Origin-equivalence of two-way word transducers is in PSPACE

Sougata Bose, Anca Muscholl, Vincent Penelle
LaBRI, University of Bordeaux

Gabriele Puppis
CNRS, LaBRI

Abstract

We consider equivalence and containment problems for word transductions. These problems are known to be undecidable when the transductions are relations between words realized by non-deterministic transducers, and become decidable when restricting to functions from words to words. Here we prove that decidability can be equally recovered the *origin semantics*, that was introduced by Bojańczyk in 2014. We prove that the equivalence and containment problems for two-way word transducers in the origin semantics are PSPACE-complete. We also consider a variant of the containment problem where two-way transducers are compared under the origin semantics, but in a more relaxed way, by allowing distortions of the origins. The possible distortions are described by means of a *resynchronization* relation. We propose MSO-definable resynchronizers and show that they preserve the decidability of the containment problem under resynchronizations.

2012 ACM Subject Classification Theory of computation – Formal languages and automata theory – Automata extensions – Transducers

Keywords and phrases Transducers, origin semantics, equivalence

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.22

1 Introduction

Finite-state transducers over words were studied in computer science very early, at the same time as finite-state automata, see e.g. [19, 1, 12, 4]. A transducer defines a binary relation between words by associating an output with each transition. It is called functional if this relation is a partial function. Whereas the class of functions defined by one-way transducers, i.e. transducers that process their input from left to right, has been extensively considered in the past, the study of two-way transducers is quite recent. Connections to logic, notably through the notion of graph transformations definable in monadic second-order logic (MSO) [7, 13], have shown that the functions realized by two-way word transducers can be equally defined in terms of MSO transductions [14]. This result is reminiscent of Büchi-Elgot characterizations that hold for many classes of objects (words, trees, traces, etc). For this reason, transductions described by functional two-way word transducers are called *regular functions*. For a recent, nice survey on logical and algebraic properties of regular word transductions the reader is referred to [17].

Non-determinism is a very natural and desirable feature for most types of automata. However, for word transducers, non-determinism means less robustness. As an example, non-deterministic transducers are not equivalent anymore to NMSOT (the non-deterministic version of MSO transductions), however the latter is equivalent to non-deterministic streaming transducers [3]. A major problem is the undecidability of the equivalence of non-deterministic, one-way word transducers [18] (also called NGSM, and capturing the class of *rational relations*). In contrast, equivalence of functional, two-way word transducers is

decidable [9], even in PSPACE. This complexity is mainly based on the fact that it can be checked in PSPACE if two non-deterministic, two-way automata are equivalent (see e.g. [21]).

The equivalence test is one of the most widely used operation on automata, so that it becomes a natural question to know what is needed to recover decidability of equivalence for rational relations, and even for *regular relations*, which are transductions defined by non-deterministic, two-way transducers. The main result of our paper is that equivalence of non-deterministic, two-way transducers is decidable if one adopts a semantics based on origin information. According to this *origin semantics* [5], each letter of the output is tagged with the input position that generated it. Thus, a relation in the origin semantics becomes a relation over $\Sigma^* \times (\Gamma \times \mathbb{N})^*$. Surprisingly, the complexity of the equivalence test turns out to be as low as it could be, namely in PSPACE (thus PSPACE-complete for obvious reasons).

As a second result, we introduce a class of MSO-definable resynchronizations for regular relations. A resynchronization [15] is a binary relation over $\Sigma^* \times (\Gamma \times \mathbb{N})^*$ that preserves the input and the output (i.e., the fist two components), but can change the origins (i.e., the third component). Resynchronizations allow to compare transducers under the origin semantics in a more relaxed way, by allowing distortions of origins. Formally, given two non-deterministic, two-way transducers $\mathcal{T}_1, \mathcal{T}_2$, and a resynchronization R, we want to compare $\mathcal{T}_1, \mathcal{T}_2$ under the origin semantics modulo R. Containment of $\mathcal{T}_1$ in $\mathcal{T}_2$ modulo R means that for each tagged input/output pair σ' generated by $\mathcal{T}_1$, there should be some tagged input/output pair σ generated by $\mathcal{T}_2$ such that $(\sigma, \sigma') \in R$. In other words, the resynchronization R describes possible distortions of origin, and we ask whether $\mathcal{T}_1$ is contained in $R(\mathcal{T}_2)$. The resynchronizations defined here correspond to MSO formulas that describe the change of origin by mainly considering the input (and to some small extent, the output). The containment problem under such resynchronizations turns to be undecidable, unless we enforce some restrictions. It is decidable for those resynchronizations R that use formulas satisfying a certain (decidable) 'boundedness' property. In addition, if R is fixed, then the containment problem modulo R is solvable in PSPACE, thus with the same complexity as the origin-equivalence problem. This is shown by providing a two-way transducer $\mathcal{T}_2'$ that is equivalent to $R(\mathcal{T}_2)$ (we say that $R(\mathcal{T}_2)$ is realizable by a transducer), and then we check containment of $\mathcal{T}_1$ in $\mathcal{T}_2'$ using our first algorithm. We conjecture that our class of resynchronizations captures the rational resynchronizations of [15], but we leave this for future work.

Related work and discussion. The origin semantics for transducers has been introduced in [5], and was shown to enjoy several nice properties, in particular a Myhill-Nerode characterization that can be used to decide the membership problem for subclasses of transductions, like first-order definable ones. The current state of the art counts quite a number of results related to the origin semantics of transducers. In [6] a characterization of the class of origin graphs generated by (functional) streaming transducers is given (the latter models were studied in [2, 3]). Decision problems for tree transducers under the origin semantics have been considered in [16], where it is shown that origin-equivalence of top-down tree transducers is decidable. Note that top-down transducers correspond on words to one-way transducers, so the result of [16] is incomparable with ours.

As mentioned before, the idea of resynchronizing origins of word transducers has been introduced by Filiot et al. in [15] for the case of one-way transducers. Rational resynchronizers as defined in [15] are one-way transducers $\mathcal{R}$ that read sequences of the form $u_1v_1u_2v_2 \cdots u_nv_n$, where $u_1 \cdots u_n$ represents the input and $v_1 \cdots v_n$ the output, with the origin of v_i being the last letter of u_i. It is required that any image of $u_1v_1u_2v_2 \cdots u_nv_n$ through $\mathcal{R}$ leaves the input and the output part unchanged, thus only origins change. The

definition of resynchronizer in the one-way case is natural. However, it cannot be extended to two-way transducers, since there is no word encoding of tagged input-output pairs realized by arbitrary two-way transducers. Our approach is logic-based: we define MSO resynchronizations that refer to origin graphs. More precisely, our MSO resynchronizations are formulas that talk about the input and, to a limited extent, about the tagged output.

Overview. After introducing the basic definitions and notations in Section 2, we present the main result about the equivalence problem in Section 3. Resynchronizations are considered in Section 4. A full version of the paper is available at `https://arxiv.org/abs/1807.08053`.

2 Preliminaries

Given a word $w = a_1 \dots a_n$, we denote by $\mathrm{dom}(w) = \{1, \dots, n\}$ its domain, and by $w(i)$ its i-th letter, for any $i \in \mathrm{dom}(w)$.

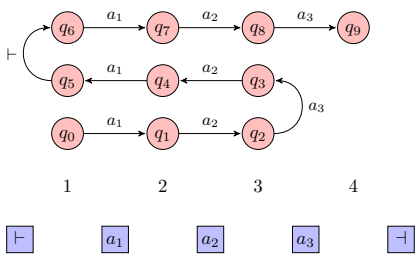
Automata. To define two-way automata, and later two-way transducers, it is convenient to adopt the convention that, for any given input $w \in \Sigma^*$, $w(0) = \vdash$ and $w(|w|+1) = \dashv$, where $\vdash, \dashv \notin \Sigma$ are special markers used as delimiters of the input. In this way an automaton can detect when an endpoint of the input has been reached and avoid moving the head outside.

A *two-way automaton* (*2NFA* for short) is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, where Σ is the input alphabet, $Q = Q_\prec \uplus Q_\succ$ is the set of states, partitioned into a set $Q_\prec$ of left-reading states and a set $Q_\succ$ of right-reading states, $I \subseteq Q_\succ$ is the set of initial states, $F \subseteq Q$ is the set of final states, and $\Delta \subseteq Q \times (\Sigma \uplus \{\vdash, \dashv\}) \times Q \times \{\mathsf{left}, \mathsf{right}\}$ is the transition relation. The partitioning of the set of states is useful for specifying which letter is read from each state: left-reading states read the letter to the left, whereas right-reading states read the letter to the right. A transition $(q, a, q', d) \in \Delta$ is *leftward* (resp. *rightward*) if $d = \mathsf{left}$ (resp. $d = \mathsf{right}$). Of course, we assume that no leftward transition is possible when reading the left marker $\vdash$, and no rightward transition is possible when reading the right marker $\dashv$. We further restrict Δ by asking that $(q, a, q', \mathsf{left}) \in \Delta$ implies $q' \in Q_\prec$, and $(q, a, q', \mathsf{right}) \in \Delta$ implies $q' \in Q_\succ$.

To define runs of 2NFA we need to first introduce the notion of configuration. Given a 2NFA $\mathcal{A}$ and a word $w \in \Sigma^*$, a *configuration* of $\mathcal{A}$ on w is a pair (q, i), with $q \in Q$ and $i \in \{1, \dots, |w|+1\}$. Such a configuration represents the fact that the automaton is in state q and its head is *between* the $(i-1)$-th and the i-th letter of w (recall that we are assuming $w(0) = \vdash$ and $w(|w|+1) = \dashv$). The transitions that depart from a configuration (q, i) and read a are denoted $(q, i) \xrightarrow{a} (q', i')$, and must satisfy one of the following conditions:

- $q \in Q_\succ$, $a = w(i)$, $(q, a, q', \mathsf{right}) \in \Delta$, and $i' = i + 1$,
- $q \in Q_\succ$, $a = w(i)$, $(q, a, q', \mathsf{left}) \in \Delta$, and $i' = i$,
- $q \in Q_\prec$, $a = w(i-1)$, $(q, a, q', \mathsf{right}) \in \Delta$, and $i' = i$,
- $q \in Q_\prec$, $a = w(i-1)$, $(q, a, q', \mathsf{left}) \in \Delta$, and $i' = i - 1$.

A configuration (q, i) on w is *initial* (resp. *final*) if $q \in I$ and $i = 1$ (resp. $q \in F$ and $i = |w| + 1$). A *run* of $\mathcal{A}$ on w is a sequence $\rho = (q_1, i_1) \xrightarrow{b_1} (q_2, i_2) \xrightarrow{b_2} \cdots \xrightarrow{b_m} (q_{m+1}, i_{m+1})$ of configurations connected by transitions. The figure to the right depicts an input $w = a_1 a_2 a_3$ (in blue) and a possible run on it (in red), where $q_0, q_1, q_2, q_6, q_7, q_8 \in Q_\succ$ and $q_3, q_4, q_5 \in Q_\prec$, and



$1, 2, 3, 4$ are the positions associated with the various configurations. A run is *successful* if it starts with an initial configuration and ends with a final configuration. The *language* of $\mathcal{A}$ is the set $[\![\mathcal{A}]\!] \subseteq \Sigma^*$ of all words on which $\mathcal{A}$ has a successful run.

When $\mathcal{A}$ has only right-reading states (i.e. $Q_\prec = \emptyset$) and rightward transitions, we say that $\mathcal{A}$ is a *one-way automaton* (*NFA* for short).

Transducers. Two-way transducers are defined similarly to two-way automata, by introducing an output alphabet Γ and associating an output from Γ^* with each transition rule. So a *two-way transducer* (*2NFT* for short) $\mathcal{T} = (Q, \Sigma, \Gamma, \Delta, I, F)$ is basically a 2NFA as above, but with a transition relation $\Delta \subseteq Q \times (\Sigma \uplus \{\vdash, \dashv\}) \times \Gamma^* \times Q \times \{\mathsf{left}, \mathsf{right}\}$. A transition is usually denoted by $(q, i) \xrightarrow{a|v} (q', i')$, and describes a move of the transducer from configuration (q, i) to configuration (q', i') that reads the input letter a and outputs the word v. The same restrictions and conventions for two-way automata apply to the transitions of two-way transducers, and configurations and runs are defined in a similar way.

The *output* associated with a successful run $\rho = (q_1, i_1) \xrightarrow{b_1|v_1} (q_2, i_2) \xrightarrow{b_2|v_2} \cdots \xrightarrow{b_m|v_m} (q_{m+1}, i_{m+1})$ is the word $v_1 v_2 \cdots v_m \in \Gamma^*$. A two-way transducer $\mathcal{T}$ defines a relation $[\![\mathcal{T}]\!] \subseteq \Sigma^* \times \Gamma^*$ consisting of all the pairs (u, v) such that v is the output of some successful run ρ of $\mathcal{T}$ on u. Throughout the paper, a transducer is non-deterministic and two-way.

Origin semantics. In the origin semantics for transducers [5], the output is tagged with information about the position of the input where it was produced. If reading the i-th letter of the input we output v, then all letters of v are tagged with i, and we say they have *origin* i. We use the notation (v, i) to denote that all positions in v have origin i. The outputs associated with a successful run $\rho = (q_1, i_1) \xrightarrow{b_1|v_1} (q_2, i_2) \xrightarrow{b_2|v_2} (q_3, i_3) \cdots \xrightarrow{b_m|v_m} (q_{m+1}, i_{m+1})$ in the origin semantics are the words of the form $\nu = (v_1, j_1)(v_2, j_2) \cdots (v_m, j_m)$ over $\Gamma \times \mathbb{N}$, where, for all $1 \leq k \leq m$, $j_k = i_k$ if $q_k \in Q_\succ$, and $j_k = i_k - 1$ if $q_k \in Q_\prec$. Under the origin semantics, the relation defined by $\mathcal{T}$, denoted $[\![\mathcal{T}]\!]_o$, is the set of pairs $\sigma = (u, \nu)$ —called *synchronized pairs*— such that $u \in \Sigma^*$ and $\nu \in (\Gamma \times \mathbb{N})^*$ is the output of some successful run on u. Take as example the 2NFA run depicted in the previous figure, and assume that any transition on a letter $a \in \Sigma$ outputs a, while a transition on a marker $\vdash$ or $\dashv$ outputs the empty word ε. The output associated with that run in the origin semantics is $(a_1, 1)(a_2, 2)(a_3, 3)(a_2, 2)(a_1, 1)(a_1, 1)(a_2, 2)(a_3, 3)$.

Given two transducers $\mathcal{T}_1, \mathcal{T}_2$, we say they are *origin-equivalent* if $[\![\mathcal{T}_1]\!]_o = [\![\mathcal{T}_2]\!]_o$. Note that two transducers $\mathcal{T}_1, \mathcal{T}_2$ can be equivalent in the classical semantics, i.e. $[\![\mathcal{T}_1]\!] = [\![\mathcal{T}_2]\!]$, while they can have different origin semantics, so $[\![\mathcal{T}_1]\!]_o \neq [\![\mathcal{T}_2]\!]_o$.

Regular outputs. The transducers we defined just above consume input letters while outputting strings of bounded length. In order to perform some crucial constructions later —notably, to shortcut factors of runs with empty output— we need to slightly generalize the notion of output associated with a transition, so as to allow producing arbitrarily long words on reading a single letter. Formally, the transition relation of a transducer *with regular outputs* is allowed to be any subset Δ of $Q \times (\Sigma \uplus \{\vdash, \dashv\}) \times \mathbf{2}^{\mathbf{\Gamma^*}} \times Q \times \{\mathsf{left}, \mathsf{right}\}$ such that, for all $q, q' \in Q$, $a \in \Sigma$, $d \in \{\mathsf{left}, \mathsf{right}\}$, there is at most one language $L \subseteq \Gamma^*$ such that $(q, a, L, q', d) \in \Delta$; moreover, this language L must be non-empty and regular. A transition $(q, i) \xrightarrow{a|L} (q', i')$ means that the transducer can move from configuration (q, i) to configuration (q', i') while reading a and outputting any word $v \in L$. Accordingly, the outputs that are associated with a successful run $\rho = (q_1, i_1) \xrightarrow{b_1|L_1} (q_2, i_2) \xrightarrow{b_2|L_2} (q_3, i_3) \cdots \xrightarrow{b_n|L_m} (q_{m+1}, i_{m+1})$ in the origin semantics are the words of the form $\nu = (v_1, j_1)(v_2, j_2) \cdots (v_m, j_m)$, where $v_k \in L_k$ and $j_k = i_k$ or $j_k = i_{k-1}$ depending on whether $q_k \in Q_\succ$ or $q_k \in Q_\prec$. We say that two runs ρ_1, ρ_2 are *origin-equivalent* if they have the same sets of associated outputs. Clearly, the extension with regular outputs is only syntactical,

and it preserves the expressiveness of the class of transducers we consider. In the remaining of the paper, we will tacitly refer to above notion of transducer.

PSpace-constructibility. As usual, we call the *size* of an automaton or a transducer the number of its states, input symbols, transitions, plus, if present, the sizes of the NFA descriptions of the regular output languages associated with each transition rule.

In our complexity analysis, however, we will often need to work with online presentations of automata and transducers. For example, we may say that an automaton or a transducer can be computed using a polynomial amount of working space (at thus its size would be at most exponential) w.r.t. a given input. The terminology introduced below will be extensively used throughout the paper to describe the computational complexity of an automaton, a transducer, or a part of it, in terms of a specific parameter.

Given a parameter $n \in \mathbb{N}$, we say that an automaton or a transducer has PSpace-*constructible transitions w.r.t.* n if its transition relation can be enumerated by an algorithm that uses working space polynomial in n, and in addition, when the device is a transducer, every transition has at most polynomial size in n. In particular, if a transducer has PSpace-constructible transitions, then the size of the NFA representing every output language is polynomial. Similarly, we say that an automaton is PSpace-*constructible w.r.t.* n if all its components, —the alphabets, the state set, the transition relation, etc.— are enumerable by algorithms that use space polynomial in n.

3 Equivalence of transducers with origins

We focus on the equivalence problem for two-way transducers. In the classical semantics, this problem is known to be undecidable even if transducers are one-way [18] (called NGSM in the latter paper). We consider this problem in the origin semantics. We will show that, in this setting, equivalence becomes decidable, and can even be solved in PSpace – so with no more cost than equivalence of non-deterministic two-way automata:

▶ **Theorem 1.** *Containment and equivalence of two-way transducers under the origin semantics is* PSpace-*complete.*

The proof of this result is quite technical. As a preparation, we first show how to check origin-equivalence for transducers in which all transitions produce non-empty outputs, namely, where the transition relation is of the form $\Delta \subseteq Q \times (\Sigma \uplus \{\vdash, \dashv\}) \times \mathbf{2}^{\Gamma+} \times Q \times \{\mathsf{left}, \mathsf{right}\}$. We call *busy* any such transducer.

3.1 Origin-equivalence of busy transducers

An important feature of our definition of transducers is that, along any possible run, an input position is never read twice in a row. In other words, our transducers do not have "stay" transitions. For a busy transducer, this implies that the origins of outputs of consecutive transitions are always different. As a consequence, runs of two busy transducers can be only origin-equivalent if they visit the same sequences of positions of the input and have the same possible outputs transition-wise. To give the intuition, we note that origin-equivalence of busy classical transducers, with single output words associated with transitions, can be reduced to a version of equivalence of 2NFA, where we ask that runs have the same shape. Already the last condition does not allow to apply the PSpace algorithm for equivalence of 2NFA of [21], and the naive algorithm would be of exponential time. Some more complications arise when we assume regular outputs, which will be required when we shall deal with

non-busy transducers. We introduce now the key notions of transition shape and witness procedure.

Let $\mathcal{T}_1, \mathcal{T}_2$ be busy transducers over the same input alphabet, with $\mathcal{T}_i = (Q_i, \Sigma, \Gamma_i, \Delta_i, I_i, F_i)$ for $i = 1, 2$. We say that two transitions $t_1 \in \Delta_1$ and $t_2 \in \Delta_2$, with $t_i = (q_i, a_i, q'_i, L_i, d_i)$, have the *same shape* if $a_1 = a_2$, $q_1 \in Q_{1,\prec} \Leftrightarrow q_2 \in Q_{2,\prec}$, and $q'_1 \in Q_{1,\prec} \Leftrightarrow q'_2 \in Q_{2,\prec}$ (and hence $d_1 = d_2$).

We assume that there is a non-deterministic procedure $\mathcal{W}$, called *witness procedure*, that does the following. Given a transition $t_1 = (q_1, a_1, q'_1, L_1, d_1)$ of $\mathcal{T}_1$, $\mathcal{W}$ returns a set $X \subseteq \Delta_2$ of transitions of $\mathcal{T}_2$ satisfying the following property: *for some word* $v \in L_1$, we have

$$X \;=\; \big\{t_2 = (p_2, a_2, q_2, L_2, d_2) \in \Delta_2 \;:\; v \in L_2, \text{ and } t_2 \text{ has } \textit{same shape} \text{ as } t_1\big\}.$$

Note that $\mathcal{W}$ is non-deterministic: it can return several sets based on the choice of v. If $t_1 \in \Delta_1$ and X is a set that could be returned by $\mathcal{W}$ on t_1, we write $X \in \mathcal{W}(t_1)$. However, if $\mathcal{T}_1$ and $\mathcal{T}_2$ were classical transducers, specifying a single output word for each transition, then $\mathcal{W}$ could return only one set on $t_1 \in \Delta_1$, that is, the set of transitions of $\mathcal{T}_2$ with the same shape and the same output as t_1.

The intuition behind the procedure $\mathcal{W}$ is the following. Consider a successful run ρ_1 of $\mathcal{T}_1$ on u. Since $\mathcal{T}_1, \mathcal{T}_2$ are both busy, $[\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2]\!]_o$ necessarily means that for all possible outputs produced by the transitions of ρ_1, there is some run ρ_2 of $\mathcal{T}_2$ on u that has the same shape as ρ_1 and the same outputs, transition-wise. Procedure $\mathcal{W}$ will precisely provide, for each transition t_1 of $\mathcal{T}_1$ with output language $L \subseteq \Gamma^+$, and for each choice of $v \in L$, the set of all transitions of $\mathcal{T}_2$ with the same shape as t and that could produce the same output v.

We introduce a last piece of terminology. Given a run $\rho_1 = t_1 \dots t_m$ of $\mathcal{T}_1$ of length m and a sequence $\xi = X_1, \dots, X_m$ of subsets of Δ_2 (*witness sequence*), we write $\xi \in \mathcal{W}(\rho_1)$ whenever $X_i \in \mathcal{W}(t_i)$ for all $1 \leq i \leq m$. We say that a run $\rho_2 = t'_1 \dots t'_m$ of $\mathcal{T}_2$ is *ξ-compatible* if $t'_i \in X_i$ for all $1 \leq i \leq m$. The following result (proved in the appendix) is crucial:

▶ **Proposition 2.** *Let $\mathcal{T}_1, \mathcal{T}_2$ be two busy transducers over the same input alphabet, and $\mathcal{W}$ a witness procedure. Then $[\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2]\!]_o$ if and only if for every successful run ρ_1 of $\mathcal{T}_1$, and for every witness sequence $\xi \in \mathcal{W}(\rho_1)$, there is a successful run ρ_2 of $\mathcal{T}_2$ which is ξ-compatible.*

Next, we reduce the problem $[\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2]\!]_o$ to the emptiness problem of a one-way automaton (NFA) $\mathcal{B}$. In this reduction, the NFA $\mathcal{B}$ can be exponentially larger than $\mathcal{T}_1, \mathcal{T}_2$, but is PSPACE-constructible under suitable assumptions on $\mathcal{T}_1, \mathcal{T}_2$, and $\mathcal{W}$.

▶ **Lemma 3.** *Given two busy transducers $\mathcal{T}_1, \mathcal{T}_2$ with input alphabet Σ and a witness procedure $\mathcal{W}$, one can construct an NFA $\mathcal{B}$ that accepts precisely the words $u \in \Sigma^*$ for which there exist a successful run ρ_1 of $\mathcal{T}_1$ on u and a witness sequence $\xi \in \mathcal{W}(\rho_1)$ such that no ξ-compatible run ρ_2 of $\mathcal{T}_2$ is successful.*

Moreover, if $\mathcal{T}_1, \mathcal{T}_2$ have a total number n of states and PSPACE*-constructible transitions w.r.t. n, and $\mathcal{W}$ uses space polynomial in n, then $\mathcal{B}$ is* PSPACE*-constructible w.r.t. n.*

The proof of the lemma (in the appendix) is based on variants of the classical techniques of subset construction and crossing sequences [20, 21]. Below, we state an immediate consequence of the previous proposition and lemma:

▶ **Corollary 4.** *Given two busy transducers $\mathcal{T}_1, \mathcal{T}_2$ with a total number n of states, and given a witness procedure that uses space polynomial in n, the problem of deciding $[\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2]\!]_o$ is in* PSPACE *w.r.t. n.*

3.2 Origin-equivalence of arbitrary transducers

We now consider transducers that are not necessarily busy. To show that origin-equivalence remains decidable in PSPACE, we will modify the transducers so as to make them busy, and reduce in this way the origin equivalence problem to the case treated in Section 3.1.

A naive idea would be to modify the transitions that output the empty word ε and make them output a special letter $\#$. This however would not give a correct reduction towards origin-equivalence with busy transducers. Indeed, a transducer may produce non-empty outputs, say $v_1, v_2, \ldots$, with transitions that occur at the same position, say i, and that are interleaved by runs traversing other positions of the input but producing only ε. In that case, we would still need to compare where the words $v_1, v_2, \ldots$ were produced, and see that they may form a contiguous part of the output with origin i. The above idea is however useful if we first *normalize* our transducers in such a way that maximal subruns generating empty outputs are unidirectional. Paired with the fact that the same input position is never visited twice on two consecutive transitions, this will give the following characterization: two arbitrary transducers are origin-equivalent if and only if their normalized versions, with empty outputs replaced by $\#$, are also origin-equivalent.

For simplicity, we fix a single transducer $\mathcal{T} = (Q, \Sigma, \Gamma, \Delta, I, F)$, which could be thought of as any of the two transducers $\mathcal{T}_1, \mathcal{T}_2$ that are tested for origin-equivalence. To normalize $\mathcal{T}$ we consider runs that start and end in the same position, and that produce empty output. Such runs are called lazy U-turns and are formally defined below. We will then abstract lazy U-turns by pairs of states, called U-pairs for short.

▶ **Definition 5.** Given an input word u, a *left* (resp. *right*) *lazy U-turn at position i of u* is any run of $\mathcal{T}$ on u of the form $(q_1, i_1) \xrightarrow{b_1|v_1} (q_2, i_2) \xrightarrow{b_2|v_2} \cdots \xrightarrow{b_m|v_m} (q_{m+1}, i_{m+1})$, with $i_1 = i_{m+1} = i$, $i_k < i$ (resp. $i_k > i$) for all $2 \le k \le m$, and $v_k = \varepsilon$ for all $1 \le k \le m$.

For brevity, we shall often refer to a *left/right lazy U-turn* without specifying the position i and the word u, assuming that these are clear from the context.

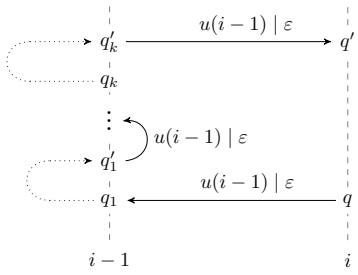
The pair (q_1, q_{m+1}) of states at the extremities of a left/right lazy U-turn is called a *left/right U-pair* (*at position i of u*). We denote by $U_i^{\circlearrowright}$ (resp. $U_i^{\circlearrowleft}$) the set of all left (resp. right) U-pairs at position i (again, the input u is omitted from the notation as it usually understood from the context).

Note that we have $U_i^{\circlearrowright} \subseteq Q_\prec \times Q_\succ$ and $U_i^{\circlearrowleft} \subseteq Q_\succ \times Q_\prec$. Accordingly, we define the word $u^{\circlearrowright}$ over $2^{Q_\prec \times Q_\succ}$ that has the same length as u and labels every position i with the set $U_i^{\circlearrowright}$ of left U-pairs. This $u^{\circlearrowright}$ is seen as an annotation of the original input u, and can be computed from $\mathcal{T} = (Q, \Sigma, \Gamma, \Delta, I, F)$ and u using the following recursive rule:

$$(q, q') \in u^{\circlearrowright}(i) \quad \text{if and only if} \quad q \in Q_\prec \ \wedge\ \big(q, u(i-1), \varepsilon, q', \mathsf{right}\big) \in \Delta$$

$$\text{or} \quad q \in Q_\prec \ \wedge\ \exists\, (q_1, q_1'), \ldots, (q_k, q_k') \in u^{\circlearrowright}(i-1)$$

$$\begin{cases} \big(q, u(i-1), \varepsilon, q_1, \mathsf{left}\big) \in \Delta \\ \big(q_j', u(i-1), \varepsilon, q_{j+1}, \mathsf{left}\big) \in \Delta \quad \forall 1 \le j \le k \\ \big(q_k', u(i-1), \varepsilon, q_k, \mathsf{right}\big) \in \Delta. \end{cases}$$



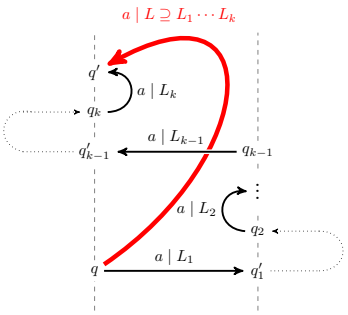
The annotation $u^{\circlearrowleft}$ with right U-pairs satisfies a symmetric recursive rule.

Of course, the annotations $u^{\circlearrowright}, u^{\circlearrowleft}$ are over alphabets of exponential size. This does not raise particular problems concerning the complexity, since we aim at deciding origin-equivalence in PSPACE, and towards this goal we could work with automata and transducers that have PSPACE-constructible transitions. In particular, we can use the recursive rules for $u^{\circlearrowright}$ and $u^{\circlearrowleft}$ to get the following straightforward lemma (proof omitted):

▶ **Lemma 6.** *Given a transducer $\mathcal{T}$, one can compute an NFA $\mathcal{U}$ that has the same number of states as $\mathcal{T}$ and such that $[\![\mathcal{U}]\!] = \{u \otimes u^{\circlearrowright} \otimes u^{\circlearrowleft} : u \in \Sigma^*\}$, where $\otimes$ denotes the convolution of words of the same length. The NFA $\mathcal{U}$ is* PSPACE*-constructible w.r.t. the size of $\mathcal{T}$.*

To normalize the transducer $\mathcal{T}$ it is convenient to assume that the sets of left and right U-pairs can be read directly from the input (we refer to this as *annotated input*). For this, we introduce the transducer $\mathcal{T}_U$, that is obtained from $\mathcal{T}$ by extending its input alphabet from Σ to $\Sigma \times 2^{Q_\prec \times Q_\succ} \times 2^{Q_\succ \times Q_\prec}$, and by modifying the transitions in the obvious way, that is, from (q, a, L, q', d) to $\big(q, (a, U^{\circlearrowright}, U^{\circlearrowleft}), L, q', d\big)$ for any $U^{\circlearrowright} \subseteq Q_\prec \times Q_\succ$ and $U^{\circlearrowleft} \subseteq Q_\succ \times Q_\prec$. Note that $\mathcal{T}_U$ does not check that the input is correctly annotated, i.e. of the form $u \otimes u^{\circlearrowright} \otimes u^{\circlearrowleft}$ (this is done by the NFA $\mathcal{U}$). Further note that $\mathcal{T}_U$ has exponential size w.r.t. $\mathcal{T}$, but its state space remains the same as $\mathcal{T}$, and its transitions can be enumerated in PSPACE.

We are now ready to describe the normalization of $\mathcal{T}_U$, which produces an origin-equivalent transducer $\mathsf{Norm}(\mathcal{T}_U)$ with no lazy U-turns. The transducer $\mathsf{Norm}(\mathcal{T}_U)$ is obtained in two steps. First, using the information provided by the annotation of the input, we shortcut all runs of $\mathcal{T}_U$ that consist of multiple transitions outputting at the same position and interleaved by lazy U-turns. The resulting transducer is denoted $\mathsf{Shortcut}(\mathcal{T}_U)$. After this step, we will eliminate the lazy U-turns, thus obtaining $\mathsf{Norm}(\mathcal{T}_U)$. Formally, $\mathsf{Shortcut}(\mathcal{T}_U)$ has for transitions the tuples of the form $\big(q, (a, U^{\circlearrowright}, U^{\circlearrowleft}), L, q', d\big)$, where L is the smallest language that contains every language of the form $L_1 \cdot L_2 \cdots L_k$ for which there are $q_1, q'_1, \ldots, q_k, q'_k$ and $d_1, \ldots, d_k$, with $q = q_1$, $q'_k = q'$ (and hence $d_k = d$), $(q_i, a, L_i, q'_i, d_i) \in \Delta$, and $(q'_i, q_{i+1}) \in U^{\circlearrowright} \cup U^{\circlearrowleft}$ for all i (see the figure to the right). Note that there is no transition $\big(q, (a, U^{\circlearrowright}, U^{\circlearrowleft}), L, q', d\big)$ when there are no languages $L_1, L_2, \ldots, L_k$ as above. The output languages associated with the transitions of $\mathsf{Shortcut}(\mathcal{T}_U)$ can be constructed using a classical saturation mechanism, which is omitted here, they are regular, and their NFA representations are polynomial-sized w.r.t. the size of the NFA representations of the output languages of $\mathcal{T}_U$. This implies that $\mathsf{Shortcut}(\mathcal{T}_U)$ has PSPACE-constructible transitions w.r.t. the number of its states, exactly like $\mathcal{T}_U$.



Recall that two runs are *origin-equivalent* if they produce the same synchronized pairs. The next lemma shows that, on correctly annotated inputs, $\mathsf{Shortcut}(\mathcal{T}_U)$ is origin-equivalent to $\mathcal{T}_U$, even when avoiding U-turns. The proof of the lemma is in the appendix.

▶ **Lemma 7.** *Let $w = u \otimes u^{\circlearrowright} \otimes u^{\circlearrowleft}$ be an arbitrary input annotated with left and right U-pairs. Every successful run of $\mathcal{T}_U$ on w is origin-equivalent to some successful run of $\mathsf{Shortcut}(\mathcal{T}_U)$ on w without lazy U-turns. Conversely, every successful run of $\mathsf{Shortcut}(\mathcal{T}_U)$ on w is origin-equivalent to some successful run of $\mathcal{T}_U$ on w.*

The next step of the normalization consists of restricting the runs of $\mathsf{Shortcut}(\mathcal{T}_U)$ so as to avoid any lazy U-turn. This is done by simply forbidding the shortest possible lazy U-turns, namely, the transitions that output ε and that remain on the same input position. On the

one hand, since every lazy U-turn contains a transition of the previous form, forbidding this type of transitions results in forbidding arbitrary lazy U-turns. On the other hand, thanks to Lemma 7, this will not affect the semantics of $\mathsf{Shortcut}(\mathcal{T}_U)$. Formally, we construct from $\mathsf{Shortcut}(\mathcal{T}_U)$ a new transducer $\mathsf{Norm}(\mathcal{T}_U)$ by replacing every transition rule (q, a, L, q', d) with (q, a, L', q', d), where L' is either L or $L \setminus \{\varepsilon\}$, depending on whether $q \in Q_\prec \Leftrightarrow q' \in Q_\prec$ or not. We observe that $\mathsf{Norm}(\mathcal{T}_U)$ has the same set of states as the original transducer $\mathcal{T}$, and PSPACE-constructible transitions w.r.t. the size of $\mathcal{T}$. The proof of the following result is straightforward and thus omitted.

▶ **Lemma 8.** *$\mathcal{T}_U$ and $\mathsf{Norm}(\mathcal{T}_U)$ are origin-equivalent when restricted to correctly annotated inputs, i.e.: $[\![\mathcal{T}_U]\!]_o \cap R = [\![\mathsf{Norm}(\mathcal{T}_U)]\!]_o \cap R$, where $R = [\![\mathcal{U}]\!] \times (\Gamma \times \mathbb{N})^*$.*

We finally come to the last step of the reduction. This amounts to consider two normalized transducers $\mathsf{Norm}(\mathcal{T}_{1,U})$ and $\mathsf{Norm}(\mathcal{T}_{2,U})$ that read inputs annotated with the left/right U-pairs of both $\mathcal{T}_1$ and $\mathcal{T}_2$, and replacing, in their transitions, the empty output by a special letter $\# \notin \Gamma$. Formally, in the transition relation of $\mathsf{Norm}(\mathcal{T}_{i,U})$, for $i = 1, 2$, we replace every tuple (q, a, L, q', d), where q is left-reading iff q' is left-reading, with the tuple (q, a, L', q', d), where L' is either $(L \setminus \{\varepsilon\}) \cup \{\#\}$ or L, depending on whether $\varepsilon \in L$ or not. The transducer obtained in this way is *busy*, and is thus called $\mathsf{Busy}(\mathcal{T}_{i,U})$. Moreover, the states of $\mathsf{Busy}(\mathcal{T}_{i,U})$ are the same as those of $\mathcal{T}_i$, and its transitions are PSPACE-constructible. The proposition below follows immediately from the previous arguments, and reduces origin-containment between $\mathcal{T}_1$ and $\mathcal{T}_2$ to an origin-containment between $\mathsf{Busy}(\mathcal{T}_{1,U})$ and $\mathsf{Busy}(\mathcal{T}_{2,U})$, but relativized to correctly annotated inputs.

▶ **Proposition 9.** *Given two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$,*

$$[\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2]\!]_o \quad \text{if and only if} \quad [\![\mathsf{Busy}(\mathcal{T}_{1,U})]\!]_o \cap R \subseteq [\![\mathsf{Busy}(\mathcal{T}_{2,U})]\!]_o \cap R.$$

where $R = [\![\mathcal{U}']\!] \times (\Gamma \times \mathbb{N})^*$ and $\mathcal{U}'$ is an NFA that recognizes inputs annotated with left/right U-pairs of both $\mathcal{T}_1$ and $\mathcal{T}_2$.

It remains to show that, given the transducers $\mathcal{T}_1, \mathcal{T}_2$, there is a PSPACE witness procedure for $\mathsf{Busy}(\mathcal{T}_{1,U}), \mathsf{Busy}(\mathcal{T}_{2,U})$:

▶ **Proposition 10.** *Let $\mathcal{T}_1, \mathcal{T}_2$ be transducers with a total number n of states, and $\mathsf{Busy}(\mathcal{T}_{i,U}) = (Q_i, \hat{\Sigma}, \Gamma, \Delta_i, I_i, F_i)$ for $i = 1, 2$ (the input alphabet $\hat{\Sigma}$ is the same as for $\mathcal{U}'$). There is a non-deterministic procedure $\mathcal{W}$ that works in polynomial space in n and returns on a given transition $t_1 = (q_1, a_1, q_1', L_1, d_1)$ of $\mathsf{Busy}(\mathcal{T}_{1,U})$ any set $X \subseteq \Delta_2$ of transitions of $\mathsf{Busy}(\mathcal{T}_{2,U})$ such that for some $v \in L_1$:*

$$X = \{t_2 = (p_2, a_2, q_2, L_2, d_2) \in \Delta_2 \,:\, v \in L_2, \text{ and } t_2 \text{ has same shape as } t_1\}.$$

We can now conclude with the proof of our main result, which we recall here:

▶ **Theorem 1.** *Containment and equivalence of two-way transducers under the origin semantics is* PSPACE*-complete.*

Proof. The lower bound follows from a straightforward reduction from the classical equivalence problem of NFA. For the upper bound, in view of Proposition 9, we can consider origin containment between the transducers $\mathsf{Busy}(\mathcal{T}_{1,U})$ and $\mathsf{Busy}(\mathcal{T}_{2,U})$, obtained from $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. We recall that $\mathsf{Busy}(\mathcal{T}_{i,U})$ has at most $n_i = |\mathcal{T}_i|$ states and PSPACE-constructible transitions w.r.t. n_i, for $i = 1, 2$. We then apply Proposition 2 and Lemma 3 to reduce the containment problem to an emptiness problem for the intersection of two

PSPACE-constructible NFA, i.e. $\mathcal{B}$ and $\mathcal{U}'$. We observe that $\mathcal{U}'$ is basically the product of two NFA obtained from Lemma 6 by letting $\mathcal{T} = \mathcal{T}_i$, once for $i = 1$ and once for $i = 2$. Finally, we use the same arguments as in the proof of Corollary 4 to conclude that the latter emptiness problem is decidable in PSPACE w.r.t. $n = n_1 + n_2$. ◀

4 Containment modulo resynchronization

In this section we aim at generalizing the equivalence and containment problems for transducers with origins. The goal is to compare the origin semantics of any two transducers up to 'distortions', that is, differences in the origin tagging each position of the output.

Recall that $[\![\mathcal{T}]\!]_o$ is the set of synchronized pairs $\sigma = (u, \nu)$, where $u \in \Sigma^*$ is a possible input for the transducer $\mathcal{T}$ and $\nu \in (\Gamma \times \mathbb{N})^*$ is an output (tagged with origins) produced by a successful run of $\mathcal{T}$ on u. Given a pair $\sigma = (u, \nu) \in [\![\mathcal{T}]\!]_o$, we denote by $\mathsf{in}(\sigma)$, $\mathsf{out}(\sigma)$, and $\mathsf{orig}(\sigma)$ respectively the input word u, the output word obtained by projecting ν onto the finite alphabet Γ, and the sequence of input positions obtained by projecting ν onto $\mathbb{N}$. This notation is particularly convenient for describing resynchronizations, that is, relations between synchronized pairs. Following prior terminology from [15], we call *resynchronization* any relation R between synchronized pairs that preserves input and output words, but can modify the origins, namely, such that $(\sigma, \sigma') \in R$ implies $\mathsf{in}(\sigma) = \mathsf{in}(\sigma')$ and $\mathsf{out}(\sigma) = \mathsf{out}(\sigma')$.

The *containment problem modulo a resynchronization* R [15] is the problem of deciding, given two transducers $\mathcal{T}_1, \mathcal{T}_2$, if for every $\sigma' \in [\![\mathcal{T}_1]\!]_o$, there is $\sigma \in [\![\mathcal{T}_2]\!]_o$ such that $(\sigma, \sigma') \in R$, or in other words if every synchronized pair of $\mathcal{T}_1$ can be seen as a distortion of a synchronized pair of $\mathcal{T}_2$. In this case we write for short $\mathcal{T}_1 \subseteq_R \mathcal{T}_2$. We remark that, despite the name 'containment' and the notation, the relation $\subseteq_R$ is not necessarily transitive, in the sense that it may happen that $\mathcal{T}_1 \subseteq_R \mathcal{T}_2$ and $\mathcal{T}_2 \subseteq_R \mathcal{T}_3$, but $\mathcal{T}_1 \not\subseteq_R \mathcal{T}_3$.

We propose a class of resynchronizations that can be described in monadic second-order logic (*MSO*). The spirit is that, as synchronized pairs are (special) graphs, graph transformations à la Courcelle and Engelfriet [8] are an adequate tool to define resynchronizers. However, we cannot directly use MSO logic over origin graphs, since this would result in an undecidable containment problem. Our MSO resynchronizers will be able to talk about (regular) properties of the input word and the output word, and say how origins are distorted. We will show that containment modulo MSO resynchronizations is decidable, assuming a (decidable) restriction on the change of origins.

▶ **Definition 11.** An *MSO resynchronizer* R is a tuple $(\alpha, \beta, \gamma, \delta)$, where

- $\alpha(\overline{I})$ is an MSO formula over the signature of the input word, and has some free monadic variables $\overline{I} = (I_1, \ldots, I_m)$, called *input parameters*,
- $\beta(\overline{O})$ is an MSO formula over the signature of the output word, and has some free monadic variables $\overline{O} = (O_1, \ldots, O_n)$, called *output parameters*,
- γ is a function that maps any element $\tau \in \Gamma \times \mathbb{B}^n$, with $\mathbb{B} = \{0, 1\}$, to an MSO formula $\gamma(\tau)(\overline{I}, y, z)$ over the input signature that has a tuple of free monadic variables $\overline{I}$ (input parameters) and two free first-order variables y, z (called *source* and *target*, respectively),
- δ is a function that maps any pair of elements $\tau, \tau' \in \Gamma \times \mathbb{B}^n$, with $\mathbb{B} = \{0, 1\}$, to an MSO formula $\delta(\tau, \tau')(\overline{I}, z, z')$ over the input signature that has a tuple of free monadic variables $\overline{I}$ (input parameters) and two free first-order variables z, z' (called *targets*).

The input and output parameters $\overline{I}, \overline{O}$ that appear in the formulas of an MSO resynchronizer play the same role as the parameters of an NMSO-transduction [14]. They allow to express regular properties of the input and output word, respectively, through the first two formulas,

$\alpha(\overline{I})$ and $\beta(\overline{O})$. The other two formulas are used to describe how the origin function is transformed. They depend on the input and output parameters, in particular, on the "type" of the positions of the output word, as defined next.

Given an output word $v = \mathsf{out}(\sigma)$ and an interpretation $\overline{O} = O_1, \ldots, O_n \subseteq \mathrm{dom}(v)$ for the output parameters, let us call *output-type* of a position $x \in \mathrm{dom}(v)$ the element $\tau = (v(x), b_1, \ldots, b_n) \in \Gamma \times \mathbb{B}^n$, where each b_i is either 1 or 0 depending on whether $x \in O_i$ or not. Based on the output-type τ of x, the formula $\gamma(\tau)(\overline{I}, y, z)$ describes how the origin of x is redirected from a source y to a target z in the input word. Similarly, $\delta(\tau, \tau')(\overline{I}, z, z')$ constraints the origins z, z' of two consecutive output positions $x, x+1$, based on their output-types τ and τ'. This is formalized below.

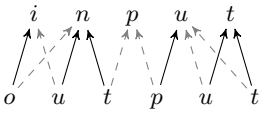
▶ **Definition 12.** An MSO resynchronizer $R = (\alpha, \beta, \gamma, \delta)$ induces the resynchronization $[\![R]\!]$ defined by $(\sigma, \sigma') \in [\![R]\!]$ if and only if $\mathsf{in}(\sigma) = \mathsf{in}(\sigma')$, $\mathsf{out}(\sigma) = \mathsf{out}(\sigma')$, and there are $\overline{I} = I_1, \ldots, I_m \subseteq \mathrm{dom}(\mathsf{in}(\sigma))$ and $\overline{O} = O_1, \ldots, O_n \subseteq \mathrm{dom}(\mathsf{out}(\sigma))$ such that:

- $(\mathsf{in}(\sigma), \overline{I}) \vDash \alpha$ (or equally, $(\mathsf{in}(\sigma'), \overline{I}) \vDash \alpha$),
- $(\mathsf{out}(\sigma), \overline{O}) \vDash \beta$ (or equally, $(\mathsf{out}(\sigma'), \overline{O}) \vDash \beta$),
- for all $x \in \mathrm{dom}(\mathsf{out}(\sigma))$ with output-type τ, if $\mathsf{orig}(\sigma)(x) = y$ and $\mathsf{orig}(\sigma')(x) = z$, then $(\mathsf{in}(\sigma), \overline{I}, y, z) \vDash \gamma(\tau)$,
- for all pairs of consecutive positions x and $x+1$ in $\mathsf{out}(\sigma)$ with output-types τ and τ', respectively, if $\mathsf{orig}(\sigma')(x) = z$ and $\mathsf{orig}(\sigma')(x+1) = z'$, then $(\mathsf{in}(\sigma), \overline{I}, z, z') \vDash \delta(\tau, \tau')$.

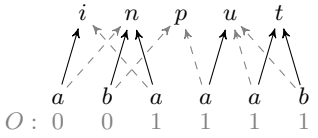
▶ **Example 13.** We provide a few examples of MSO resynchronizers:

1. The most unconstrained resynchronization, called *universal resynchronization*, groups any two synchronized pairs with the same input and output strings. This is readily defined by an MSO resynchronizer $R_{univ} = (\alpha, \beta, \gamma, \delta)$ without parameters, where $\alpha = \beta = \gamma(\tau)(y, z) = \delta(\tau, \tau')(z, z') = \mathit{true}$ for all $\tau, \tau' \in \Gamma$.

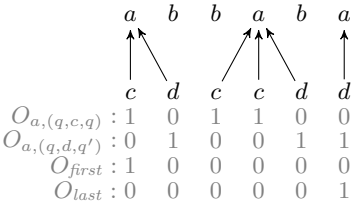
2. Consider a resynchronization that displaces the origin in a synchronized pair by one position to the left or to the right. An instance of this resynchronization is shown to the right, where solid arrows represent origins in the initial synchronized pair, and dashed arrows represent origins in the modified synchronized pair. This transformation can be defined by a parameterless MSO resynchronizer $R_{\pm 1} = (\alpha, \beta, \gamma, \delta)$, where $\alpha = \beta = \mathit{true}$, $\gamma(\tau)(y, z) = (z = y - 1) \vee (z = y + 1)$, and $\delta(\tau, \tau')(z, z') = \mathit{true}$ for all $\tau, \tau' \in \Gamma$. Note that the reflexive and transitive closure of $[\![R_{\pm 1}]\!]$ gives the universal resynchronization $[\![R_{univ}]\!]$.



3. We give a variant of the previous resynchronizer $R_{\pm 1}$, where the direction of movement of the origin is controlled by a property of the output position, e.g. the parity of the number of b's that precede that position in the output. We see an instance of the resynchronization to the right (as usual, solid and dashed arrows represent initial and modified origins). The transformation can be defined by an MSO resynchronizer with a single output parameter O that encodes the parity condition (an interpretation of O is shown in the figure as an annotation of the output over $\mathbb{B} = \{0, 1\}$). Formally, we let $R'_{\pm 1} = (\alpha, \beta, \gamma, \delta)$, where $\alpha = \mathit{true}$, $\beta(O) = \forall x \, \big(b(x) \to (x \in O \leftrightarrow x + 1 \notin O)\big) \wedge \big(\neg b(x) \to (x \in O \leftrightarrow x + 1 \in O)\big) \wedge (x = \mathit{first} \to x \in O)\big)$, and γ and δ are defined on the basis of the output-types $\tau, \tau' \in \Gamma \times \mathbb{B}$, as follows: $\gamma(\tau)(y, z)$ enforces either $z = y + 1$ or $z = y - 1$ depending on whether $\tau \in \Gamma \times \{0\}$ or $\tau \in \Gamma \times \{1\}$, and $\delta(\tau, \tau')(z, z') = \mathit{true}$.

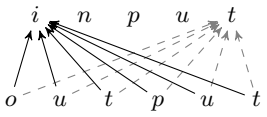


4. Let us now consider a resynchronization that does not modify the origins, but only constrain them so as to obtain a 'regular' subset of synchronization pairs. Here the allowed synchronization pairs are those that correspond to the process of applying a rational substitution $f : \Sigma \to 2^{\Gamma^*}$ to an input over Σ. The figure to the right describes a possible synchronized pair for f defined by $f(a) = c^*d$ and $f(b) = \varepsilon$. Below, we show how to define such a resynchronization for an arbitrary rational substitution f. We fix, for each letter $a \in \Sigma$, an NFA $\mathcal{A}_a$ that recognizes the regular language $f(a)$. We then define an MSO resynchronizer R_f that uses one output parameter $O_{a,t}$ for every letter $a \in \Sigma$ and every transition t of $\mathcal{A}_a$, plus two additional output parameters O_{first} and O_{last}. By a slight abuse of notation, given the output-type τ of a position x, we write $\tau[a,t] = 1$ whenever $x \in O_{a,t}$, and similarly for $\tau[\mathit{first}]$ and $\tau[\mathit{last}]$. The first formula α of the resynchronizer holds vacuously, as we have no restriction on the input. The second formula β requires that the parameters $O_{a,t}$ form a partition of the output domain in such a way that if a position x is labeled by a letter $c \in \Gamma$ and $x \in O_{a,t}$, then t is a c-labeled transition of $\mathcal{A}_a$. In addition, β requires that the parameters O_{first} and O_{last} are singletons consisting of the first and the last output position, respectively. The third component γ of the resynchronizer is defined by $\gamma(\tau)(y,z) = a(y) \wedge (y = z)$ whenever $\tau[a,t] = 1$ (the origin is not modified, and the transition annotating the output position must belong to the correct NFA). The last component δ restricts further the parameters and the origins for consecutive output positions, so as to simulate successful runs of the NFA. Formally, $\delta(\tau,\tau')(z,z')$ enforces the following constraints:



- if $\tau[a,t] = 1$, $\tau'[a,t'] = 1$, and $z = z'$, then the target state of t coincide with the source state of t', namely, tt' forms a factor of a run of $\mathcal{A}_a$,
- if $\tau[a,t] = 1$, $\tau'[a',t'] = 1$, and $z < z'$, then the target state of t must be final, the source state of t' must be initial, and every input letter strictly between z and z' is mapped via f to a language that contains ε,
- if $\tau[\mathit{first}] = 1$ and $\tau[a,t]$, then the source state of t must be initial, and every input letter strictly before z is mapped via f to a language that contains ε,
- if $\tau'[\mathit{last}] = 1$ and $\tau'[a,t]$, then the target state of t must be final, and every input letter strictly after z' is mapped via f to a language that contains ε.

5. We conclude the list of examples with a resynchronization that moves the origin of an arbitrarily long output over an arbitrarily long distance. This resynchronization contains the pairs (σ,σ'), where σ (resp. σ') maps every output position to the first (resp. last) input position, as shown in the figure. This is defined by the MSO resynchronizer $R_{1st-to-last} = (\alpha,\beta,\gamma,\delta)$, where $\alpha = \beta = \delta(\tau,\tau')(z,z') = \mathit{true}$ and $\gamma(\tau)(y,z) = (y = \mathit{first}) \wedge (z = \mathit{last})$, for all $\tau,\tau' \in \Gamma$. Note that the resynchronization $[\![R_{1st-to-last}]\!]$ is also 'one-way', in the sense that it contains only synchronized pairs that are admissible outcomes of runs of one-way transducers. However, $[\![R_{1st-to-last}]\!]$ is not captured by the formalism of one-way *rational* resynchronizers from [15].



Recall the Example 13.1 above, which defines the universal resynchronization $[\![R_{univ}]\!]$. The containment problem modulo $[\![R_{univ}]\!]$ boils down to testing classical containment between transducers *without origins*, which is known to be undecidable [18]. Based on this, it is clear that in order to compare effectively transducers modulo resynchronizations, we need to

restrict further our notion of MSO resynchronizer. Intuitively, what makes the containment problem modulo resynchronization undecidable is the possibility of redirecting many sources to the same target. This possibility is explicitly forbidden in the definition below. Also observe that, since $[\![R_{univ}]\!]$ is the reflexive and transitive closure of $[\![R_{\pm 1}]\!]$, we cannot take our resynchronizations to be equivalence relations.

▶ **Definition 14.** An MSO resynchronizer $(\alpha, \beta, \gamma, \delta)$ is *k-bounded* if for all inputs u, parameters $\overline{I} = I_1, \ldots, I_m \subseteq \mathrm{dom}(u)$, output-types $\tau \in \Gamma \times \mathbb{B}^n$, and targets $z \in \mathrm{dom}(u)$, there are at most k distinct sources $y_1, \ldots, y_k \in \mathrm{dom}(u)$ such that $(u, \overline{I}, y_i, z) \vDash \gamma(\tau)$ for all $i = 1, \ldots, k$. An MSO resynchronizer is *bounded* if it is k-bounded for some k.

Note that all resynchronizers from Example 13 but R_{univ} are bounded. For instance, $R_{1st-to-last}$ is 1-bounded and $R_{\pm 1}$ is 2-bounded.

It is also easy to decide the boundedness property of an MSO resynchronizer by reducing it to a problem of finite-ambiguity for finite automata [22]:

▶ **Proposition 15.** *t is decidable to know whether a given MSO resynchronizer is bounded.*

The goal is to reduce the problem of containment modulo a bounded MSO resynchronizer to a standard containment problem in the origin semantics. For this, the natural approach would be to show that transducers are effectively closed under bounded MSO resynchronizers. However, this closure property cannot be proven in full generality, because of the (input) parameters that occur in the definition of resynchronizers. More precisely, *two-way* transducers cannot guess parameters in a consistent way (different guesses could be made at different visits of the same input position). We could show the closure if we adopted a slightly more powerful notion of transducer, with so-called *common guess* [6]. Here we prefer to work with classical two-way transducers and explicitly deal with the parameters. Despite the different terminology, the principle is the same: parameters are guessed beforehand and accessed by the two-way transducer as explicit annotations of the input. Given a transducer $\mathcal{T}$ over an expanded input alphabet $\Sigma \times \Sigma'$ and an NFA $\mathcal{A}$ over $\Sigma \times \Sigma'$, we let

$$[\![\mathcal{T}]\!]_o \restriction_\Sigma^{\mathcal{A}} \;=\; \big\{(u, \nu) \in \Sigma^* \times (\Gamma \times \mathbb{N})^* \;:\; \exists\, u' \in \Sigma'^{|u|} \;\; u \otimes u' \in [\![\mathcal{A}]\!],\; (u \otimes u', \nu) \in [\![\mathcal{T}]\!]_o\big\}.$$

In other words, $[\![\mathcal{T}]\!]_o \restriction_\Sigma^{\mathcal{A}}$ is obtained from $[\![\mathcal{T}]\!]_o$ by restricting the inputs of $\mathcal{T}$ via $\mathcal{A}$, and then projecting them on Σ.

The following result is the key to reduce containment modulo bounded MSO resynchronizers to containment in the origin semantics.

▶ **Theorem 16.** *Given a bounded MSO resynchronizer R with m input parameters, a transducer $\mathcal{T}$ with input alphabet $\Sigma \times \Sigma'$ and an NFA $\mathcal{A}$ over $\Sigma \times \Sigma'$, one can construct a transducer $\mathcal{T}'$ with input alphabet $\Sigma \times \Sigma' \times \mathbb{B}^m$ and an NFA $\mathcal{A}'$ over $\Sigma \times \Sigma' \times \mathbb{B}^m$ such that*

$$[\![\mathcal{T}']\!]_o \restriction_\Sigma^{\mathcal{A}'} \;=\; \big\{\sigma' \;:\; (\sigma, \sigma') \in [\![R]\!] \text{ for some } \sigma \in [\![\mathcal{T}]\!]_o \restriction_\Sigma^{\mathcal{A}}\big\}.$$

Moreover, if R is fixed, $\mathcal{T}$ has n states and PSpace*-constructible transitions w.r.t. n, and $\mathcal{A}$ is* PSpace*-constructible w.r.t. n, then $\mathcal{T}'$ and $\mathcal{A}'$ have similar properties, namely, $\mathcal{T}'$ has a number of states polynomial in n and* PSpace*-constructible transitions w.r.t. n, and $\mathcal{A}'$ is* PSpace*-constructible w.r.t. n.*

Proof. To prove the claim, it is convenient to assume that R has no input nor output parameters. If this were not the case, we could modify $\mathcal{T}$ in such a way that it reads inputs over $\Sigma \times \Sigma' \times \mathbb{B}^m$, exposing a valuation of the parameters $\overline{I}$, and produces outputs over $\Gamma \times \mathbb{B}^n$, exposing a valuation $\overline{O}$. We could then apply the constructions that follow, and finally modify the resulting transducer $\mathcal{T}'$ by projecting away the input and output

annotations. We observe that the projection operation on the output is easier and can be implemented directly at the level of the transitions of $\mathcal{T}'$, while the projection of the input requires the use of the notation $[\![\mathcal{T}']\!]_o|_\Sigma^{\mathcal{A}'}$, for the reasons that we discussed earlier. In both cases the complexity bounds are preserved.

Let $R = (\alpha, \beta, \gamma, \delta)$ a bounded MSO resynchronizer with no input/output parameters. Since there are no existentially quantified parameters, $[\![R]\!]$ can be seen as the relational composition of four different resynchronizations, as induced by the formulas of R. Formally, we have $[\![R]\!] = [\![R_\alpha]\!] \circ [\![R_\beta]\!] \circ [\![R_\gamma]\!] \circ [\![R_\delta]\!]$, where $R_\alpha = (\alpha, \mathit{true}, \gamma_{id}, \delta_{\mathit{true}})$, $R_\beta = (\mathit{true}, \beta, \gamma_{id}, \delta_{\mathit{true}})$, $R_\gamma = (\mathit{true}, \mathit{true}, \gamma, \delta_{\mathit{true}})$, $R_\delta = (\mathit{true}, \mathit{true}, \gamma_{id}, \delta)$, $\gamma_{id}(\tau)(y, z) = (y = z)$ and $\delta_{\mathit{true}}(\tau, \tau')(z, z') = \mathit{true}$ for all output-types $\tau, \tau' \in \Gamma$. This means that, to prove the claim, it suffices to consider only one resynchronizer at a time among R_α, R_β, R_γ, R_δ. The case of R_α, R_β is quite straightforward, since it amounts to intersect the input and output with the regular language of α and β, respectively.

We now consider the most interesting case, that of $R_\gamma = (\mathit{true}, \mathit{true}, \gamma, \delta_{\mathit{true}})$, which modifies the origins. The case of δ is similar and can be found in the full version. The rough idea here is that $\mathcal{T}'$ has to simulate an arbitrary run of $\mathcal{T}$, by displacing the origins of any output letter with type τ from a source y to a target z, as indicated by the formula $\gamma(\tau)(x, y)$. Since a factor of an output of $\mathcal{T}$ that originates at the same input position can be broken up into multiple sub-factors with origins at different positions, here it is convenient to assume that $\mathcal{T}$ outputs at most a single letter at each transition. This assumption can be made without loss of generality, since we can reproduce any longer word v that is output by some transition $(q, i) \xrightarrow{a|L} (q', i')$ letter by letter, with several transitions that move back and forth around position i.

The idea of the construction is as follows. Whenever $\mathcal{T}$ outputs a letter b with origin in y, $\mathcal{T}'$ non-deterministically moves to some position z that, together with y, satisfies the formula $\gamma(b)$ (note that b is also the output-type of the produced letter). Then $\mathcal{T}'$ produces the same output b as $\mathcal{T}$, but at position z. Finally, it moves back to the original position y. For the latter step, we will exploit the fact that R is bounded.

Now, for the details, we construct from γ a finite monoid $(M, \cdot)$, a monoid morphism $h : (\Sigma \times \Sigma' \times \mathbb{B} \times \mathbb{B})^* \to M$, and some subsets F_τ of M_τ, for each output-type $\tau \in \Gamma$, such that $(u, x, y) \vDash \gamma(\tau)$ if and only if $h(u_{x,y}) \in F_\tau$, where $u_{x,y}$ is the encoding on u of the positions x and y, namely, $u_{x,y}(i) = \big((u(i), b_{i=x}, b_{i=y})\big)$ for all $1 \leq i \leq |u|$, and $b_{i=x}$ (resp. $b_{i=y}$) is either 1 or 0 depending on whether $i = x$ (resp. $i = y$) or not. Similarly, we denote by $u_{\emptyset,\emptyset}$ the encoding on u of two empty monadic predicates. For all $1 \leq i \leq j \leq |u|$, we then define $\ell_i = h(u_{\emptyset,\emptyset}[1, i-1])$, $r_j = h(u_{\emptyset,\emptyset}[j+1, |u|])$, and $m_{i,j} = h(u_{i,j}[i, j])$. We observe that

$$(u, y, z) \vDash \gamma(\tau) \qquad \text{iff} \qquad \begin{cases} \ell_y \cdot m_{y,z} \cdot r_z \in F_\tau & \text{if } y \leq z \\ \ell_z \cdot m_{z,y} \cdot r_y \in F_\tau & \text{if } y > z. \end{cases}$$

The elements ℓ_i and r_i associated with each position i of the input u are functionally determined by u. In particular the word $\ell_1 \dots \ell_{|u|}$ (resp. $r_1 \dots r_{|u|}$) can be seen as the run of a deterministic (resp. co-deterministic) automaton on u. Without loss of generality, we can assume that the values ℓ_i and r_i are readily available as annotations of the input at position i, and checked by means of a suitable refinement $\mathcal{A}'$ of the NFA $\mathcal{A}$.

We now describe how $\mathcal{T}'$ simulates a transition of $\mathcal{T}$, say $q \xrightarrow{a|b} q'$, that originates at a position y and produces the letter b (the simulation of a transition with empty output is straightforward). The transducer $\mathcal{T}'$ stores in its control state the transition rule to be simulated and the monoid element ℓ_y associated with the current position y (the source). It then guesses whether the displaced origin z (i.e. the target) is to the left or to the right

of y. We only consider the case where $z \geq y$ (the case $z < y$ is symmetric). In this case $\mathcal{T}'$ starts moving to the right, until it reaches some position $z \geq y$ such that $(u, y, z) \vDash \gamma(b)$ (as we explained earlier, this condition is equivalent to checking that $\ell_y \cdot m_{y,z} \cdot r_z \in F_b$). Once a target z is reached, $\mathcal{T}'$ produces the same output b as the original transition, and begins a new phase for backtracking to the source y. During this phase, the transducer will maintain the previous monoid elements ℓ_y, $m_{y,z}$, and, while moving leftward, compute $m_{z',z}$ for all $z' \leq z$. We claim that there is a unique z' such that $\ell_{z'} = \ell_y$ and $m_{z',z} = m_{y,z}$, and hence such z' must coincide with the source y. Indeed, if this were not the case, we could pump the factor of the input between the correct source y and some $z' \neq y$, showing that the MSO resynchronizer R_γ is not bounded. Based on this, the transducer $\mathcal{T}'$ can move back to the correct source y, from which it can then simulate the change of control state from q to q' and move to the appropriate next position. Any run of $\mathcal{T}'$ that simulates a run of $\mathcal{T}$ on input u, as described above, results in producing the same output v as $\mathcal{T}$, but with the origin mapping modified from $i \mapsto y_i$ to $i \mapsto z_i$, for all $1 \leq i \leq |v|$ and for some $1 \leq y_i, z_i \leq |u|$ such that $(u, y_i, z_i) \vDash \gamma(v(i))$. In other words, we have $[\![\mathcal{T}']\!]_o = \{\sigma' : (\sigma, \sigma') \in [\![R]\!],\ \sigma \in [\![\mathcal{T}]\!]_o\}$.

With the above constructions, if R is fixed and if $\mathcal{T}$ has n states and PSPACE-constructible transitions w.r.t. n, then similarly $\mathcal{T}'$ has a number of states polynomial in n and PSPACE-constructible transitions w.r.t. n. Finally, a PSPACE-constructible NFA $\mathcal{A}'$ can be obtained from a direct product of the NFA $\mathcal{A}$, $\mathcal{U}$, and a suitable NFA for checking inputs annotated with the monoids elements ℓ_z, r_z. ◀

The above result is used to reduce the containment problem modulo a bounded MSO resynchronizer R to a containment problem with origins (relativized to correctly annotated inputs). That is, if $\mathcal{T}_1, \mathcal{T}_2$ are transducers with input alphabet Σ, and $\mathcal{T}_2', \mathcal{A}'$ are over the input alphabet $\Sigma \times \Sigma'$ and constructed from $\mathcal{T}_2$ using Theorem 16, then

$$\mathcal{T}_1 \subseteq_R \mathcal{T}_2 \quad \text{iff} \quad [\![\mathcal{T}_1]\!]_o \subseteq [\![\mathcal{T}_2']\!]_o \downarrow_\Sigma^{\mathcal{A}'} \quad \text{iff} \quad [\![\mathcal{T}_1]\!]_o \uparrow^{\Sigma\times\Sigma'} \cap R' \subseteq [\![\mathcal{T}_2']\!]_o \cap R'$$

where $R' = [\![\mathcal{A}']\!] \times (\Gamma \times \mathbb{N})^*$ and $\uparrow^{\Sigma'}$ is the inverse of the input-projection operation, i.e. $[\![\mathcal{T}]\!]_o \uparrow^{\Sigma\times\Sigma'} = \{(u \otimes u', \nu) \in (\Sigma \times \Sigma')^* \times (\Gamma \times \mathbb{N})^* : u \otimes u' \in [\![\mathcal{A}]\!],\ (u, \nu) \in [\![\mathcal{T}]\!]_o\}$. We also recall from Section 3 that the latter containment reduces to emptiness of a PSPACE-constructible NFA, which can then be decided in PSPACE w.r.t. the sizes of $\mathcal{T}_1$ and $\mathcal{T}_2$. We thus conclude with the following result:

▶ **Corollary 17.** *The problem of deciding whether $\mathcal{T}_1 \subseteq_R \mathcal{T}_2$, for any pair of transducers $\mathcal{T}_1, \mathcal{T}_2$ and for a fixed bounded MSO resynchronizer R, is* PSPACE*-complete.*

5 Conclusions

We studied the equivalence and containment problems for non-deterministic, two-way word transducers in the origin semantics, and proved that the problems are decidable in PSPACE, which is the lowest complexity one could expect given that equivalence and containment of NFA are already PSPACE-hard. This result can be contrasted with the undecidability of equivalence of non-deterministic, one-way word transducers in the classical semantics.

We have also considered a variant of containment up to 'distortions' of the origin, called containment modulo a resynchronization R, and denoted $\subseteq_R$. We identified a broad class of resynchronizations, definable in MSO, and established decidability of the induced containment problem. In fact, we obtained an optimal fixed-parameter complexity result: testing a containment $\mathcal{T}_1 \subseteq_R \mathcal{T}_2$ modulo a bounded MSO resynchronization R is PSPACE-complete in the size of the input transducers $\mathcal{T}_1, \mathcal{T}_2$, where the fixed parameter is the size of the formulas used to describe R.

Our logical definition of resynchronizations talks implicitly about origin graphs. Since we cannot encode the origin semantics of arbitrary two-way transducers by words, we have chosen to work directly with origin graphs defined by two-way transducers. A classical way to define resynchronizations of origin graphs would be to use logical formalisms for graph transformations. Unfortunately, the classical MSO approach [7, 13] does not work in our setting, since satisfiability of MSO over origin graphs of two-way transducers is already undecidable, which means that the definable resynchronizations would not be realizable by two-way transducers.

Another possibility would be to use a decidable logic over origin graphs, like the one introduced by Filiot et al. in [10]. Their logic is not suited either for our purposes, since it allows predicates of arbitrary arity defined using MSO over the input word. The reason is that single head devices would not be able to move between the tuples of positions related by those definable predicates, and thus, in particular, we would not be able to guarantee that the definable resynchronizations are realizable by two-way transducers.

Yet an alternative approach to the above problem consists in viewing tagged outputs as data words, and using transducers to transform data words. Durand-Gasselin and Habermehl introduced in [11] a framework for transformations of data words. However, this approach would be unsatisfactory, because the transformation does not take the input into account.

We conjecture that the bounded MSO resynchronizers defined here strictly capture the rational resynchronizers introduced in [15]. In particular, although MSO resynchronizers do not have the ability to talk about general origin graphs, they presumably can describe 'regular' origin graphs of one-way transducers, i.e., graphs expressed by regular languages over sequences that alternate between the input and the output word. We also recall that the MSO resynchronizer $R_{1st-to-last}$ from Example 13.5 contains only origin graphs of one-way transducers, but cannot be defined by a rational resynchronizer.

Another natural question that we would like to answer concerns compositionality: are bounded MSO resynchronizers closed under relational composition? In other words, given two bounded MSO resynchronizers R, R', is it possible to find (possibly effectively) a bounded MSO resynchronizer R'' such that $[\![R'']\!] = [\![R]\!] \circ [\![R']\!]$?

References

1 A.V. Aho, J.E. Hopcroft, and J.D. Ullman. A general theory of translation. *Math. Syst. Theory*, 3(3):193–221, 1969.

2 Rajeev Alur and Pavel Cerný. Expressiveness of streaming string transducer. In *Proc. of FSTTCS'10*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

3 Rajeev Alur and Jyotirmoy Deshmukh. Nondeterministic streaming string transducers. In *Automata, Languages and Programming - 38th International Colloquium (ICALP'11)*, volume 6756 of *LNCS*. Springer, 2011.

4 Jean Berstel. *Transductions and context-free languages*. Teubner Studienbücher Stuttgart, 1979.

5 Mikolaj Bojańczyk. Transducers with origin information. In *ICALP'14*, LNCS, pages 26–37. Springer, 2014.

6 Mikolaj Bojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. Which classes of origin graphs are generated by transducers? In *ICALP'17*, volume 80 of *LIPIcs*, pages 114:1–114:13, 2017.

7 Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Transformations: Foundations*, volume 1, pages 165–254. World Scientific, 1997.

8 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic. A language-theoretic approach*. Encyclopedia of Mathematics and its applications, Vol. 138. Cambridge University Press, 2012.

9 Karel Culik II and Juhani Karhumäki. The equivalence problem for single-valued two-way transducers (on NPDT0L languages) is decidable. *SIAM J. Comput*, 16(2):221–230, 1987.

10 Luc Dartois, Emmanuel Filiot, and Nathan Lhote. Logics for word transductions with synthesis. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 295–304. ACM, 2018.

11 Antoine Durand-Gasselin and Peter Habermehl. Regular transformations of data words through origin information. In *Proc. of FoSSaCS'15*, LNCS, pages 285–300. Springer, 2016.

12 Samuel Eilenberg. *Automata, languages, and machines*. Academic press, 1974.

13 Joost Engelfriet. Context-free graph grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 125–213. Springer, 1997.

14 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2(2):216–254, 2001.

15 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *ICALP'16*, volume 55 of *LIPIcs*, pages 125:1–125:14, 2016.

16 Emmanuel Filiot, Sebastian Maneth, Pierre-Alain Reynier, and Jean-Marc Talbot. Decision problems of tree transducers with origin. *Inf. Comput.*, 261:311–335, 2018.

17 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *ACM SIGLOG News*, pages 4–19, 2016.

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

19 M.-P. Schützenberger. A remark on finite transducers. *Information and Control*, 4(2-3):185–196, 1961.

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

21 Moshe Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30(5):261–264, 1989.

22 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theor. Comput. Sci.*, 88(2):325–349, 1991.