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.2019.55

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].

© S. Bose, S. Krishna, A. Muscholl, V. Penelle, G. Puppis;
licensed under Creative Commons License CC-BY

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.

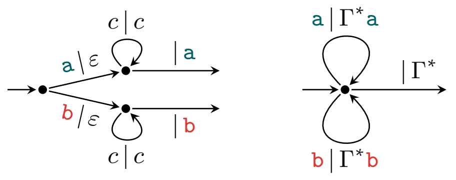
Full proofs of the results presented in this paper can be found in the extended version `https://arxiv.org/abs/1906.08688`.

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, the figure below shows two one-way transducers with input alphabet $\Sigma = \{a, b\}$ 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.


$c \mid c$
$\mathtt{a} \mid \varepsilon$
$\mid \mathtt{a}$
$\mathtt{b} \mid \varepsilon$
$\mid \mathtt{b}$
$c \mid c$
$\mathtt{a} \mid \Gamma^* \mathtt{a}$
$\mid \Gamma^*$
$\mathtt{b} \mid \Gamma^* \mathtt{b}$

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, $Q_{\prec}$ and $Q_{\succ}$, the set I of initial states is contained in $Q_{\succ}$, and the set E of transition rules is contained in $(Q \times \Sigma \times Q) \uplus (Q_{\prec} \times \{\vdash\} \times Q_{\succ}) \uplus (Q_{\succ} \times \{\dashv\} \times Q_{\prec})$. The partitioning of the set of states is useful for specifying which letter is read from each state: states from $Q_{\prec}$ read the letter to the left, whereas states from $Q_{\succ}$ 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 $i \in \{1, \dots, |u|+1\}$. 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 $L(q, a, q')$):

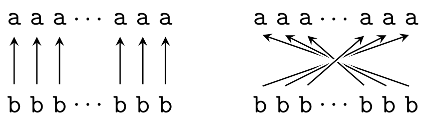
- $(q, i) \xrightarrow{a \mid v} (q', i+1)$ if $(q, a, q') \in E$, $q, q' \in Q_{\succ}$, and $a = u(i)$,
- $(q, i) \xrightarrow{a \mid v} (q', i)$ if $(q, a, q') \in E$, $q \in Q_{\succ}$, $q' \in Q_{\prec}$, and $a = u(i)$,
- $(q, i) \xrightarrow{a \mid v} (q', i-1)$ if $(q, a, q') \in E$, $q, q' \in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

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*. The figure 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

[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.

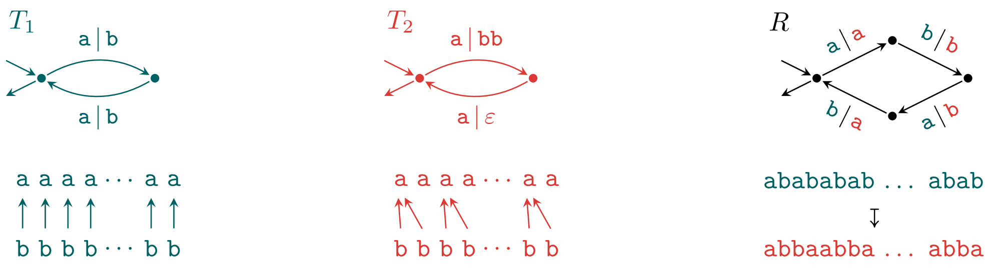


Figure 1 Two functional 1NFT T_1, T_2, their origin graphs, and a rational resynchronizer R.

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 1. 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.

Regular resynchronizers.

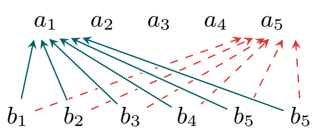
While languages of synchronized words are a faithful representation of rational relations, this notation does not capture regular relations, so relations realized by two-way transducers. An alternative formalism for resynchronizations of relations defined by two-way transducers was proposed in [6] under 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 that contains the pairs (G, G'), where the origin graph G (resp. G') maps every output position to the first (resp. last) input position, as shown in the figure. Note that 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, for example $\big((\mathsf{move}_\gamma)_{\gamma\in\Gamma}, (\mathsf{next}_{\gamma,\gamma'})_{\gamma,\gamma'\in\Gamma}\big)$, where $\mathsf{move}_\gamma = \{(u, y, z) \mid u \in \Sigma^*,\ y = 1,\ z = |u|\}$ and $\mathsf{next}_{\gamma,\gamma'} = \Sigma^* \times \mathbb{N} \times \mathbb{N}$.



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 it 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.

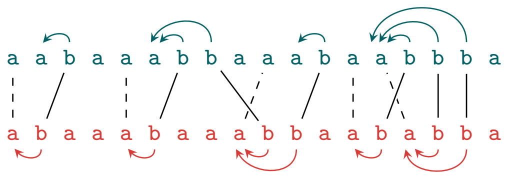
Unless otherwise stated, hereafter we assume that two-way transducers are enhanced with common guess, and regular resynchronizers are enhanced with parameters.

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, the figure to the right represents a possible pair of synchronized words, denoted w and w', shown in blue and in red, respectively, such that $(w, w') \in R$. We assume that $\Sigma = \{a\}$ and $\Gamma = \{b\}$.


a a b a a a b b a a a b a a b b b a
a b a a a b a a a b b a a b a b b a

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

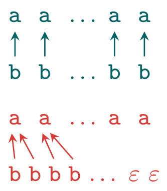
Recall that it can be decided in PSpace whether a transducer (be it one-way or two-way) is functional [2], and that the classical containment problem for functional (one-way/two-way) transducers is also in PSpace [3]. The following result shows that, for functional one-way transducers, classical containment 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. The origin graphs of T_1 and T_2 are shown to the right. We have $T_1 \subseteq T_2$, but $T_1 \nsubseteq_o T_2$. 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 [17]:

▶ **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 [17], 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 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?

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

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. URL: `https://doi.org/10.1142/S0129054113400339`, `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. URL: `https://doi.org/10.1147/rd.91.0047`, `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 Anca Muscholl and Gabriele Puppis. The many facets of string transducers (invited talk). In *36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 126 of *LIPIcs*, pages 2:1–2:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.

16 J.C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.*, 3(2):198–200, 1959.

17 Andreas Weber. Decomposing a k-valued transducer into k unambiguous ones. *ITA*, 30(5):379–413, 1996.

A Proofs of Section 3

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

We fix a one-way transducer R over $\Sigma \uplus \Gamma$ that defines a rational resynchronizer. We assume without loss of generality that R is *letter-to-letter*, as well as *trimmed*, namely, every state in R occurs in some successful run. Note that R maps synchronized words to synchronized words. With a slight abuse of terminology, we shall use the terms 'source' (resp. 'target') to refer to a synchronized word that is an input (resp. an output) of R. When depicting examples, we will often adopt the convention that source synchronized words are shown in blue, while target synchronized words are shown in red. On the other hand, we shall use the terms 'input' and 'output' to refer to the projections of a synchronized word over Σ and Γ, respectively (note that, in this case, it does not matter whether the synchronized word is the source or the target, since these have the same projections over Σ and Γ). The goal is to construct a 1-bounded, regular resynchronizer R', with parameters, that defines the same resynchronization as R.

We begin by introducing the key concept of *lag*, which represents the difference between the number of input symbols consumed and number of input symbols produced along a certain run (not necessarily successful) of R. Formally, given a run of R of the form $\rho = q_0 \xrightarrow{c_1 \,|\, d_1} q_1 \xrightarrow{c_2 \,|\, d_2} \dots \xrightarrow{c_n \,|\, d_n} q_n$, we define its lag $\mathsf{lag}(\rho)$ as $|\pi_\Sigma(c_1 \dots c_n)| - |\pi_\Sigma(d_1 \dots d_n)|$, where π_Σ denotes the operation of projection onto the alphabet Σ. Note that, because R is letter-to-letter, one could have equally defined $\mathsf{lag}(\rho)$ by counting the difference between produced output symbols and consumed output symbols. Further note that the lag of a successful run is always 0, since R preserves the input projection. Notice that the lag of a run is a notion distinct of the *delay* of a rational resynchronizer presented in [10] which is the maximum distance between the target origin of an output position and its source origin. The following lemma shows that the lag is in fact a property of the initial and final states of a run.

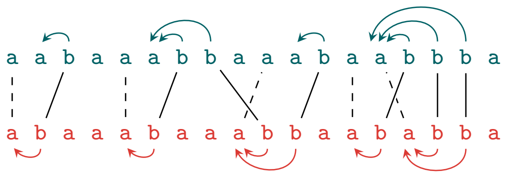
▶ **Lemma 15.** *For every two runs ρ_1 and ρ_2 of R that begin with the same state and end with the same state, $\mathsf{lag}(\rho_1) = \mathsf{lag}(\rho_2)$.*

Proof. Since R is trimmed, both runs ρ_1 and ρ_2 can be completed to some successful runs of the form $\rho'\rho_1\rho''$ and $\rho'\rho_2\rho''$. From $\mathsf{lag}(\rho'\rho_1\rho'') = 0 = \mathsf{lag}(\rho'\rho_2\rho'')$, it immediately follows that $\mathsf{lag}(\rho_1) = 0 - (\mathsf{lag}(\rho') + \mathsf{lag}(\rho'')) = \mathsf{lag}(\rho_2)$. ◀

In view of the above lemma, we can associate a lag $\mathsf{lag}(q)$ with each state q of R as follows: we choose an arbitrary run ρ that starts with the initial state of R and ends with q, and let $\mathsf{lag}(q) = \mathsf{lag}(\rho)$. This is well-defined since $\mathsf{lag}(q)$ does not depend on the particular choice of ρ. For instance, if we consider the letter-to-letter resynchronizer R of Example 1, the only state with non-zero lag is the bottom one, which has lag 1. Note that, because each transition of R can only increase or decrease the lag by 1, all lags range over the finite set $\{-|Q|, \dots, +|Q|\}$, where Q is the state space of R.

Next, we consider a successful run of R, say $\rho = q_0 \xrightarrow{c_1 \,|\, d_1} q_1 \xrightarrow{c_2 \,|\, d_2} \dots \xrightarrow{c_n \,|\, d_n} q_n$, and define relations omatch_ρ and imatch_ρ between positions of ρ. These relations are used later to define a bijection between source and target origins. The relation omatch_ρ consists of all pairs (i,j) of positions of ρ such that c_i and d_j are output letters and $c_1 c_2 \dots c_i =_\Gamma d_1 d_2 \dots d_j$ (the latter is a shorthand for $\pi_\Gamma(c_1 c_2 \dots c_i) = \pi_\Gamma(d_1 d_2 \dots d_j)$). Note that omatch_ρ is in fact a partial bijection. In a similar way, we define imatch_ρ as the partial bijection that contains all pairs (i,j) of positions of ρ such that c_i and d_j are input letters and $c_1 c_2 \dots c_i =_\Sigma d_1 d_2 \dots d_j$.

▶ **Example 16.** Consider the pair of source and target synchronized words over $\Sigma \uplus \Gamma$ shown to the right, where $\Sigma = \{a\}$ and $\Gamma = \{b\}$, which could be realized by a successful run ρ of R. For the moment, we overlook the blue and red arrows. Because R is letter-to-letter, any position in any of the two words corresponds precisely to a position in the run ρ, so we can represent the relations omatch_ρ and imatch_ρ by means of edges between source and target positions. In the figure, the solid edges represent pairs of omatch_ρ, while the dashed edges represent some pairs of imatch_ρ (precisely, those pairs (i,j) such that the transition at position j produces an input letter, while the next transition produces an output letter).


a a b a a a b b a a a b a a b b b a
a b a a a b a a a b b a a b a b b a

Mapping the source to target origins.

We now explain how the relations imatch_ρ and omatch_ρ can be used to define a mapping from source to target origins. We do so by first using the figure of Example 16. Consider any output letter at position i in the source synchronized word w (e.g. the first blue letter b). Let j be the last Σ-labelled position before i, as indicated by the blue arrow. This position j determines the source origin $y = |\pi_\Sigma(w[1,j])|$ of the output letter. To find the corresponding target origin, we observe that the position i is mapped via the relation omatch_ρ (solid line) to some position k in the target synchronized word. Let h be the last Σ-labelled position before k (red arrow), and map h back to a position ℓ in the source via the relation imatch_ρ (dashed line). The position ℓ determines precisely the target origin $z = |\pi_\Sigma(w[1,\ell])|$ of the considered output letter. The above steps describe a correspondence between two positions j and ℓ in ρ, with labels over Σ, that is precisely defined by

$$\exists i,k,h \quad \begin{cases} \rho[j,i] \text{ consumes a word in } \Sigma\Gamma^+ \\ (i,k) \in \mathsf{omatch}_\rho \\ \rho[h,k] \text{ produces a word in } \Sigma\Gamma^+ \\ (h,\ell) \in \mathsf{imatch}_\rho. \end{cases} \tag{$\star$}$$

In the above $\rho[j,i]$ represents the part of ρ between positions j,i (both j,i included).

We denote by match_ρ the relation of all pairs (j,ℓ) that satisfy Equation $(\star)$. Note that match determines an analogous correspondence between source and target origins of the input projection. However, match has two issues: it is not yet a partial bijection (since different output positions may have the same source origin), and it needs to be implemented by means of a regular relation move_γ that only considers positions of the input, plus the label γ of a single position in the output. Below, we explain how to overcome those issues.

The case of bounded output blocks.

Hereafter, we call *output block* any maximal factor of a synchronized word that is labelled over Γ. Intuitively, this corresponds to a maximal factor of the output that originates at the same input position. We first consider, as a simpler case, a rational resynchronizer R that reads *source* synchronized words where the lengths of the output blocks are uniformly bounded by some constant, say B (a similar property holds for the blocks of the target synchronized words, using lag-based arguments). In this case we can encode any successful run ρ of R entirely on the input, by annotating every Σ-labelled position y with a factor ρ_y

of ρ that reads the input symbol at position y, followed by the sequence of output symbols up to the next input symbol. Note that every factor ρ_y has length at most $B+1$. The correctness of this input annotation can be checked by the regular language ipar. Given a factor $\rho_y \in \Sigma\Gamma^+$, $\rho_y[1] \in \Sigma$ is the first position of the factor ρ_y. Likewise, $\rho_y[i,j]$ denotes the subfactor of ρ_y consisting of positions $i, i+1, \dots, j$.

In addition, we also annotate the output word with indices from $\{1, \dots, B\}$, called *offsets*, in such a way that an output position x is annotated with an offset o if and only if it is the o-th output position with the same source origin. Note that the correctness of the annotation cannot be checked by a regular language such as opar that refers only to the output. The check will be done instead by a combined use of the relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$.

We first check that, for every pair of consecutive output positions x and $x+1$ annotated with the offsets o and o', respectively, it holds that $o' = o+1$ or $o' = 1$, depending on whether the *source* origins of x and $x+1$ coincide or not. For this we let $(u, z, z') \in \mathsf{next}_{\gamma,\gamma'}$, with $\gamma = (a, o)$ and $\gamma' = (a', o')$, if

1. either $o' = o+1$ and there is $y = y'$ such that $(u, y, z) \in \mathsf{move}_\gamma$ and $(u, y', z') \in \mathsf{move}_{\gamma'}$,
2. or $o' = 1$ and there are $y < y'$ such that $(u, y, z) \in \mathsf{move}_\gamma$ and $(u, y', z') \in \mathsf{move}_{\gamma'}$.

Recall that the relation $\mathsf{next}_{\gamma,\gamma'}$ must be defined in terms of the *target* origins of x and $x+1$. So it needs to rely on the relation move_γ in order to determine the source origins from the target origins. We assume that for every output type γ the relation move_γ, which will be defined later, determines a *partial bijection* between input positions (we will see that this is indeed the case). Based on these assumptions, the above definition of $\mathsf{next}_{\gamma,\gamma'}$ guarantees that the offsets annotating consecutive positions in the output are either incremented or reset, depending on whether they have the same origin or not.

It remains to check that maximal offset occurring in an output block with origin y coincides with number of output symbols produced by the corresponding factor ρ_y of the run. Thus, we modify slightly the definition of $\mathsf{next}_{\gamma,\gamma'}$ in case 2., as follows:

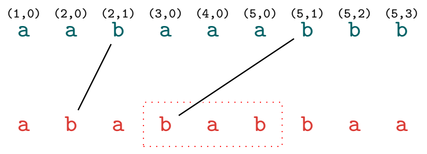
2'. or $o' = 1$ and there are $y < y'$ such that $(u, y, z) \in \mathsf{move}_\gamma$ and $(u, y', z') \in \mathsf{move}_{\gamma'}$, *and* $o = |\rho_y| - 1$.

Note that the factor ρ_y can be derived by inspecting the annotation of the input position y. The modification suffices to guarantee that the output annotation is correct for all output blocks but the last one. The annotation for the last output block can be checked by marking the last output position with a distinguished symbol and by requiring that if γ witnesses the marked symbol and the offset o, then move_γ can only contain a triple of the form (u, y, z), with $o = |\rho_y| - 1$. We omit the tedious definitions in this case.

Now, having the input correctly annotated with the factors ρ_y of ρ and the output correctly annotated with the offsets, we can encode any position i of ρ by a pair (y, o) that consists of a position y of the input and an offset $o \in \{0, 1, \dots, B\}$. The encoding is defined in such a way that $i = \sum_{y'<y} |\rho_{y'}| + o + 1$ (in particular, $o = 0$ when the transition at position i consumes an input symbol, otherwise $o \geq 1$). We use this encoding to translate the relations omatch_ρ, imatch_ρ, and match_ρ, to equivalent finite unions of partial bijections between input positions. We begin by explaining the translation of omatch_ρ.

Translation of omatch_ρ.

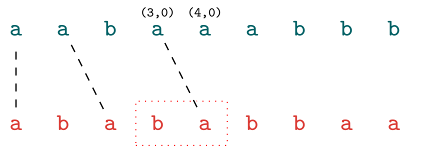
Consider any pair $(i, j) \in \mathsf{omatch}_\rho$. Since the transition at position i of ρ consumes an output symbol, it is encoded by a pair of the form (y, o), with $o \geq 1$. On the other hand, the transition at position j may consume either an input symbol or an output symbol (but does produce an output symbol). In the former case, j is encoded by a pair $(y', 0)$; in the latter case, it is encoded by a pair (y', o'), with $o' \geq 1$. As an example, in the figure below, $(7, 4) \in \mathsf{omatch}_\rho$. Position 7 of the run is encoded as $(5, 1)$ on the input. The transition at position 4 consumes an input symbol a, and produces the output symbol b, and is encoded as $(3, 0)$.



In general, we observe that the lag induced just after the o-th transition of ρ_y must be equal to the number of output symbols produced between the $(o' + 1)$-th transition of $\rho_{y'}$ and the o-th transition of ρ_y, both included (when the lag is negative one follows the transitions in reverse order, counting negatively). As an illustration in the figure, the lag after the first transition of ρ_5 is 2, which is the number of output symbols in the dotted box. The dotted box consists of the symbols produced between the first transition of $\rho_{3'}$ and the first transition of ρ_5, and has two output symbols.

Translation of imatch_ρ.

The translation of the relation imatch_ρ is similar. The only difference is that now the pairs $(i, j) \in \mathsf{imatch}_\rho$ are encoded by tuples of the form $\big((y, o), (y', o')\big)$, with $o = 0$ since the transition at i consumes an input symbol. The transition at position j as before, can consume an input symbol or an output symbol. Consider the figure below, where $(2, 3) \in \mathsf{imatch}_\rho$. Position $i = 2$ is encoded as $(2, 0)$. The transition at position 3 consumes an output symbol b (and produces the input symbol a). Position 3 is encoded as $(2, 1)$.



The only difference here is that one has to relate the lag with the number of *input* letters produced between (both positions included) the first transition of ρ_y and the o'-th transition of $\rho_{y'}$. Again, in the figure, the lag after the first transition of ρ_3 is 1, which is the number of input symbols in the dotted box. The dotted box contains the symbols produced between the first transition of ρ_3 and the first transition of $\rho_{4'}$, and has one input symbol.

Relations encoding omatch_ρ and imatch_ρ.

So we can represent omatch_ρ as a finite union of relations $O_{o,o'} \subseteq (\Sigma \times \Sigma')^* \times \mathbb{N} \times \mathbb{N}$, each describing a regular property of annotated inputs with two distinguished positions in it, in such a way that the positions are bijectively related to one another.

Likewise, we can represent imatch_ρ as a finite union of relations $I_{0,o'}$, each describing a regular property of annotated inputs with two distinguished positions encoded as $(y, 0)$ and (y', o') in it, which are bijectively related to one another.

Translation of match_ρ.

We finally turn to the translation of the relation match_ρ, which will eventually determine the relations move_γ of the desired regular resynchronizer R'. This is done by mimicking Equation $(\star)$ via the encoding of positions in the run ρ using pairs of input positions and offsets, and more precisely, by replacing the variables j, i, k, h, ℓ of Equation (*) with the pairs $(y,0)$, (y,o), (y',o'), (y'',o''), $(z,0)$.

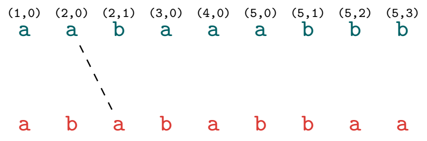
Formally, for every offset $o \in \{1,\dots,B\}$, we define the set M_o of all triples (u,y,z), where u is an annotated input and y, z are positions in it that satisfy the following property:

$$\exists y', y'' \bigvee_{0 \le o', o'' \le B} \begin{cases} \rho_y[1, o+1] \text{ consumes a word in } \Sigma\Gamma^+ \\ (u, y, y') \in O_{o,o'} \\ \rho_{y''}[o''+1, |\rho_{y''}|]\ \rho_{y''+1} \dots \rho_{y'-1}\ \rho_{y'}[1, o'+1] \text{ produces a word in } \Sigma\Gamma^+ \\ (u, z, y'') \in I_{0,o''}. \end{cases} \tag{$\star\star$}$$

Note that the first condition holds trivially by definition of ρ_y, while the third condition is easily implemented by accessing the factors $\rho_{y''}, \dots, \rho_{y'}$ of ρ that are encoded by the input parameters. For simplicity, here we assumed that (y'', o'') is lexicographically before (y', o'); to treat the symmetric case, one has to interpret the definition by considering the sequence of transitions in reverse. The intended meaning of $(u,y,z) \in M_o$ is as follows. Suppose that the input is correctly annotated with the factors ρ_y of a successful run ρ of R, and that the output position x of ρ is correctly annotated with an offset o. Assuming that x is the o-th output position with source origin y, then z is its target origin in ρ.

Continuing with our running example, we determine the target origin for the point b annotated (5,1), whose source origin is (5,0). We will find the target origin of this b annotated (5,1). As seen in the computation of omatch_ρ, we know that $(u,5,3) \in O_{1,0}$. The factor $\rho_5 = abbb$, and $\rho_5[1,2] = ab \in \Sigma\Gamma^+$, and as we have seen, $(u,5,3) \in O_{1,0}$. Now, consider the part of the source u annotated with $(2,1)(3,0)$. This produces the output $ab \in \Sigma\Gamma^+$. That is, for $y'' = 2, o'' = 1$, and $y' = 3, o' = 0$, we have $\rho_{y''}[o''+1, 2]\ \rho_{y'}[1, o'+1] = \rho_2[2,2]\rho_3[1,1] = ba$ produces the output $ab \in \Sigma\Gamma^+$.

Consider $(z,0) = (2,0)$. The lag after the a at $i = 2$ annotated $(2,0)$ is 1. Also, $(2,3) \in \mathsf{imatch}_\rho$. The position 3 consumes an output and produces an input a. Indeed, the lag after the first transition of ρ_2 is 1, which is the number of input symbols between the first transition of ρ_2 and the second transition ($(o'+1)$th transition) of ρ_2. That is, $(u,2,2) \in I_{0,1}$. Thus, starting with the b annotated $(y,o) = (5,1)$ such that $\rho_5[1,2] \in \Sigma\Gamma^+$, we first obtain $(y',o') = (3,0)$ with $(u,5,3) \in O_{1,0}$. Further, $\rho_2[2,2]\rho_3[1,1]$ produces a word in $\Sigma\Gamma^+$. Finally, we have $(u,2,2) \in I_{0,1}$, obtaining $(u,5,2) \in M_1$.



Definition of move_γ.

It is tempting to define move_γ just as M_o, for every $\gamma = (a,o) \in \Gamma \times \{1,\dots,B\}$. However, we recall that the correctness of the output annotation is guaranteed only once we are sure that every relation move_γ defines a partial bijection between input positions y and z (hereafter we say for short that the relation is bijective), which is not known a priori. Bijectiveness must then be enforced syntactically, without relying on annotations: for this it suffices to define move_γ as $\{(u,y,z) \in M_o \mid \forall (u,y',z') \in M_o\ (y = y') \leftrightarrow (z = z')\}$, and observe that either M_o is bijective, and hence $\mathsf{move}_\gamma = M_o$, or it is not, and in this case move_γ is a subrelation of M_o that is still bijective. Note that, in the case where move_γ is a subrelation of M_o, there

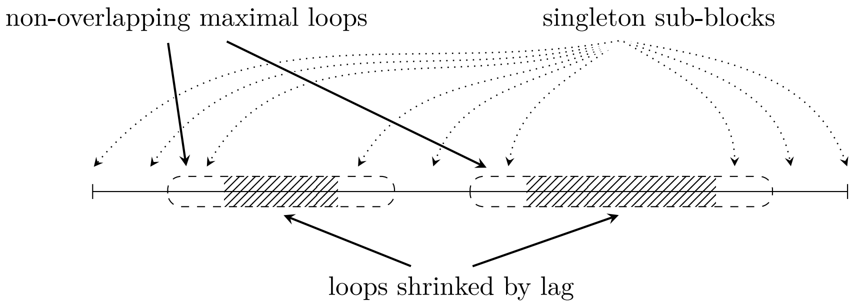


Figure 2 Factorization of an output block.

will be no induced pair of synchronized words, since the origins of some output elements could not be redirected. This is fine, and actually needed, in order to avoid generating with R' spurious pairs of synchronized words that are not also generated by R. On the other hand, observe that the relation move_γ does generate, for appropriate choices of the output annotations, all the pairs of synchronized words that are generated by R. We finally observe that the relations move_γ and $\mathsf{next}_{\gamma,\gamma'}$ are regular. We obtain in this way, a 1-bounded, regular resynchronizer R' equivalent to R.

The general case.

We now aim at generalizing the previous ideas to capture a rational resynchronizer R with source output blocks of possibly unbounded length. One additional difficulty is that we cannot anymore encode a successful run ρ of R entirely on the input, as ρ may have arbitrarily long factors on outputs blocks. Another difficulty is that we cannot uniquely identify the positions in an output block using offsets ranging over a fixed finite set. We will see that a solution to both problems comes from covering most of the output by factors in which the positions behave similarly in terms of the source-to-target origin transformation. Intuitively, each of these factors can be thought of as a 'pseudo-position', and accordingly the output blocks can be thought of as having boundedly many pseudo-positions. This will make it possible to apply the same ideas as before. We now state the key lemma that identifies the aforesaid factors. By a slight abuse of terminology, we call output blocks also the maximal Γ-labelled factors of a synchronized word.

▶ **Lemma 17.** *Let ρ be a successful run of R, and let w and w' be the source and target synchronized words induced by ρ.*

- *Every output block v of w can be factorized into $\mathcal{O}(|Q|^2)$ sub-blocks $v_1, \dots, v_n$ such that if $|v_i| > 1$ and ρ_i is the factor of ρ that corresponds to v_i, then all states in ρ_i have the same lag, say ℓ_i, and the factor obtained by extending ρ_i to the left and to the right by exactly $|\ell_i|$ transitions forms a loop of R.*
- *Moreover, for every factorization $v = v_1 \dots v_n$ as above, each sub-block v_i is also a factor of w', and hence all positions in v_i have the same target origin.*

Proof. We prove the first claim of the lemma (Figure 2 provides an intuitive account of the constructions). Let v be an output block of the source synchronized word w and let ρ' be the factor of the run ρ aligned with v. As a preliminary step, we fix a maximal set of pairwise non-overlapping maximal loops inside ρ', say $\rho'_1, \dots, \rho'_m$. A simple counting argument shows that $m \leq |Q|$ and that there are at most $|Q|$ positions in ρ' that are not covered by the loops $\rho'_1, \dots, \rho'_m$. The latter positions determine some sub-blocks of v of length 1. The remaining sub-blocks of v will be obtained by factorizing the loops $\rho'_1, \dots, \rho'_m$, as follows. Consider any loop ρ'_j. By construction, all letters consumed by ρ'_j occur in v, so they must be output letters. Similarly, all letters produced by ρ'_j are also output letters, since otherwise, by considering repetitions of the loop ρ'_j, one could get different lags, violating Lemma 15. This means that the lag associated with the states along ρ'_j is constant, say ℓ_j ($\leq |Q|$). If ρ'_j has length at most $2|\ell_j|$, then we simply decompose it into $2|\ell_j|$ factors of length 1. Otherwise, we cover a prefix of ρ'_j with $|\ell_j|$ factors of length 1, and a suffix of ρ'_j with $|\ell_j|$ other factors of length 1. The remaining part of ρ'_j is covered by a last factor of length $|\rho'_j| - 2|\ell_j|$. Overall, this induces a factorization of v into at most $|Q|$ (the sub-blocks not covered by a loop) + $|Q| \cdot (2|Q| + 1)$ (Each ρ'_j is decomposed into $(2\ell_j + 1) \leq (2|Q| + 1)$ sub-blocks). This gives $\mathcal{O}(|Q|^2)$ sub-blocks $v_1, \dots, v_n$. Moreover, by construction, if $|v_i| > 1$, then in the corresponding factor ρ_i of ρ, all states have the same lag, say ℓ_i, and if we extend ρ_i to the left and to the right by exactly $|\ell_i|$ transitions, we get back one of the loops ρ'_j (recall that each loop ρ'_j of length $> 2|\ell_j|$ is decomposed into $|\ell_j|$ blocks of length 1, then a block of length $|\rho'_j| - |\ell_j|$, and finally, $|\ell_j|$ blocks of length 1. Clearly, if we extend the middle block on either side by blocks of length $|\ell_j|$, then we get back ρ'_j. This proves the first claim of the lemma.

As for the second claim, suppose that $v_1, \dots, v_n$ is a factorization of an output block v of w satisfying the first claim. Clearly, every sub-block v_i of length 1 is also a factor of the target synchronized word w'. The interesting case is when a sub-block v_i has length larger than 1. In this case, by the previous claim, we know that in the corresponding factor ρ_i of ρ, all states have the same lag ℓ_i, and the factor ρ'_i of ρ that is obtained by expanding ρ_i to the left and to the right by $|\ell_i|$ transition is a loop. In fact, since ρ'_i is a loop, we also know that all states in it have lag ℓ_i. Now, to prove that v_i is a factor of the target synchronized word w', it suffices to show that every two consecutive positions of ρ_i are mapped to consecutive positions via the relation omatch_ρ. This follows almost by construction, since for every pair $(i', k') \in \mathsf{omatch}_\rho$, if i' occurs inside the factor ρ_i, then k' occurs inside the loop ρ'_i (recall that ρ'_i consumes and produces only output symbols), and hence $k' = i' - \ell_i$. In addition, if $i' + 1$ also occurs inside ρ_i, then clearly $(i' + 1, k' + 1) \in \mathsf{omatch}_\rho$. This proves that v_i is a factor of the target synchronized word w', and hence all positions in it have the same target origin. ◀

In view of the above lemma we can guess a suitable factorization of the output into sub-blocks that refine the output blocks, and treat each sub-block as if it were a single position. In particular, we can annotate every sub-block with a unique offset from a finite set of quadratic size w.r.t. $|Q|$. The role of the offsets will be the same as in the previous proof, where blocks had bounded length, namely, determine some partial bijections $O_{o,o'}$, $I_{0,o'}$, and M_o between positions of the input. In addition, we annotate every sub-block with the pair consisting of the first and last states of the factor of the successful run that consumes that sub-block. We call such a pair of states a *pseudo-transition*, as it plays the same role of a transition associated with a single output position. Finally, we annotate every input position y with a sequence of bounded length that represents a single transition on y followed by the pseudo-transitions on the subblocks with source origin y. The resulting input annotation provides an abstraction of a successful run of R.

The correctness of the above annotations can be enforced by defining suitable relations ipar, opar, $\mathsf{next}_{\gamma,\gamma'}$ for the regular resynchronizer R'. We omit the tedious details concerning these relations, and only observe that, as before, the definition $\mathsf{next}_{\gamma,\gamma'}$ relies on the fact that move_γ and $\mathsf{move}_{\gamma'}$ define partial bijections between input positions.

Finally, we turn to describing the relation move_γ that maps source to target origins for γ-labelled output positions. The definition is basically the same as before, based on some auxiliary relations $O_{o,o'}$ and $I_{0,o''}$ that implement omatch_ρ and imatch_ρ at the level of input positions. As before, we guarantee, by means of a syntactical trick, that move_γ determines a partial bijection between input positions. In conclusion, we get a regular resynchronizer R', with input and output parameters, that is equivalent to the rational resynchronizer R.

B Proofs of Section 4

▶ **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. One implication, from 2. to 1., is trivial, since origin containment implies classical containment, and since applying an arbitrary resynchronization R to T_2 cannot result in having more input-output pairs (it can however modify the origin, as well as discard some input-output pairs). The implication from 3. to 2. is also trivial.

For the remaining implication, from 1. to 3., suppose that T_1, T_2 are functional one-way transducers such that $T_1 \subseteq T_2$. We construct a rational resynchronizer R over the disjoint union $\Sigma \uplus \Gamma$ of the input and output alphabets of T_1, T_2, using a variant of the direct product of T_1 and T_2. More precisely, let $T_1 = (Q_1, q_1, \Delta_1, F_1)$, $T_2 = (Q_2, q_2, \Delta_2, F_2)$, and $R = (Q, q, \Delta, F)$, where $Q = Q_1 \times Q_2$, $q = (q_1, q_2)$, $F = F_1 \times F_2$, Δ contains all transitions of the form $(s_1, s_2) \xrightarrow{aw_2 \,|\, aw_1} (t_1, t_2)$, with $s_i \xrightarrow{a \,|\, w_i} t_i$ in Δ_i for both $i = 1$ and $i = 2$. Intuitively, the transducer R simulates a run of T_1 and a run of T_2 in parallel, by repeatedly consuming an input symbol a and the corresponding output w_2 produced by T_2, and producing the same input symbol a and the corresponding output w_1 of T_1. Since T_1 and T_2 are functional and classically contained one in the other, we have that R maps strings over $\Sigma \uplus \Gamma$ to strings over $\Sigma \uplus \Gamma$ while preserving the projections on the input and on the output alphabets. This means that R is indeed a resynchronizer. Finally, T_1 is clearly origin equivalent to $R(T_2)$. ◀

▶ **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. We first prove the reduction from synthesis of rational resynchronizers to language-boundedness of OCA, and then prove the reduction in the opposite direction.

From synthesis to language-boundedness.

Let T_1, T_2 be real-time, one-way transducers with unary output alphabet. We suppose in addition that T_1 is trimmed. We construct an OCA A that reads encodings of successful runs of T_1. If the input is not a successful run of T_1, then, as soon as an error is detected, A resets its counter and accepts any continuation of the input. In particular, thanks to this behaviour and to T_1 being trimmed, badly-formed encodings of runs will not cause the counter of A to be unbounded.

Consider now an input for A that is a correct encoding of a successful run of T_1, say ρ_1. In this case, A guesses and simulates a successful run ρ_2 of T_2 having the same input as ρ_1. The counter of A is used as expected: it is incremented according to the outputs produced using the transitions of ρ_1, and decremented according to the outputs produced using the transitions of ρ_2, or vice versa when one needs to represent a negative value (recall that

OCA work with counter over natural numbers). The detail regarding which among T_1, T_2 is "leading", resulting in the non-negative counter value can be stored in the finite control of the OCA.

Intuitively, a configuration of A determines 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. The OCA A accepts with empty counter. Note that this construction is close to the direct product of T_1 and T_2, the main difference being the treatment of the badly formed encodings and the role played by the counter.

Let us now prove that the OCA A is language-bounded if and only if $T_1 \subseteq_o R(T_2)$ for some rational resynchronizer R.

Suppose first that the OCA A is language-bounded, namely, that there is some $k \in \mathbb{N}$ such that every word is accepted by A with a counter that never exceeds k. We can think of the successful runs of A that maintain the counter between 0 and k as runs of a k-delay resynchronizer R. More precisely, we can define a letter-to-letter resynchronizer R, the states of which are the configurations of A with the value of the counter inside $\{0, \ldots, k\}$. On consuming an input letter, R produces the same input letter; on consuming a sequence of j output letters, depending on the simulated transition of A, R produces an output of length $j + h$ if the counter is incremented by h. Likewise, if the simulated transition of A decrements the counter by h, then on reading a sequence of j output symbols, R produces an output of length $j - h$. The run of R is successful if an only if the simulated run of A is so. The fact that A accepts every word with a counter that never exceeds k, immediately implies that $T_1 \subseteq_o R(T_2)$.

Conversely, suppose that $T_1 \subseteq_o R(T_2)$ for some rational resynchronizer R. By Theorem 8, we can assume without loss of generality that R is a k-delay resynchronizer, for some k (that can be even computed from T_1, T_2, and R, but this is immaterial here). From this it is easy to see that A is language-bounded, and precisely, that A accepts every word with a counter that never exceeds k, as when reading a run ρ of T_1, it can guess a run ρ' of T_2 such that $R(\rho') = \rho$.

From language-boundedness to synthesis.

Let A be an OCA. We construct two real-time, one-way transducers T_1, T_2 that have the same input alphabet as A, say Σ, and a singleton output alphabet, say $\Gamma = \{c\}$. The transducer T_1 reads any word $a_1 \ldots a_n \in \Sigma^*$ and outputs one letter c for each consumed input symbol. In particular, the synchronization language of T_1 is $\{a_1 c \ldots a_n c : a_i \in \Sigma, n \geq 0\}$. Note that T_1 is real-time and functional. The transducer T_2 does the following: upon reading $a_1 \ldots a_n$, it guesses a successful run of the OCA A. Whenever the counter is incremented along the guessed run of A, T_2 outputs cc; whenever the counter is decremented, T_2 outputs ε; whenever the counter is unchanged, T_2 outputs c. Note that T_2 is also real-time, but not necessarily functional.

Let us now prove that A is language-bounded if and only if $T_1 \subseteq_o R(T_2)$ for some rational resynchronizer R.

Suppose first that A is language-bounded, with bound k. We obtain from this a k-delay resynchronizer R that reads a synchronized word $a_1 c^{i_1} \ldots a_n c^{i_n}$ of T_2, where $i_j \in \{0, 1, 2\}$ for all j. The resynchronizer R simulates a counter taking values in $[-k, k]$, and outputs $a_1 c \ldots a_n c$, accepting if and only if the counter is 0. Each time an $a_i c^2 a_j$ is encountered, it corresponds to an increment in the OCA; then R outputs $a_i c$, and the simulated counter decreases by 1 in R; likewise, each time an $a_i c a_j$ is encountered, R outputs $a_i c$ with no change in the simulated counter value, and finally, when two consecutive input symbols $a_i a_j$

are read by R, R outputs $a_i c$ and the simulated counter value increases by 1. Since the counter value is bounded by k in the OCA, the simulated counter in R is within $[-k, k]$. Clearly, $T_1 \subseteq_o R(T_2)$.

Conversely, suppose that $T_1 \subseteq_o R(T_2)$ for some rational resynchronizer R. We argue as before, using Theorem 8: we assume without loss of generality that R is a k-delay resynchronizer, for some k, and derive from this that A is language-bounded. ◀

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

Proof. The reduction is from the boundedness problem for multi-counter (Minksy) machines. Such a machine M can increment, decrement and test for zero. The question is whether there exists some bound k such that all computations of M (not necessarily accepting) from the initial configuration with all counters zero, have all counters stay below k. One can assume w.l.o.g. that if M is not bounded then for every k there is some initial run of M where *all* counters exceed k.

The OCA A reads sequences of transitions of M. At the beginning, A guesses a counter index j of M and starts simulating the sequence of transitions on counter j. If the sequence of transitions is incorrect because of counter j, the OCA accepts and stops after emptying counter j. Note that there are two types of error: either the counter is zero but should be decremented, or the counter is tested for zero, but is not zero. Both kinds of error can be checked by the OCA. Otherwise, if the simulation goes through for counter j, then the OCA accepts with empty counter at the end.

Assume that M is bounded, with bound k. If a sequence ρ of transitions is a run of M, then all simulations on any counter will be bounded by k. If ρ is not a run, then there is a first position of ρ where an error occurs, for instance because of counter j. Then the run of A simulating counter j will accept ρ within bound k.

If M is unbounded then for every k there is a run ρ where *all* counters exceed k. In this case all runs of A on ρ exceed k, so A is not language-bounded. ◀

▶ **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. The implications from 2. to 1. and from 3. to 2. are as in the proof of Theorem 4. The only interesting implication is from 1. to 3, where we suppose that $T_1 \subseteq T_2$ and we aim at constructing a 1-bounded Parikh resynchronizer R such that $T_1 =_o R(T_2)$, and with the same target set as T_1. The proof exploits some constructions based on crossing sequences, which are classically used to translate two-way automata to equivalent one-way automata [16], as well as to reduce containment of functional two-way transducers to emptiness of languages recognized by Parikh automata [15]. We briefly recall the key notions here, by adapting them in a way that is convenient for the presentation (notably, considering transitions instead of states).

A *crossing sequence* of a two-way automaton or a functional two-way transducer is a tuple $\bar{t} = (t_1, \dots, t_n)$ of transitions such that the source states of $t_1, t_3, \dots$ are right-reading and the source states of $t_2, t_4, \dots$ are left-reading. The tuple is meant to describe the transitions along a successful run that depart from configurations at a certain position y. Formally, given a run ρ, the crossing sequence of ρ at input position y, denoted $\rho[y]$, consists of the quadruples

(q, a, v, q') such that $(q, y) \xrightarrow{a \mid v} (q', y')$ is a transition of ρ, where the occurrence order on transitions induces a corresponding order on the quadruples of the crossing sequence. Without loss of generality, for two-way automata, as well as for functional two-way transducers, one can restrict to successful runs that never visit the same state twice at the same position. Accordingly, we can assume that the length of a crossing sequence never exceeds the total number of states of the device. Moreover, when the two-way automaton or transducer is unambiguous, the crossing sequences are uniquely determined by the input and the specific position in it. More precisely, there are regular languages $L_{\bar{t}}$, one for each possible crossing sequence, that contains precisely those inputs u with a specific position y marked on it (for short, we denote such words by $\langle u, y\rangle$), such that the crossing sequence at y of the unique successful run on u is precisely $\bar{t}$.

We now turn to the main proof, which is divided into several steps.

Encoding output positions.

We begin by describing a natural encoding of arbitrary output positions by means of their origins. Of course, the encoding depends on the given input, denoted u, and on the transducer we consider, either T_1 or T_2, which here is generically denoted by T. Now, let ρ be the unique successful run of T on u, and let G be the induced origin graph. To simplify the notations, hereafter we tacitly assume that T produces at most one letter at each transition — the assumption is without loss of generality, since long outputs originating at the same input position can be produced incrementally by exploiting two-way head motions. Let n be the number of states of T. Since T is unambiguous, G contains at most n output positions with the same origin (otherwise, the same configuration would be visited at least twice along the successful run ρ, which could then be used to contradict the assumption of unambiguity). This means that every position x in $\mathsf{out}(G)$ can be encoded by its origin $y_x = \mathsf{orig}(G)(x)$ together with a suitable index $i_x \in \{1, \dots, n\}$, describing the number of output positions $x' \leq x$ with the same origin y_x as x. Moreover, we recall that y_x can be represented as an annotated input of the form $\langle u, y_x\rangle$.

Decoding by Parikh automata.

We now show that there are Parikh automata that compute the inverse of the encoding $x \mapsto (y_x, i_x)$ described above. More precisely, there are unambiguous Parikh automata $A_1, \dots, A_n$ such that each A_i receives as input a word $\langle u, y\rangle$ having a special position marked on it, and outputs the unique output position x such that $(y, i) = (y_x, i_x)$, if this exists, otherwise the output is undefined. Each automaton A_i can be constructed from T and i by unambiguously guessing the crossing sequences of the unique run of T on u, and by counting the number of output symbols emitted until a *productive* transition at the marked position y is executed for the i-th time — a productive transition is a transition that produces non-empty output.

Redirecting origins.

We now apply the constructions outlined above in order to obtain the desired Parikh resynchronizer R from T_1 and T_2. Let u be some input and G_1, G_2 be the origin graphs induced by the unique successful runs of T_1, T_2 on u. Since $T_1 \subseteq T_2$, we can further let $v = \mathsf{out}(G_1) = \mathsf{out}(G_2)$. Consider any output position $x \in \mathsf{dom}(v)$. According to G_2, x is encoded by an input position y_x and an index $i_x \in \{1, \dots, n_2\}$, where n_2 is the number of states of T_2. In a similar way, according to G_1, the same position x is encoded by some input position z_x and an index $j_x \in \{1, \dots, n_1\}$, where n_1 is the number of states of T_1. Moreover, based on the previous constructions, there are unambiguous Parikh automata $A_{2,i}$ and $A_{1,j}$ such that

- $A_{2,i}(\langle u, y\rangle) = x$ if and only $(y, i) = (y_x, i_x)$,
- $A_{1,j}(\langle u, z\rangle) = x$ if and only $(z, j) = (z_x, j_x)$.

Since unambiguous Parikh automata are closed under pointwise difference, there is a unambiguous Parikh automaton $A_{i,j}$ that recognizes precisely the language of annotated words $\langle u, y, z\rangle$ such that

$$A_{2,i}(\langle u, y\rangle) - A_{1,j}(\langle u, z\rangle) = 0 \qquad (\star)$$

Note that the above language defines a partial bijection between pairs of positions y, z in the input u in such a way that y and z are the origins of the same output position x according to the unique origin graphs G_1, G_2 of T_1, T_2 such that $\mathsf{in}(G_1) = \mathsf{in}(G_2) = u$. This property can be used to define the component move_γ of the desired resynchronizer R, by simply letting

$$\mathsf{move}_\gamma = \{(u, y, z) \mid A_{i,j}(\langle u, y, z\rangle) = 0\}$$

where $\gamma = (a, i, j) \in \Gamma \times \{1, \dots, n_2\} \times \{1, \dots, n_1\}$.

For the correctness of the above definition we rely on guessing the correct pairs of indices (i, j) as annotations of output positions. More precisely, we have that:

- for every output position x with source origin $y = \mathsf{orig}(G_2)(x)$ and with label $\gamma = (a, i_x, j)$, there is at most one input position z such that $(u, y, z) \in \mathsf{move}_\gamma$; in addition, if we also have $j = j_x$, then $z = \mathsf{orig}(G_1)(x)$ is the target origin of x; symmetrically,
- for every output position x with target origin $z = \mathsf{orig}(G_1)(x)$ and with label $\gamma = (a, i, j_x)$, there is at most one input position y such that $(u, y, z) \in \mathsf{move}_\gamma$; in addition, if we also have $i = i_x$, then $y = \mathsf{orig}(G_2)(x)$ is the source origin of x.

Based on the above properties, we need to guess suitable output parameters that associate with each position x, a correct pair (i_x, j_x). We explain below how this is done using the components opar and $\mathsf{next}_{\gamma,\gamma'}$ of the resynchronizer.

Constraining output parameters.

We first focus on the indices j_x related to T_1; we will later explain how to adapt the constructions to check the indices i_x related to T_2. As usual, we fix an input u and the unique successful run ρ_1 of T_1 on u. The idea is that each index j_x corresponds to a certain element of the crossing sequence of ρ_1 at the target origin z_x, and knowing the correct index for x determines the correct index for the next output position $x+1$. Based on this, correctness can be verified inductively using the guessed crossing sequences and the relation $\mathsf{next}_{\gamma,\gamma'}$ of the resynchronizer, as follows. For the base case, we check that the first output position is correctly annotated with the index $j = 1$: this is readily done by a regular language opar.

For the inductive step, we consider an output position x and assume that it is correctly annotated with $j = j_x$. Let j' be the annotation of the next position $x+1$. To check that j' is also correct, we consider pairs of productive transitions in the crossing sequences associated with the target origins of x and $x+1$, and verify that they are connected by a non-productive run. More precisely, let z and z' be the target origins of x and $x+1$, respectively, and let $\bar{t}_z$ and $\bar{t}_{z'}$ be the crossing sequences of ρ_1 at those positions. We have that $j' = j_{x+1}$ if and only if the j-th productive transition of $\bar{t}_z$ and the j'-th productive transition of $\bar{t}_{z'}$ are connected by a factor of the run that consists only of non-productive transitions. The latter property can be translated to a regular property $\mathsf{next}_{\gamma,\gamma'}$ concerning the input annotated with two specific positions, z and z', assuming that $\gamma = (a, i, j)$ and $\gamma = (a', i', j')$ are the letters of the output positions x and $x+1$.

It now remains to check the correctness of the output annotations w.r.t. the indices i for the second transducer T_2. We follow a principle similar to the one described above for T_1. The only difference is that now, in the inductive step, we have work with the source origins y and y' of consecutive output positions x and $x+1$. The additional difficulty is that, by definition, the relation $\mathsf{next}_{\gamma,\gamma'}$ can only refer to target origins. We overcome this problem by exploiting the partial bijection between target and source origins, as defined by the relations move_γ and $\mathsf{move}_{\gamma'}$. Formally, we first define a relation $\mathsf{next}^{\text{source}}_{\gamma,\gamma'}$ as before, that constrain the indices i and i' associated with two consecutive output positions x and $x+1$ labeled by γ and γ', respectively. We do this as if $\mathsf{next}^{\text{source}}_{\gamma,\gamma'}$ were able to speak about source origins. We then intersect the following relation with the previously defined relation $\mathsf{next}_{\gamma,\gamma'}$:

$$\big\{(u, z, z') \;\mid\; \exists y, y' \; (u, y, y') \in \mathsf{next}^{\text{source}}_{\gamma,\gamma'},\; (u, y, z) \in \mathsf{move}_\gamma,\; (u, y', z') \in \mathsf{move}_{\gamma'}\big\}.$$

Since in the inductive step we assume that x is correctly annotated with the pair (i, j) and $x+1$ is annotated with (i', j'), where $j' = j_x$ is correct by the previous arguments, there are unique y, y' that satisfy $(u, y, z) \in \mathsf{move}_\gamma$ and $(u, y', z') \in \mathsf{move}_\gamma$ in the above definition, and these must be the source origins of x and $x+1$. This means that the above relation, which is definable by a unambiguous Parikh automaton, correctly verifies the correctness of the index i' associated with $x+1$.

We conclude by observing a few properties of the defined Parikh resynchronizer R. As already explained, the relation move_γ defines a bijection between pairs of input positions, so R is a 1-bounded Parikh resynchronizer. As concerns its target set, that is the set of pairs (u, z) such that $(u, y, z) \in \mathsf{move}_\gamma$ for some $z \in \mathsf{dom}(u)$ and some $\gamma \in \Gamma \times \{1, \dots, n_2\} \times \{1, \dots, n_1\}$, it coincides by construction with the target set of T_1. Finally, since the relation $\mathsf{next}_{\gamma,\gamma'}$ is defined by conjoining a regular property with the properties defined by the relations move_γ and $\mathsf{move}_{\gamma'}$, we have that $\mathsf{next}_{\gamma,\gamma'}$ is regular if move_γ and $\mathsf{move}_{\gamma'}$ are regular. ◀

▶ **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.*

Proof. We prove the following implications in the order: 1. → 2. → 3. → 4. → 1.

From 1. to 2.

This is trivial since $\hat{R}$ is bounded and satisfies $T_1 =_o \hat{R}(T_2)$, and hence $T_1 \subseteq_o \hat{R}(T_2)$.

From 2. to 3.

Let R be a k-bounded regular resynchronizer. The goal is to construct an equivalent 1-bounded regular resynchronizer R' (note that this part of the proof does not depend on T_1 and T_2). For this, we introduce a parameter $i_x \in \{1, \dots, k\}$ associated with each output position x, and require that for all output positions x, x' having the same label γ, and for all input positions y, z such that $(u, y, z) \in \mathsf{move}_\gamma$, if $i_x = i_{x'}$, then $x = x'$. The existence of such a mapping $x \mapsto i_x$ follows easily from the assumption that R is k-bounded. The relation

$\mathsf{move}'_{(\gamma,i)}$ of the new resynchronizer R' redirects origins of output positions based on their annotations $(\gamma,i) \in \Gamma \times \{1,\dots,k\}$, as follows:

$$\mathsf{move}'_{(\gamma,i)} = \big\{(w,y,z) \;\mid\; (w,y,z) \in \mathsf{move}_\gamma,\; \exists^{!i} y' \;\; y' < y \wedge (w,y',z) \in \mathsf{move}_\gamma\big\}$$

where $\exists^{!i} y'$ is an abbreviation for "there exist exactly i positions y' such that…". As for the relation $\mathsf{next}'_{(\gamma,i),(\gamma',j)}$, this coincides with $\mathsf{next}_{\gamma,\gamma'}$, so it does not take into account the new annotations. Thus, the defined resynchronizer R' is 1-bounded, regular, and defines the same resynchronization as R.

From 3. to 4.

Suppose that R is a 1-bounded regular resynchronizer with input alphabet Σ and output alphabet Γ, such that $T_1 \subseteq_o R(T_2)$. The goal is to construct a 1-bounded regular resynchronizer R' with the same target set as T_1 and such that $T_1 =_o R'(T_2)$. For the sake of simplicity, we assume that R has no input parameters, and similarly T_1 has no common guess (the more general cases can be dealt with by annotating the considered inputs with the possible parameters and the common guess). The idea for defining the desired resynchronizer R' is as follows. We first restrict each relation move_γ so as to make it a partial bijection, that is, for every input u, and every source origin $y \in \mathsf{dom}(u)$, there is an annotation w of the input and at most one target origin z that corresponds to y in $u \otimes w$ (and conversely, since R is 1-bounded, for every target origin z there is a unique source origin y that corresponds to z). This step requires the use of appropriate input parameters that determine a unique target origin z from any given source origin y. Then, we restrict further the relation move_γ so that every target origin z is witnessed by T_1. Formally, we introduce input parameters ranging over $\mathbb{B}^\Gamma$ and work with annotated inputs of the form $u \otimes w$, with $u \in \Sigma^*$ and $w \in (\mathbb{B}^\Gamma)^*$. Given $u \in \Sigma^*$, we define O_u as the set of all positions $z = \mathsf{orig}(G)(x)$ where G is an origin graph of T_1, $x \in \mathsf{dom}(\mathsf{out}(G))$, and $\mathsf{in}(G) = u$. The new relation move'_γ that redirects source origins to target origins is defined as the following restriction of move_γ:

$$\mathsf{move}'_\gamma = \big\{(u \otimes w, y, z) \;\mid\; (u,y,z) \in \mathsf{move}_\gamma,\; w(z)(\gamma) = 1,\; z \in O_u \;\big\}.$$

Clearly, the above relation is regular and contained in move_γ. However, it is still possible that move'_γ associates multiple target origins with the same source origin.

To get a partial bijection from move'_γ we need to constrain the possible annotated input $u \otimes w$. We do so by requiring that, for every output letter $\gamma \in \Gamma$ and every position y in $u \otimes w$, if there is z satisfying $(u,y,z) \in \mathsf{move}_\gamma$, then there is *exactly one* z' satisfying $(u,y,z') \in \mathsf{move}_\gamma$ and $w(\gamma)(z) = 1$. Note that the latter property is again regular, and thus could be conjoined with the original relation ipar to form the new relation ipar'. Accordingly, the relation $\mathsf{next}'_{\gamma,\gamma'}$ of the desired resynchronizer R' defines the same language as $\mathsf{next}_{\gamma,\gamma'}$, but expanded with arbitrary input annotations over $\mathbb{B}^\Gamma$.

It is now easy to see that the the resulting resynchronizer R' is 1-bounded, and in fact, on each input, defines a partial bijection between source and target origins in such a way that the target set coincides with that of T_1. By pairing this with the containments $R'(T_2) \subseteq_o R(T_2)$ and $T_1 \subseteq_o R(T_2)$, we obtain $T_1 =_o R'(T_2)$.

From 4. to 1.

Knowing that $\hat{R}(T_2) =_o T_1 =_o R(T_2)$ for two 1-bounded resynchronizers $R, \hat{R}$ with the same target sets as T_1 implies that the relations move_γ and move'_γ, from R and $\hat{R}$ respectively, coincide. Moreover, since the relation move_γ of R is assumed regular, this means that move'_γ is regular too. Finally, we recall that $\hat{R}$ is such that $\mathsf{next}'_{\gamma,\gamma'}$ is regular whenever move_γ and $\mathsf{move}_{\gamma'}$ are. We can then conclude that the relations $\mathsf{next}'_{\gamma,\gamma'}$ from $\hat{R}$ are also regular, and hence $\hat{R}$ is a regular resynchronizer. ◀