. From a computational viewpoint, the origin semantics has the advantage that it allows to recover the decidability of equivalence and containment of non-deterministic (and even two-way) transducers [6].

It can be argued that comparing two transducers in the origin semantics is rather restrictive, because it requires that the same output is generated at precisely the same place. A natural approach to allow some 'distortion' of the origin information when comparing two transducers was proposed in [10]. *Rational* resynchronizers allow to compare *one-way* transducers (hence, the name 'rational') under origin distortions that are generated with finite control. A rational resynchronizer is simply a one-way transducer that processes an interleaved input-output string, producing another interleaved interleaved input-output string with the same input and output projection. For two-way transducers (or equivalently, streaming string transducers [1]) a different formalism is required to capture origin distortion, since the representation of the origin information through interleaved input-output pairs does not work anymore. To this purpose, *regular resynchronizers* were introduced in [6] as a logic-based transformation of origin graphs, in the spirit of Courcelle's monadic second-order logic definable graph transductions [8]. In [6] it was shown that containment of two-way transducers up to a (bounded) regular resynchronizer is decidable.

In this paper we first show that bounded regular resynchronizers capture the rational ones. This result is rather technical, because rational resynchronizers work on explicit origin graphs, encoded as input-output pairs, which is not the case for regular resynchronizers. Then we consider the following problem: given two transducers T_1, T_2, we ask whether some rational, or bounded regular, resynchronizer R exists such that T_1 is origin-contained in T_2 up to R. So here, the resynchronizer R is not part of the input, and we want to synthesize such a resynchronizer, if one exists.

Our main contributions can be summarized as follows:

1. synthesis of rational resynchronizers for functional (or even finite-valued) one-way transducers is decidable,
2. synthesis of rational resynchronizers for unrestricted one-way transducers is undecidable,
3. synthesis of bounded regular resynchronizers for unambiguous two-way transducers is decidable.

Somewhat surprisingly, for both decidable cases above the existence of a resynchronizer turns out to be equivalent to the classical inclusion of the two transducers.

Full proofs of the results presented in this paper can be found in the extended version `https://arxiv.org/abs/1906.08688`.

2 Preliminaries

One-way transducers

One of the simplest transducer model is the one-way non-deterministic finite-state transducer (hereafter, simply one-way transducer), capturing the class of so-called *rational relations*. This is basically an automaton in which every transition consumes one letter from the input and appends a word of any length to the output.

Formally, a *one-way transducer* is a tuple $T = (\Sigma, \Gamma, Q, I, E, F, L)$, where Σ, Γ are finite input and output alphabets, Q is a finite set of states, $I, F \subseteq Q$ are subsets of initial and final states, $E \subseteq Q \times \Sigma \times Q$ is a finite set of transition rules, and $L : E \uplus F \to 2^{\Gamma^*}$ is a function specifying a regular language of partial outputs for each transition rule and each final state. The relation defined by T contains pairs (u, v) of input and output words, where $u = a_1 \dots a_n$ and $v = v_1 \dots v_n\, v_{n+1}$, for which there is a run $q_0 \xrightarrow{a_1 \mid v_1} q_1 \xrightarrow{a_2 \mid v_2} \dots q_n \xrightarrow{\mid v_{n+1}}$ such that $q_0 \in I$, $q_n \in F$, $(q_{i-1}, a_i, q_i) \in E$, $v_i \in L(q_{i-1}, a_i, q_i)$, and $v_{n+1} \in L(q_n)$. The transducer is called *functional* if it associates at most one output with each input, namely, if it realizes a partial function. For example, Figure 1 shows two one-way transducers with input alphabet

Figure 1 Examples of functional and relational one-way transducers.

$\Sigma = \{a, b\}$ and output alphabet $\Gamma \supseteq \Sigma$. The first transducer is functional, and realizes the cyclic rotation $f : cu \mapsto uc$, for any letter $c \in \{a, b\}$ and any word $u \in \{a, b\}^*$. The second transducer is not functional, and associates with an input $u \in \Sigma^*$ any possible word $v \in \Gamma^*$ as output such that u is a sub-sequence of v.

Two-way transducers

Allowing the input head to move in any direction, to the left or to the right, gives a more powerful model of transducer, which captures e.g. the relation $\{(u, u^n) \ : \ u \in \Sigma^*, n \in \mathbb{N}\}$. To define two-way transducers, we adopt the convention that, for any given input $u \in \Sigma^*$, $u(0) = \vdash$ and $u(|u| + 1) = \dashv$, where $\vdash, \dashv \notin \Sigma$ are special markers used as delimiters of the input. In this way, a transducer can detect when an endpoint of the input has been reached.

A *two-way transducer* is a tuple $T = (\Sigma, \Gamma, Q, I, E, F, L)$, whose components are defined just like those of a one-way transducer, except that the state set Q is partitioned into two subsets, $Q_<$ and $Q_>$, the set I of initial states is contained in $Q_>$, and the set E of transition rules is contained in $(Q \times \Sigma \times Q) \uplus (Q_< \times \{\vdash\} \times Q_>) \uplus (Q_> \times \{\dashv\} \times Q_<)$. The partitioning of the set of states is useful for specifying which letter is read from each state: states from $Q_<$ read the letter to the left, whereas states from $Q_>$ read the letter to the right. Given an input $u \in \Sigma^*$, a configuration of a two-way transducer is a pair (q, i), with $q \in Q$ and $i \in \{1, \ldots, |u| + 1\}$. Based on the types of source and target states in a transition rule, we can distinguish four types of transitions between configurations (the output v is always assumed to range over the language $L(q, a, q')$):

- $(q, i) \xrightarrow{a \mid v} (q', i + 1)$ if $(q, a, q') \in E$, $q, q' \in Q_>$, and $a = u(i)$,
- $(q, i) \xrightarrow{a \mid v} (q', i)$ if $(q, a, q') \in E$, $q \in Q_>$, $q' \in Q_<$, and $a = u(i)$,
- $(q, i) \xrightarrow{a \mid v} (q', i - 1)$ if $(q, a, q') \in E$, $q, q' \in Q_<$, and $a = u(i - 1)$,
- $(q, i) \xrightarrow{a \mid v} (q', i)$ if $(q, a, q') \in E$, $q \in Q_<$, $q' \in Q_>$, and $a = u(i - 1)$.

Note that, when reading a marker $\vdash$ or $\dashv$, the transducer is obliged to make a U-turn, either left-to-right or right-to-left. The notions of successful run, realized relation, and functional transducer are naturally generalized from the one-way to the two-way variant, (we refer to [6] for more details).

In [5], a slight extension of two-way transducers, called two-way transducers *with common guess*, was proposed. Before processing its input, such a transducer can non-deterministically guess some arbitrary annotation of the input over a fixed alphabet. Once an annotation is guessed, it remains the same during the computation. Transitions may then depend on the input letter and the guessed annotation at the current position. For example, this extension allows to define relations of the form $\{(u, vv) \ \mid \ u \in \Sigma^*, v \in \Gamma^*, |u| = |v|\}$. Note that the extension with common guess does not increase the expressiveness of one-way transducers, since these are naturally closed under input projections. Likewise, common guess does not affect the expressive power of functional two-way transducers, since one can guess a canonical annotation at runtime.

Figure 2 Input-output pairs annotated with origin information.

Classical vs origin semantics

In the previous definitions, we associated a classical semantics to transducers (one-way or two-way), which gives rise to relations or functions between input words over Σ and output words over Γ. In [4] an alternative semantics for transducers, called *origin semantics*, was introduced with the goal of getting canonical transducers for any given word function. Roughly speaking, in the origin semantics, every position of the output word is annotated with the position of the input where that particular output element was produced. This yields a bipartite graph, called *origin graph*, with two linearly ordered sets of nodes, representing respectively the input and the output elements, and edges directed from output nodes to input nodes, representing the so-called *origins*. Figure 2 depicts an input-output pair (a^n, b^n) annotated with two different origins: in the first graph, a position i in the output has its origin at the same position i in the input, while in the second graph it has origin at position $n - i$.

Formally, the origin semantics of a transducer is a relation $S_o \subseteq \Sigma^* \times (\Gamma \times \mathbb{N})^*$ consisting of pairs (u, ν), where $u = a_1 \dots a_n \in \Sigma^*$ is a possible input and $\nu = \nu_1 \dots \nu_{m+1} \in (\Gamma \times \mathbb{N})^*$ is the corresponding output tagged with input positions, as induced by a successful run of the form $(q_0, i_0) \xrightarrow{a_1 \mid \nu_1} (q_1, i_1) \xrightarrow{a_2 \mid \nu_2} \dots (q_m, i_m) \xrightarrow{\mid \nu_{m+1}}$, with each $\nu_j \in (\Gamma \times \{i_j\})^*$. We identify a pair (u, ν) with the origin graph obtained by arranging the input elements and the output elements along two lines (we omit the successor relation in the graph notation), and adding edges from every output element (a, i) to the i-th element of the input. Given an origin graph $G = (u, \nu)$, we denote by $\mathsf{in}(G)$, $\mathsf{out}(G)$, and $\mathsf{orig}(G)$ respectively the input word u, the output word obtained by projecting ν onto the finite alphabet Γ, and the sequence of input positions (origins) obtained by projecting ν onto $\mathbb{N}$.

For one-way transducers, there is a simpler presentation of origin graphs in the form of interleaved words. Assuming that the alphabets Σ and Γ are disjoint, we interleave the input and output word by appending after each input symbol the output word produced by reading that symbol. For example, if $\Sigma = \{a\}$ and $\Gamma = \{b\}$, then a word of the form $abb \dots abb$ represents an origin graph (a^n, ν), where $|\nu| = 2n$ and $\nu(2i - 1) = \nu(2i) = (b, i)$, for all $i = 1, \dots, n$. Words over $\Sigma \uplus \Gamma$ are called *synchronized words*. Just as every synchronized word represents an origin graph, a regular language over $\Sigma \uplus \Gamma$ represents a rational relation with origins, or equally the origin semantics of a one-way transducer.

In general, when comparing transducers, we can refer to one of the two possible semantics. Clearly, two transducers that are equivalent in the origin semantics are also equivalent in the classical semantics, but the converse is not true.

3 Resynchronizations

The central concept of this paper is that of resynchronization, which is a transformation of origin graphs that preserves the underlying input and output words. The concept was originally introduced in [10], and mostly studied in the setting of rational relations. Here we use the concept in the more general setting of relations definable by two-way transducers.

Formally, a *resynchronization* is any relation $R \subseteq (\Sigma^* \times (\Gamma \times \mathbb{N})^*)^2$ that contains only pairs (G, G') of origin graphs such that $\mathsf{in}(G) = \mathsf{in}(G')$ and $\mathsf{out}(G) = \mathsf{out}(G')$, namely, with the same projections onto the input and output alphabets.[1] A resynchronization R can be used to modify the origin information of a relation, while preserving the underlying input-output pairs. Formally, for every relation $S_o \subseteq \Sigma^* \times (\Gamma \times \mathbb{N})^*$ with origins, we define the *resynchronized relation* $R(S_o) = \{G' \in S_o \mid (G, G') \in R,\ G \in S_o\}$. Note that if the origin information is removed from both $R(S_o)$ and S_o, then $R(S_o) \subseteq S_o$. Moreover, $R(S_o) = S_o$ when R is the *universal resynchronization*, that is, when R contains all pairs (G, G'), with $G, G' \in \Sigma^* \times (\Gamma \times \mathbb{N})^*$, $\mathsf{in}(G) = \mathsf{in}(G')$, and $\mathsf{out}(G) = \mathsf{out}(G')$.

Definability of resynchronized relations

An important property that we need to guarantee in order to enable some effective reasoning on resynchronizations is the definability of the resynchronized relations. More precisely, given a class $\mathcal{C}$ of transducers, we say that a resynchronization R *preserves definability in* $\mathcal{C}$ if for every transducer $T \in \mathcal{C}$, the relation $R(T)$ is realized by some transducer $T' \in \mathcal{C}$, that can be effectively constructed from R and T. The class $\mathcal{C}$ will usually be the class of one-way transducers or the class of two-way transducers, and this will be clear from the context.

Below, we recall the definitions of two important classes of resynchronizations, called rational [10] and regular resynchronizers [6], that preserve definability by one-way transducers and by two-way transducers, respectively. We will then compare the expressive power of these two formalisms, showing that rational resynchronizers are strictly less expressive than regular resynchronizers.

Rational resynchronizers

A natural definition of resynchronizers for one-way transducers is obtained from rational relations over the disjoint union $\Sigma \uplus \Gamma$ of the input and output alphabets. Any such relation consists of pairs of synchronized words (w, w'), and thus represents a transformation of origin graphs. In addition, if the induced synchronized words w and w' have the same projections over the input and output alphabets, then the relation represents a resynchronization. We also recall that rational relations are captured by one-way transducers, so, by analogy, we call *rational resynchronizer* any one-way transducer over $\Sigma \uplus \Gamma$ that preserves the input and output projections.

It is routine to see that rational resynchronizers preserve definability of relations by one-way transducers. It is also worth noting that every rational resynchronizer is a length-preserving transducer. By a classical result of Elgot and Mezei [9] every rational resynchronizer can be assumed to be a *letter-to-letter* one-way transducer, namely, a transducer with transitions of the form $q \xrightarrow{a \mid b} q'$, with $a, b \in \Sigma \uplus \Gamma$.

▶ **Example 1.** Consider the functional one-way transducers T_1, T_2 in Figure 3. The domain of both transducers is $(aa)^*$. An origin graph of T_1 is a one-to-one mapping from the output to the input (each a produces one b). On the other hand, in an origin graph of T_2, every a at input position $2i+1$ is the origin of two b's at output positions $2i+1, 2i+2$. The transducer R depicted to the right of the figure transforms synchronized words while preserving their input and output projections. It is then a rational resynchronizer. In particular, R transforms origin graphs of T_1 to origin graphs of T_2.

[1] In [10], resynchronizers were further restricted to contain at least the pairs of identical origin graphs. Here we prefer to avoid this additional restriction and reason with a more general class of resynchronizations.

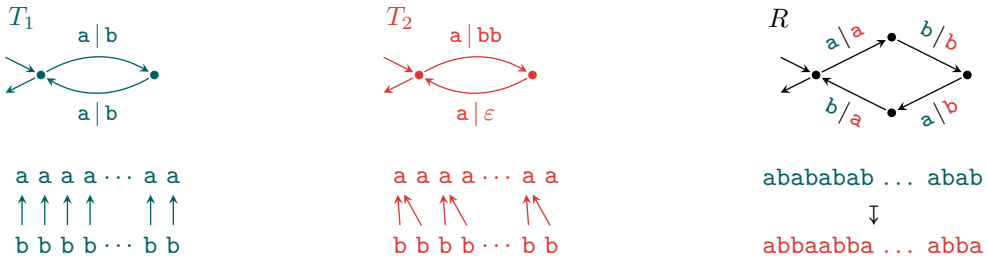

Figure 3 Two functional 1NFT T_1, T_2, their origin graphs, and a rational resynchronizer R.

Regular resynchronizers

While languages of synchronized words are a faithful representation of rational relations, this notation does not capture regular relations, so relations realized by two-way transducers. An alternative formalism for resynchronizations of relations defined by two-way transducers was proposed in [6] under the name of MSO resynchronizer (here we call it simply "resynchronizer"). The formalism describes pairs (G, G') of origin graphs by means of two relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ $(\gamma, \gamma' \in \Gamma)$ in the spirit of MSO graph transductions. More precisely:

- move_γ describes how the origin y of an output position x labeled by γ is redirected to a new origin z (for short, we call y and z the *source* and *target* origins of x). Formally, move_γ is a relation contained in $\Sigma^* \times \mathbb{N} \times \mathbb{N}$ that induces resynchronization pairs (G, G') such that, for all output positions x, if $\mathsf{out}(G)(x) = \gamma$, $\mathsf{orig}(G)(x) = y$, and $\mathsf{orig}(G')(x) = z$, then $(\mathsf{in}(G), y, z) \in \mathsf{move}_\gamma$.
- $\mathsf{next}_{\gamma,\gamma'}$ constrains the target origins z and z' of any two consecutive output positions x and $x+1$ that are labelled by γ and γ', respectively. Formally, $\mathsf{next}_{\gamma,\gamma'}$ is a relation contained in $\Sigma^* \times \mathbb{N} \times \mathbb{N}$ that induces resynchronization pairs (G, G') such that, for all output positions x and $x+1$, if $\mathsf{out}(G)(x) = \gamma$, $\mathsf{out}(G)(x+1) = \gamma'$, $\mathsf{orig}(G')(x) = z$, and $\mathsf{orig}(G')(x+1) = z'$, then $(\mathsf{in}(G), z, z') \in \mathsf{next}_{\gamma,\gamma'}$.

A *resynchronizer* is a tuple $\big((\mathsf{move}_\gamma)_{\gamma\in\Gamma}, (\mathsf{next}_{\gamma,\gamma'})_{\gamma,\gamma'\in\Gamma}\big)$, and defines the resynchronization R with pairs (G, G') induced by the relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$, where $\gamma, \gamma' \in \Gamma$.

In order to obtain a well-behaved class of resynchronizations, that in particular preserves definability by two-way transducers, we need to enforce some restrictions. First, we require that the relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ are described by regular languages (or equally, definable in monadic second-order logic). By this we mean that we encode the input positions y, z, z' with suitable annotations over the binary alphabet $\mathbb{B} = \{0, 1\}$, so that we can identify the relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ with some *regular* languages over the expanded alphabet $\Sigma \times \mathbb{B}^2$. We call *regular resynchronizer* a resynchronizer where the relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ are given by regular languages. In addition, we also require that regular resynchronizers are *k-bounded*, for some $k \in \mathbb{N}$, in the sense that for every input u, every output letter γ, and every target origin z, there are at most k positions y such that $(u, y, z) \in \mathsf{move}_\gamma$.

▶ **Example 2.** Consider the resynchronization R of Figure 4, containing the pairs (G, G'), where G (resp. G') is the origin graph that maps every output position to the first (resp. last) input position. R is "one-way", in the sense that it contains only origin graphs that are admissible outcomes of runs of one-way transducers. However, R is not definable by any rational resynchronizer, since, in terms of synchronized words, it should map $a\,v\,u$ to $a\,u\,v$, for every $a \in \Sigma$, $u \in \Sigma^*$, and $v \in \Gamma^*$, which is clearly not a rational relation. The resynchronization R can however be defined by a 1-bounded regular resynchronizer, e.g. $\big((\mathsf{move}_\gamma)_{\gamma\in\Gamma}, (\mathsf{next}_{\gamma,\gamma'})_{\gamma,\gamma'\in\Gamma}\big)$, with $\mathsf{move}_\gamma = \{(u, y, z) \mid u \in \Sigma^*, y = 1, z = |u|\}$ and

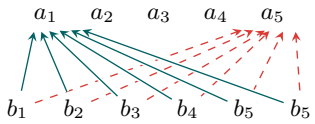

Figure 4 A 1-bounded, regular resynchronization that is not rational.

$\mathsf{next}_{\gamma,\gamma'} = \Sigma^* \times \mathbb{N} \times \mathbb{N}$.

One can observe that, in the previous example, next is not restricting the resynchronization further. For other examples that use next in a non-trivial way see for instance [6, Example 13].

The notion of resynchronizer can be slightly enhanced in order to allow some additional amount of non-determinism in the way origin graphs are transformed (this enhanced notion is indeed the one proposed in [6]). The principle is very similar to the idea of enhancing two-way transducers with common guess. More precisely, we allow additional monadic parameters that annotate the input and the output, thus obtaining words over expanded alphabets of the form $\Sigma \times \Sigma'$ and $\Gamma \times \Gamma'$. A resynchronizer *with parameters* is thus a tuple $\big(\mathsf{ipar}, \mathsf{opar}, (\mathsf{move}_\gamma)_\gamma, (\mathsf{next}_{\gamma,\gamma'})_{\gamma,\gamma'}\big)$, where $\mathsf{ipar} \subseteq (\Sigma \times \Sigma')^*$ describes the possible annotations of the input, $\mathsf{opar} \subseteq (\Gamma \times \Gamma')^*$ describes the possible annotations of the output, and, for every $\gamma, \gamma' \in \Gamma \times \Gamma'$, $\mathsf{move}_\gamma \subseteq (\Sigma \times \Sigma' \times \mathbb{B}^2)^*$ describes a transformation from source to target origins of γ-labelled output positions, and $\mathsf{next}_{\gamma,\gamma'} \subseteq (\Sigma \times \Sigma' \times \mathbb{B}^2)$ constraints the target origins of consecutive output positions labelled by γ and γ'. The resynchronization pairs (G, G') in this case are induced by $\big((\mathsf{move}_\gamma)_{\gamma\in\Gamma\times\Gamma'}, (\mathsf{next}_{\gamma,\gamma'})_{\gamma,\gamma'\in\Gamma\times\Gamma'}\big)$ and are obtained by projecting the input and output over the original alphabets Σ and Γ, under the assumption that the annotations satisfy ipar and opar. A resynchronizer with parameters is called *regular* if all its relations are regular. A regular resynchronizer is called *bounded* if is k-bounded for some k.

In [6] it was shown that, given a bounded regular resynchronizer R with parameters and a two-way transducer T with common guess, one can construct a two-way transducer T' with common guess such that $T' =_o R(T)$. The notation $T' =_o R(T)$ is used to represent the fact that T' and $R(T)$ define the same relation in the origin semantics.

Unless otherwise stated, hereafter we assume that two-way transducers are enhanced with common guess, and regular resynchronizers are enhanced with parameters.

Rational vs regular resynchronizers

Our first result shows that bounded, regular resynchronizers are more expressive than rational resynchronizers. Consider for instance Example 1: it can be captured by the regular resynchronizer with opar annotating even/odd positions. The resynchronizer shifts the origins of the even positions of the output by one to the left and keeps the origins of the odd positions unchanged. So here move_γ can be described by a regular language. On the other hand, Example 2 shows that there are bounded, regular resynchronizers that cannot be captured by rational resynchronizers.

▶ **Theorem 3.** *For every rational resynchronizer, there is an equivalent* 1*-bounded regular resynchronizer.*

The proof of the above result is rather technical and can be found in the extended version. Here we only provide a rough idea. Consider a rational resynchronizer R, that is, a one-way transducer that transforms synchronized words while preserving the input and output projections. For example, Figure 5 represents a possible pair of synchronized words,

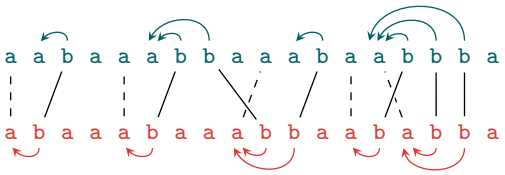

Figure 5 A 1-bounded, regular resynchronization that is not rational.

denoted w and w', shown in blue and in red, respectively, such that $(w, w') \in R$. We assume that $\Sigma = \{a\}$ and $\Gamma = \{b\}$.

From the given rational resynchronizer R we construct an equivalent 1-bounded, regular resynchronizer R'. The natural approach is to encode a successful run ρ of R over a synchronized word w. By measuring the differences between the partial inputs and the partial outputs that are consumed and produced along the run ρ, we obtain a partial bijection on the input letters that represents a mapping from source origins to target origins. This mapping determines the relation move_γ of R', and in fact depends on a suitable additional annotation γ of the underlying output position. The additional annotation is needed in order to distinguish output elements with the same origin in the source, but with different origins in the target.

For example, by referring again to the figure above, consider the first occurrence of b in w. Its origin in w is given by the closest input letter to the left (follow the blue arrow). To find the origin in w', one finds the same occurrence of b in w' (solid line), then moves to the closest input letter to the left (red arrow), and finally maps the latter input position in w' back to w (dashed line). The resulting position determines the new origin (w.r.t. w') of the considered output element.

The remaining components ipar, opar, and $\mathsf{next}_{\gamma,\gamma'}$ of R' are used to guarantee the correctness of the various annotations (notably, the correctness of the encoding of the run ρ and that of the output annotations).

4 Synthesis of Resynchronizers

Recall that containment between transducers depends on the adopted semantics. More precisely, according to the classical semantics, T_1 is contained in T_2 (denoted $T_1 \subseteq T_2$) if all input-output pairs realized by T_1 are also realized by T_2; according to the origin semantics, T_1 is contained in T_2 (denoted $T_1 \subseteq_o T_2$) if all origin graphs realized by T_1 are also realized by T_2. In this section, we study the following variant of the containment problem:

Resynchronizer synthesis problem.
Input: two transducers T_1, T_2.
Question: does there exist some resynchronization R such that $T_1 \subseteq_o R(T_2)$.

In fact, the above problem comes in several variants, depending on the model of transducers considered (one-way or two-way) and the class of admissible resynchronizations R (rational or bounded regular). Moreover, for the positive instances of the above problem, we usually ask to compute a witnessing resynchronization R from the given T_1 and T_2 (this is the reason for calling the problem a *synthesis problem*).

Clearly, the synthesis problem for unrestricted resynchronizers is equivalent to a classical containment, that is, $T_1 \subseteq T_2$ if and only if $T_1 \subseteq_o R(T_2)$ for some resynchronizer R. Therefore, the synthesis problem for unrestricted resynchronizers is undecidable. Thus we

will consider the synthesis problem of rational (resp. bounded regular) resynchronizers for one-way (resp. two-way) transducers.

We also recall that rational resynchronizers preserve definability of relations by one-way transducers [10], while bounded regular resynchronizers (which, by Theorem 3, are strictly more expressive than rational resynchronizers) preserve definability by two-way transducers [6]. For the sake of presentation, we shall first consider the synthesis of rational resynchronizers in the functional one-way setting, that is, for instances given by functional one-way transducers. We show that in this setting the problem collapses again to the classical containment problem, which is however decidable now, that is: $T_1 \subseteq T_2$ if and only if $T_1 \subseteq_o R(T_2)$ for some rational resynchronizer R. The decidability result can be slightly extended to some non-functional transducers. More precisely, we will show that synthesis of rational resynchronizers for finite-valued one-way transducers is still decidable. When moving to the relational case, however, the problem becomes undecidable.

The decidability status in the one-way setting could be also contrasted with the two-way setting. In this respect, we observe that, in the functional case, the synthesis problem does not collapse anymore to classical containment, as there are functional two-way transducers T_1, T_2 such that $T_1 \subseteq T_2$, but for which no bounded regular resynchronizer R satisfies $T_1 \subseteq_o R(T_2)$ (an example can be found at the beginning of Section 4.3). We are able to prove decidability of synthesis of bounded, regular resynchronizers for unambiguous two-way transducers. The decidability status, however, remains open in the functional two-way case, as well as in the unrestricted (non-functional) two-way case.

4.1 Resynchronizing functional, one-way transducers

Recall that it can be decided in PSPACE whether a transducer (be it one-way or two-way) is functional [2], and that the classical containment problem for functional (one-way/two-way) transducers is also in PSPACE [3]. The following result shows that, for functional one-way transducers, classical containment and rational resynchronizer synthesis are inter-reducible.

▶ **Theorem 4.** *Let T_1, T_2 be two functional one-way transducers. The following conditions are equivalent, and decidable:*

1. $T_1 \subseteq T_2$,
2. $T_1 \subseteq_o R(T_2)$ *for some resynchronization* R,
3. $T_1 =_o R(T_2)$ *for some rational resynchronizer* R.

Proof sketch. The implications from 2. to 1. and 3. to 2. are trivial. The implication from 1. to 3. is proved by constructing a rational resynchronizer R as a product of T_1, T_2: at each step, R consumes a symbol $a \in \Sigma$ and a word $v_1 \in \Gamma^*$ from a transition of T_1 and produces the same symbol a and possibly a different word $v_2 \in \Gamma^*$ from a corresponding transition of T_2. The fact that R preserves the outputs relies on functionality of T_1 and T_2. ◀

A natural question arises: can a characterization similar to Theorem 4 be obtained for transducers that compute arbitrary relations, rather than just functions? The example below provides a negative answer to this question. Later in Section 4.2, we will see that synthesis of rational resynchronizers for unrestricted one-way transducers is an undecidable problem.

▶ **Example 5.** Consider a one-way transducer T_1 that checks that the input is from $(aa)^*$ and produces a single output letter b for each consumed input letter a, and another transducer T_2 that works in two phases: during the first phase, it produces two b's for each consumed a, and during the second phase consumes the remaining part of the input without producing any output. We have $T_1 \subseteq T_2$, but $T_1 \not\subseteq_o T_2$. The only resynchronization R that satisfies $T_1 \subseteq_o R(T_2)$ must map synchronized words from $(ab)^*$ to $(abb)^*(a)^*$, while preserving the

number of a's and b's. Such a transformation cannot be defined by any rational resynchronizer, nor by a bounded regular resynchronizer.

There is however an intermediate case, between the functional and the full relational case, for which a generalization of Theorem 4 is possible. This is the case of *finite-valued* one-way transducers, that is, transducers that realize finite unions of partial functions. The generalization exploits a result from [10], stated just below, that concerns synthesis of bounded-delay resynchronizers. Formally, given two origin graphs G and G' with the same input and output projections, and given an input position y, we denote by $\mathsf{delay}_{G,G'}(y)$ the difference between the largest $x \in \mathsf{dom}(\mathsf{out}(G))$ such that $\mathsf{orig}(G)(x) = y$ and the largest $x' \in \mathsf{dom}(\mathsf{out}(G'))$ such that $\mathsf{orig}(G')(x') = y$. Given $d \in \mathbb{N}$, we define the *d-delay resynchronizer* as the resynchronization that contains all pairs (G, G') with the same input and output projections and such that $\mathsf{delay}_{G,G'}(y) \in [-d, +d]$ for all input positions y. It is easy to see that the d-delay resynchronizer is a special case of a rational resynchronizer.

▶ **Theorem 6** (Theorem 13 in [10]). *Let T_1, T_2 be one-way transducers, where T_2 is k-ambiguous.*[2] *One can compute a d-delay resynchronizer R_d, for some $d \in \mathbb{N}$, such that $T_1 \subseteq T_2$ implies $T_1 \subseteq_o R_d(T_2)$.*

As a corollary we can generalize Theorem 4 to k-valued one-way transducers, with the only difference that the witnessing rational resynchronizer now satisfies $T_1 \subseteq_o R(T_2)$ rather than $T_1 =_o R(T_2)$. We also recall that classical containment remains decidable for k-valued one-way transducers, thanks to the fact that these can be effectively transformed to finite unions of functional transducers [15]:

▶ **Corollary 7.** *Let T_1, T_2 be k-valued one-way transducers. The following conditions are equivalent, and decidable:*

1. $T_1 \subseteq T_2$,
2. $T_1 \subseteq_o R(T_2)$ *for some resynchronization R,*
3. $T_1 \subseteq_o R(T_2)$ *for some rational resynchronizer R.*

Proof. We prove the only interesting implication from 1. to 3. Suppose that T_1, T_2 are k-valued one-way transducers such that $T_1 \subseteq T_2$. Using the decomposition theorem from [15], we can construct a k-ambiguous one-way transducer T_2' that is classically equivalent to T_2 and such that $T_2' \subseteq_o T_2$. Since $T_1 \subseteq T_2'$, by Theorem 6 we can compute a d-delay (in particular, rational) resynchronizer R_d such that $T_1 \subseteq_o R_d(T_2')$. Finally, since $T_2' \subseteq_o T_2$, $T_1 \subseteq_o R_d(T_2')$, and $R_d(T_2') \subseteq_o R_d(T_2)$, we get $T_1 \subseteq_o R_d(T_2)$. ◀

4.2 Resynchronizing arbitrary one-way transducers

In the previous section we saw how to synthesize a rational resynchronizer for functional, or even finite-valued, one-way transducers. One may ask if finite-valuedness is necessary. We already know that classical containment $T_1 \subseteq T_2$ is undecidable [11, 12] for arbitrary one-way transducers, whereas origin-containment $T_1 \subseteq_o T_2$ is decidable [6]. Synthesis of a rational resynchronizer R such that $T_1 \subseteq_o R(T_2)$ is a question that lies between the two questions above. We show in this section that in the case of *real-time* transducers with unary output alphabet, the latter question is equivalent to language-boundedness of one-counter automata, a problem that we define below.

A transducer is said to be *real-time* if it produces bounded outputs for each consumed input symbol. A *one-counter automaton* (OCA) is a non-deterministic pushdown automaton

[2] A transducer is k-ambiguous if each input admits at most k successful runs.

with a single stack symbol, besides the bottom stack symbol. In the definition of the language-boundedness problem, we assume that the OCA recognizes a universal language; this assumption is used in the reduction to the synthesis problem.

Language-boundedness of OCA.
Input: An OCA A over alphabet Ω that recognizes the *universal* language $L(A) = \Omega^*$.
Question: Does there exist some bound k such that every word over Ω can be accepted by A with a run where the counter never exceeds k?

Our reductions between language-boundedness of OCA and synthesis of rational resynchronizers rely on the following result from [10], that implies that bounded-delay resynchronizers are enough for synthesizing resynchronizers of real-time transducers:

▶ **Theorem 8** (Theorem 11 in [10]). *Let T_1, T_2 be real-time, one-way transducers and R a rational resynchronizer such that $T_1 \subseteq_o R(T_2)$. One can compute a d-delay resynchronizer R_d such that $T_1 \subseteq_o R_d(T_2)$.*

▶ **Proposition 9.** *Synthesis of rational resynchronizers for real-time one-way transducers with unary output alphabet and language-boundedness of OCA are inter-reducible problems. Moreover, in the reductions, one can assume that the left hand-side transducer is functional.*

Proof sketch. Given some real-time transducers T_1, T_2, one constructs an OCA A that, when the input encodes a successful run of T_1, guesses and simulates an equivalent successful run of T_2. The OCA A keeps track in its counter, how ahead or behind is the partial output produced by the encoded run of T_1 compared to the partial output produced by the simulated run T_2, and accepts with an empty counter. Moreover, A accepts all inputs that do not encode successful runs of T_1: as soon as an error is detected, the counter is reset and frozen. Thus, badly-formed encodings do not affect language-boundedness. Then, using Theorem 8, one shows that A is language-bounded if and only if $T_1 \subseteq_o R(T_2)$ for some rational (and w.l.o.g. bounded-delay) resynchronizer R.

In the opposite reduction, one has to construct some real-time transducers T_1, T_2 from a given OCA A. Both transducers receive inputs over the same alphabet as A. T_1 is a simple functional transducer that outputs one symbol for each consumed input symbol. T_2, instead, guesses and simulates a run of A, and ouputs two symbols when the counter of the OCA increases, and no symbol when it decreases. As before, one argues using Theorem 8 that A is language-bounded if and only if $T_1 \subseteq_o R(T_2)$ for some rational resynchronizer R. ◀

The status of the problem of language-boundedness of OCA was open, to the best of our knowledge. Piotr Hofman communicated to us the following unpublished result, which can be obtained by a reduction from the undecidable boundedness problem for Minsky machines (the proof is in the extended version of this paper):

▶ **Theorem 10** ([13]). *The language-boundedness problem for OCA is undecidable.*

▶ **Corollary 11.** *Synthesis of rational resynchronizers for (real-time) one-way transducers is undecidable, and this holds even when the left hand-side transducer is functional.*

4.3 Resynchronizing unambiguous, two-way transducers

We now focus on the resynchronizer synthesis problem for two-way transducers. Here the appropriate class of resynchronizations is that of regular resynchronizers, since, differently from rational resynchronizer, they can handle origin graphs induced by two-way transducers.

The situation is more delicate, as the synthesis problem does not reduce anymore to classical containment. As an example, consider the transducer T_1 that consumes an input of the form a^* from left to right, while copying the letters to the output, and a two-way transducer T_2 that realizes the same function but while consuming the input in reverse. We have that $T_1 \subseteq T_2$, but there is no resynchronizer R that satisfies $T_1 \subseteq_o R(T_2)$ and that is bounded and regular at the same time. As we will see, extending Theorem 4 to two-way transducers is possible if we move beyond the class of regular resynchronizers and consider bounded resynchronizers defined by Parikh automata. The existence of bounded regular resynchronizers between functional two-way transducers can thus be seen as a strengthening of the classical containment relation. Unfortunately, we are only able to solve the synthesis problem of bounded regular resynchronizers for *unambiguous* two-way transducers, so the problem remains open for functional two-way transducers.

First we introduce resynchronizers definable by Parikh automata. Formally, a *Parikh automaton* is a finite automaton $A = (\Sigma, Q, I, E, F, Z, S)$ equipped with a function $Z : E \to \mathbb{Z}^k$ that associates vectors of integers to transitions and a semi-linear set $S \subseteq \mathbb{Z}^k$. A successful run of A is a run starting in I, ending in F and such as the sum of the weights of its transitions belongs to S. We say that A is *unambiguous* if the underlying finite automaton is. In this case, we can associate with each input u the vector $A(u) \in \mathbb{Z}^k$ associated with the unique accepting run of the underlying automaton of A on u, if this exists, otherwise $A(u)$ is undefined. By taking products, one can easily prove that unambiguous Parikh automata are closed under pointwise sum and difference, that is, given A_1 and A_2, there are A_+ and A_- such that $A_+(u) = A_1(u) + A_2(u)$ and $A_-(u) = A_1(u) - A_2(u)$ for all possible inputs u. Hereafter, we will only consider languages recognized by *unambiguous* Parikh automata with the trivial semilinear set $S = \{0^k\}$.

By a slight abuse of terminology, we call *Parikh resynchronizer* any resynchronizer with parameters whose relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ are recognizable by unambiguous Parikh automata, and ipar and opar are regular. We naturally inherit from regular resynchronizers the notion of boundedness. Moreover, we introduce another technical notion, that will be helpful later. Given a resynchronizer R, we define its *target set* as the set of all pairs (u, z) where u is an input, z is a position in it, and $(w, y, z) \in \mathsf{move}_\gamma$ for some annotation w of u with input parameters, some input position y, and some output type γ. Similarly, we define the *target set* of a two-way transducer T as the set of all pairs (u, z), where $u = \mathsf{in}(G)$ and $z \in \mathsf{orig}(G)(x)$ for some $x \in \mathsf{dom}(\mathsf{out}(G))$ and some origin graph G realized by T.

▶ **Theorem 12.** *Let T_1, T_2 be two unambiguous two-way transducers. The following conditions are equivalent:*

1. $T_1 \subseteq T_2$,
2. $T_1 \subseteq_o R(T_2)$ *for some resynchronization R,*
3. $T_1 =_o R(T_2)$ *for some 1-bounded Parikh resynchronizer R whose target set coincides with that of T_1 and where, each relation $\mathsf{next}_{\gamma,\gamma'}$ is regular if move_γ and $\mathsf{move}_{\gamma'}$ are regular.*

Proof sketch. We focus on the implication from 1. to 3., as the other implications are trivial. Similarly to the one-way case, to synthesize a resynchronizer, we need to annotate the input with the (unique) successful runs of T_1 and T_2 (if these runs exist). Since T_1, T_2 are two-way, the natural way of doing it is to use *crossing sequences*. Thanks to the encoding of runs by means of crossing sequence, we can describe any output position x with a pair (y, i), where y is the origin of x (according to T_1 or T_2) i is the number of output positions before x with the same origin y. Note that i is bounded, as the transducers here are unambiguous, and hence every input position is visited at most a bounded number of times.

Given $T = T_1$ or $T = T_2$ and an index i, one can construct a unambiguous Parikh automaton $A_{T,i}$ that, when receiving as input a word u with a marked position y, produces the unique position x that is encoded by the pair (y, i), according to the transducer T. It follows that, for every output element correctly annotated with $\gamma = (a, i, j)$, the relation move_γ can be defined as $\{(y, z) \mid A_{T_2,i}(y) - A_{T_1,j}(z) = 0\}$, which is a unambiguous Parikh language. This almost completes the definition of the Parikh resynchronizer R. The remaining components of R consists of suitable relations $\mathsf{next}_{\gamma,\gamma'}$ that check correctness of the annotations. In particular, the relations $\mathsf{next}_{\gamma,\gamma'}$ are obtained by pairing a regular property with properties defined in terms of the prior relations move_γ and $\mathsf{move}_{\gamma'}$, and hence $\mathsf{next}_{\gamma,\gamma'}$ is regular whenever move_γ and $\mathsf{move}_{\gamma'}$ are. ◀

We now explain how to exploit the above characterization to decide bounded regular resynchronizer synthesis problem. We provide the following characterization, whose proof follows from the previous theorem:

▶ **Theorem 13.** *Let T_1, T_2 be two unambiguous two-way transducers such that $T_1 \subseteq T_2$, and let $\hat{R}$ be the bounded Parikh resynchronizer obtained from Theorem 12. The following conditions are equivalent:*

1. *$\hat{R}$ is a regular resynchronizer,*
2. *$T_1 \subseteq_o R(T_2)$ for some bounded regular resynchronizer R,*
3. *$T_1 \subseteq_o R(T_2)$ for some 1-bounded regular resynchronizer R,*
4. *$T_1 =_o R(T_2)$ for some 1-bounded regular resynchronizer R with the same target set as T_1.*

Theorems 12 and 13 together provide a characterization of those pairs of unambiguous two-way transducers T_1, T_2 for which there is a bounded regular resynchronizer R such that $T_1 \subseteq_o R(T_2)$. The effectiveness of this characterization stems from the decidability of regularity of languages recognized by unambiguous Parikh automata [7]. This result requires unambiguity and uses Presburger arithmetics to determine for each (simple) loop a threshold such that iterating the loop more than the threshold always satisfies the Parikh constraint. The language of the Parikh automaton is regular if and only if every (simple) loop has such a threshold. We thus conclude:

▶ **Corollary 14.** *Given two unambiguous two-way transducers T_1, T_2, one can decide whether there is a regular resynchronizer R such that $T_1 \subseteq_o R(T_2)$.*

5 Conclusions

We studied two notions of resynchronization for transducers with origin, called rational resynchronizer and regular resynchronizer. Rational resynchronizers are suited for transforming origin graphs of one-way transducers, while regular resynchronizers can be applied also to origin graphs of two-way transducers. We showed that the former are strictly included in the latter, even when restricting the origin graphs to be one-way. We then studied the following variant of containment problem for transducers: given two transducers T_1, T_2, decide whether $T_1 \subseteq_o R(T_2)$ for some (rational or regular) resynchronizer R. That is, if all origin graphs of T_1 can be seen as some origin graph of T_2 transformed according to R, then compute such a resynchronizer R. This problem can be seen as a synthesis problem of resynchronizers. It is shown that the synthesis problem is decidable when T_1, T_2 are finite-valued one-way transducers and the resynchronizer is constrained to be rational, as well as when T_1, T_2 are unambiguous two-way transducers and the resynchronizer is allowed to be regular (and bounded). In the one-way setting, the problem turns out to be undecidable

already for unrestricted (non-functional) transducers and rational resynchronizers. In the two-way setting, the decidability status remains open already when the transducers are not unambiguous (be them functional or not). Concerning this last point, however, we recall that the synthesis problem becomes undecidable as soon as we consider regular resynchronizers that are unbounded, as in this case the problem is at least as hard as classical containment.

References

1 Rajeev Alur and Pavel Cerný. Expressiveness of streaming string transducer. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

2 Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theor. Comput. Sci.*, 292:45–63, 2003.

3 Meera Blattner and Tom Head. Single-valued a-transducers. *J. Comput. and System Sci.*, 15:310–327, 1977.

4 Mikolaj Bojańczyk. Transducers with origin information. In *International Colloquium on Automata, Languages and Programming (ICALP'14)*, number 8572 in LNCS, pages 26–37. Springer, 2014.

5 Mikolaj Bojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. Which Classes of Origin Graphs Are Generated by Transducers? In *International Colloquium on Automata, Languages and Programming (ICALP'17)*, volume 80 of *LIPIcs*, pages 114:1–114:13, 2017.

6 Sougata Bose, Anca Muscholl, Vincent Penelle, and Gabriele Puppis. Origin-Equivalence of Two-Way Word Transducers Is in PSPACE. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'18)*, volume 122 of *LIPIcs*, pages 1–18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

7 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Unambiguous constrained Automata. *Int. J. Found. Comput. Sci.*, 24(7):1099–1116, 2013. doi:10.1142/S0129054113400339.

8 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

9 Calvin C. Elgot and Jorge E. Mezei. On Relations Defined by Generalized Finite Automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965. doi:10.1147/rd.91.0047.

10 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On Equivalence and Uniformisation Problems for Finite Transducers. In *International Colloquium on Automata, Languages and Programming (ICALP'16)*, volume 55 of *LIPIcs*, pages 125:1–125:14, 2016.

11 Patrick C. Fischer and Arnold L. Rosenberg. Multi-tape one-way nonwriting automata. *J. Comput. and System Sci.*, 2:88–101, 1968.

12 T. V. Griffiths. The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. *J. ACM*, 15(3):409–413, 1968.

13 Piotr Hofman. Personal communication.

14 Oscar H. Ibarra. The unsolvability of the equivalence problem for e-free NGSM's with unary input (output) alphabet and applications. *SIAM J. of Comput.*, 7(4):524–532, 1978.

15 Andreas Weber. Decomposing a k-Valued Transducer into k Unambiguous Ones. *ITA*, 30(5):379–413, 1996.