Master thesis
Computing Science

Radboud University

Resynchronizability of origin transducers

Author:
Jan Martens
s4348435

Daily supervisor:
Denis Kuperberg
ENS Lyon

Assessor:
Herman Geuvers
Radboud University

Second Assessor:
Jurriaan Rot
Radboud University

December 3, 2019

Abstract

Origin semantics introduced by Bojańczyk is a fine grained semantics for transducers, that not only expresses the relation between input and output words, but also includes a function that given an output position returns the input position where it was produced: the origin. In this thesis we study *resynchronizations*, a tool to relax the notion of origin while maintaining decidable containment and equivalence. We study the notion of containment up to some unknown resynchronization, and show that this forms a pre-order strictly in between classical and origin containment. Using a notion of *desynchronized blocks* we observe the non-existence of this resynchronization and show this containment is undecidable for one-way finite state transducers, which was an open problem.

Contents

Chapter 1

Introduction

The study of transducers, models describing a function or relation between words, is pervasive in computer science. Since the beginning of automata theory a model called finite state transducers has been researched. These transducers are an extension of normal automata that also produce output. These finite state transducers share similarities with finite automata. However differently from automata, for transducers features as non-determinism or the ability to move both ways in a word (two-way transducers) does impact expressiveness. Together with the fact that many natural problems like equivalence and containment of transducers are undecidable [Iba78], this class seems less robust than the regular languages.

To tackle these problems Mikołaj Bojańczyk proposed a refined semantics [Boj14] in which transductions do not only compute a partial function from word to word, but also retain the *origin* of every output position. Transducers now not only compute a relation between input and output word but produce objects called *origin graphs* which are a relation between input word and output word and comes with an extra function mapping every output position to an input position. With this extra piece of information about how the output was computed from the input, there seems to form a more robust class. The equivalence of origin transductions is decidable, and it allows more natural algebraic characterizations [Boj14, BMPP18].

This equivalence and containment of transducers under origin semantics however differs from equivalence and containment with the classical semantics where one is only interested in the underlying relation of input and output words. In origin semantics two transducers are only equal if they compute the same origin graphs, so also the origin mapping must be equal. This equivalence is much stricter than the one in the classical sense, so this motivates the use of resynchronizations to study an intermediate notion of containment and equivalence that tells the input and output word are the same, and the origin mapping shares some similarity.

Resynchronizations as discussed in [BMPP18] are a tool to express similarity between different origin graphs that have the same underlying relation. These resynchronizations relate origin graphs that have the same input and output word but in which the origin mapping differs in a *regular* way. With a given resynchronization it is possible to expand the origin containment to containment up to some resynchronizer. That is all graphs are contained but the origins are allowed to change in a defined way. When restricting to bounded regular resynchronizers, which allow only a bounded number of origins to be redirected to each position, this containment is decidable in PSPACE [BMPP18], which is not more difficult than containment between NFAs.

In this thesis we study containment up-to some unknown bounded regular resynchronization. For two transducers T_1, T_2 we say a transducer T_1 resynchronizes to T_2 if there exists a bounded regular resynchronizer such that T_1 is contained in T_2 up to an unknown regular resynchronizer. This relation is studied in [BKM+19] where they call it the synthesis problem and show it to be decidable for a subset of transducers called k-ambiguous. The general case is left open.

To solve this open problem we study the expressiveness of relations that bounded regular resynchronizers can describe. In order to witness the absence of a bounded regular resynchronizer we introduce a notion of desynchronized, and prove that there exists a bounded regular resynchronizer inducing a relation if and only if there exists a bound on the desynchronization. Using this result we build a reduction from a problem called INFTAPE on Turing machines asking whether a given Turing machine uses an infinite amount of tape. By using a domino construction that is used in the proof that the Post Correspondence Problem is undecidable [Pos46], we reduce INFTAPE to resynchronizibilty of one-way finite state transducers. This shows the resynchronizability relation is undecidable for one-way finite state transducers.

Chapter 2

Preliminaries

2.1 Languages, automata and algebra

While classically finite automata are often used to express the regular languages, in this thesis we use two other formalisms, that express exactly the same class of languages. First we use Monadic second order (MSO) logic on strings. Secondly we consider an algebraic classification using monoids. Both techniques will be used later on in the thesis, MSO will be used to relate positions in a word and the monoid representation is used in the technical proof of Lemma 5.10.

2.1.1 MSO for words

A word can be expressed by a relational structure on the universe of positions of the word. This universe consists of a set of integers distinguishing the positions and the structure is given by predicates for the labels of the positions, and the order in which the positions occur.

Definition 2.1. *Given an alphabet Σ, a word $w \in \Sigma^*$, can be represented by the structure $\underline{w} = (dom(w), \leq, a \in \Sigma)$, in which:*

- *$dom(w) = \{1, 2, \ldots, |w|\}$ the set of positions of s*
- *$\leq$ a binary relation on positions $x \leq y$ means x is a position occurring before y or $x = y$.*
- *$a \in \Sigma$ a unary label predicate stating if the label of position x has label a. $a(x)$ is valid if and only if position x is labeld with an a.*

An MSO formula is defined by the following grammar where a ranges over an alphabet.

$$\varphi, \psi := a(x) \mid x \leq y \mid x \in X \mid \exists x.\phi \mid \exists X.\phi \mid \varphi \wedge \psi \mid \neg\varphi$$

We allow the operators $\wedge, \rightarrow, \exists$ in which $\exists$ can quantify over both sets and elements. We use uppercase letters for variables ranging over sets of positions, and lowercase letters for single positions. For example if Σ is an alphabet then $\exists x.a(x)$ on the structure of a word $w \in \Sigma^*$ means there exists a position $x \in dom(w)$ which is labeled $a \in \Sigma$.

Note that most of the usual combinators like $\forall, \vee, \neg$ can be created using the combinators which exist. For instance also other useful formula such as successor $s(x, y)$, *first*(x) and *last*(x) can be constructed:

$$s(x, y) = x < y \wedge \forall z.(z < y \rightarrow z \leq x)$$

$$first(x) = \forall y.x \leq y$$

$$last(x) = \forall y.y \leq x$$

We will use the interval notation $[y, z]$ in $\exists x \in [y, z].\psi$ as shorthand notation for the formula $\exists x.(x \leq z \wedge y \leq x) \rightarrow \psi$, and (y, z) for the open interval $\exists x.y < x \wedge x < z$.

If φ is an MSO formula and $\underline{w} = (w, \leq, a)$ an MSO structure $w \in \Sigma^*$ a word. We will write $\underline{w} \models \varphi$ if $\underline{w}$ is a valid model for φ. With classical MSO semantics meaning the formula φ is true in the model of w.

Definition 2.2. *The language of an MSO sentence φ denoted as $[\![\varphi]\!]$ is given by:*

$$[\![\varphi]\!] = \{w \mid w \in \Sigma^* \text{ and } \underline{w} \models \varphi\}$$

The class of languages that can be generated by MSO-sentences are exactly the regular languages. An important result known as the Büchi-Elgot-Trakhtenbrot theorem shown in the sixties [Büc60, Elg61, Tra61]. At some point in this thesis we will also use the notation $w \models \varphi$ for some word $w \in \Sigma^*$ and MSO-formula φ by which we mean the MSO-structure of w: $\underline{w} = (dom(w), \leq, a)$.

Example 2.3. *Some examples of languages defined by MSO formulas*

- $L_1 = \{w \mid \textit{every } a \textit{ is followed by a } b\}$

$$\varphi_1 = \forall x.a(x) \rightarrow (\exists y.s(y, x) \wedge b(y))$$

- $L_2 = \{w \mid |w| \textit{ is even}\}$

$$\begin{aligned} \varphi_2 = \exists X.(&\forall x.first(x) \rightarrow x \in X \\ &\forall x.last(x) \rightarrow x \notin X \\ &\forall x.\forall y.x \in X \wedge s(x, y) \rightarrow y \notin X \\ &\forall x.\forall y.x \notin X \wedge s(x, y) \rightarrow y \in X \\ &) \end{aligned}$$

- $L_3 = \{w \mid w \textit{ does not have an occurence of abba}\}$

$$\begin{aligned}\varphi_3 = \forall x.\forall y.\forall z.\forall u.\ (s(x,y) \wedge s(y,z) \wedge s(z,u)) \rightarrow \\ \neg(a(x) \wedge b(y) \wedge b(z) \wedge a(u))\end{aligned}$$

Free variables

We only discussed languages described by closed MSO sentences. However in this thesis we will also use MSO formulas with free variables. Extra information can be added by extending the alphabet, so that the values of the free variables is specified in the word. This way MSO formulas with free variables can be intuitively seen as a regular language.

For example consider the MSO formula s for the successor relation with two free first order variables x and y.

$$s(x,y) = x < y \wedge \forall z.(z < y \rightarrow z \leq x)$$

Now it makes no sense to talk about words which satisfy s but when x and y are fixed positions in a word $w \in \Sigma^*$ we can check the validity of s. If a word w with positions x and y satisfies s we write $w, x, y \models s$. To demonstrate this we can decorate words with the positions chosen for x, y:

$$\begin{array}{cccccc} w: & \text{a} & \text{b} & \text{c} & \text{b} & \text{a} \\ & & & \text{x} & & \\ & & & & \text{y} & \end{array} \qquad \begin{array}{cccccc} w: & \text{a} & \text{b} & \text{c} & \text{b} & \text{a} \\ & & & & \text{x} & \\ & & & & \text{y} & \end{array} \qquad \begin{array}{cccccc} w: & \text{a} & \text{b} & \text{c} & \text{b} & \text{a} \\ & \text{x} & & & & \\ & & & & \text{y} & \end{array}$$

(a) $w, x, y \models s$ (b) $w, x, y \not\models s$ (c) $w, x, y \not\models s$

These decorations can be added as an extension to the alphabet Σ, for every free variable add a boolean $b \in \mathbb{B}$ the the alphabet where a $1 \in \mathbb{B}$ indicates the position is in the free variable and $0 \in \mathbb{B}$ says the position is not in the free variable. To indicate what are the chosen positions x, y we get a word w' over the extended alphabet $w' \in \Sigma \times \mathbb{B}^2$.

$$w' : \left|\begin{array}{ccccc} \text{a} & \text{b} & \text{c} & \text{b} & \text{a} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right. \qquad w' : \left|\begin{array}{ccccc} \text{a} & \text{b} & \text{c} & \text{b} & \text{a} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right. \qquad w' : \left|\begin{array}{ccccc} \text{a} & \text{b} & \text{c} & \text{b} & \text{a} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right.$$

(a) $w' \in s$ (b) $w' \notin [\![s]\!]$ (c) $w' \notin [\![s]\!]$

Given an MSO formula φ with k free variables $X_1, \ldots X_k$ on structures over the alphabet Σ. Given a word $w \in \Sigma \times \mathbb{B}^k$ an element $x \in \mathit{dom}(w)$ labeled $(a, b_1, b_2, \ldots, b_k)$ for some letter $a \in \Sigma$ this represents the element $x \in X_i$ if and only if $b_i = 1$ for $i = 1, 2 \ldots, k$. The language of φ is a language over the alphabet $\Sigma \times \mathbb{B}^k$.

While the example uses a first order free variable this can be seen as a special case of a second order variable with an extra restriction that it only contains one element. This implies that any MSO formula with free variables can be seen as a regular language over an extended alphabet indicating the positions which are element of the free variables. For this reason when referring to MSO formulas we will sometimes call them regular properties.

2.1.2 Monoids recognizing languages

Another way to express exactly the class of regular languages is via an algebraic approach using finite monoids. This characterization gives some powerful tools which we use in this thesis.

A set M equipped with a binary operator $\cdot : M \times M \to M$ is called a monoid if the operator $\cdot$ is associative and there exists an identity element $e \in M$ such that for every $a \in M$, $a \cdot e = e \cdot a = a$. Sometimes written as a monoid $\mathcal{M} = (M, \cdot, e)$. However when it's clear from the context we will just call the set M a monoid.

A monoid morphism is a function between monoids preserving the structure of the monoid.

Definition 2.4. *Let $\mathcal{M} = (M, \cdot_m, 1_m)$, $\mathcal{N} = (N, \cdot_n, 1_n)$ be monoids then the function $f : M \to N$ is called a morphism if and only if:*

- *the identity is preserved: $h(1_m) = 1_n$ and*
- *the structure is preserved: $h(a \cdot_m b) = h(a) \cdot_n h(b)$ for every $a, b \in M$.*

We call the monoid on Σ^* the *free monoid* where the identity element is the empty word ε and the binary operator is given by string concatenation. The alphabet Σ is called the generators of the free monoid. For every monoid $\mathcal{M} = (M, \cdot, 1_m)$ and function $f : \Sigma \to M$, this can be extended in a unique morphism from the free monoid. This unique morphism is denoted by $f^* : \Sigma^* \to M$ and defined by $f^*(\varepsilon) = 1_m$ and $f^*(aw) = f(a) \cdot f^*(w)$.

A monoid $\mathcal{M}$ together with a function $f : \Sigma \to M$ and a subset $U \subseteq M$ is said to recognize the language $L \subseteq \Sigma^*$ if L is exactly the pre-image of U under the morphism f^* which is the unique morphism to the free monoid generated by f.

$$L = \{w \mid w \in \Sigma^* \text{ and } f^*(w) \in U\}$$

Theorem 2.5. *A language is regular if and only if it is recognized by a finite monoid.*

Proof. ($\Rightarrow$) On an alphabet Σ let NFA $\mathcal{N} = (Q, \delta : Q \to \mathcal{P}(Q), F \subseteq Q, I \subseteq Q)$. We now build a monoid M and function $f : \Sigma \to M$ to recognize the language of $\mathcal{N}$. Let the set of the monoid $M = \mathcal{P}(Q) \to \mathcal{P}(Q)$ since

Q is finite this is a finite set with identity element $1(P) = P$ and binary combinator which is function composition $\cdot : M \times M \to M$ as $(a \cdot b)(P) = b(a(P))$ for all $P \subseteq Q$.

Now we define the homomorphism $f : \Sigma \to M$ by $f(a)(P) = \bigcup_{q \in P} \delta(a, q)$. This can be extended in the homomorphism $f^* : \Sigma^* \to M$ by $f^*(\varepsilon)(P) = P$ and $f^*(wa) = f^*(w) \cdot f(a)$. The intituition for f^* is that describes all the paths the NFA can take on the input word. Now for every word $w \in \Sigma$, w is accepted by the NFA $\mathcal{N}$ if and only if $f^*(w)(I) \cap F \neq \emptyset$, which intuitively means that the automaton is able to reach a final state with a run of the automaton.

($\Leftarrow$) Given a finite monoid $\mathcal{M} = (M, \cdot_m, 1)$ with the homomorphism $f : \Sigma \to M$ and subset $U \subseteq M$. The language recognized by M is also recognized by DFA $\mathcal{D} = (M, \delta, 1, U)$ with $\delta(q, a) = q \cdot_m f(a)$. Now if a word $w \in \Sigma^*$ is in the language of M than $f^*(w) \in U$ the δ function has the same behaviour as $f^*(w)$ so $\delta^*(1, w) \in U$ so w is also accepted by D. $\square$

Example 2.6. *Given the alphabet $\Sigma = \{0, 1\}$ and the natural morphism $f : \Sigma \to M$ with M being the additive group $\mathbb{Z}/2\mathbb{Z}$.*

$$L_0 = \{w \mid w \in \{0,1\}^* \; |w|_1 \textit{ is even}\}$$

$$L_1 = \{w \mid w \in \{0,1\}^* \; |w|_1 \textit{ is odd}\}$$

Where the language L_0 is recognized by the monoid M, morphism f and subset $\{1\} \subset M$. The language L_1 is recognized by the monoid M, morphism f and subset $\{0\} \subset M$.

Ramsey factorization forests and the factorization forest theorem

A common and much used result in automata theory is that when the size of words grows beyond a certain point every run of a finite automaton must have a repetition of states. The run between this repetition can be repeated not changing the outcome of the automaton and thus producing a larger word. This argument is often used to prove languages are not regular and is called the pumping lemma.

For the main result of this thesis the argument of repetition is not sufficient, and we use a deep theorem by Simon [Sim90]. The theorem can be interpreted as that there exists a repetition of behaviors rather than repetitions of states. While in nature both theorems look alike, this theorem is in fact more powerful. In this section we will shortly introduce Ramsey factorization and give some example of basic intuition in the theorem. More details and an overview of the theorem in different subclasses of semigroups can be found in [Col13].

Let $\mathcal{M} = (M, \cdot, 1)$ be a monoid and $w \in M^*$ a word over the monoid. A *factorization tree* is then a tree in which each node is labelled by a monoid

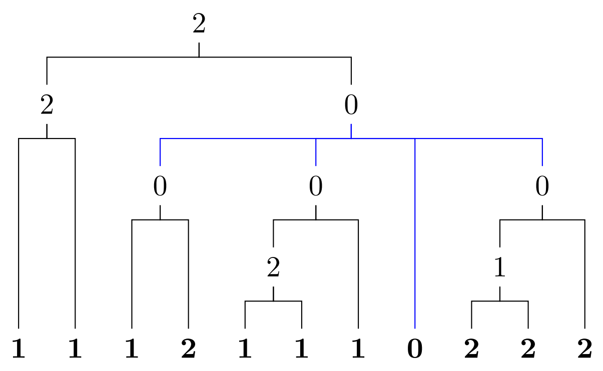


Figure 2.3: Ramsey factorization tree

element $m \in M$ and either a leaf, or an internal node with children. The value of each node is the product of it's children with the binary monoid operation. A factorization tree is called *Ramsey* if it has following structure.

Definition 2.7. *(Ramsey factorization) A Ramsey factorization tree is a tree in which each node is either:*

- *a leaf*
- *a node with two children*
- *a node in which all the children are labelled by the same idempotent value $e \in M$.*

In Figure 2.3 an example Ramsey factorization is given on the monoid $\mathcal{M} = (\{\mathbf{0}, \mathbf{1}, \mathbf{2}\}, +, 0)$ which is the additive group $\mathbb{Z}/3\mathbb{Z}$. The edge highlighted in blue is from the node where the third rule is used and indicates the repetition of behavior. Note the difference from loops because the word belonging to the values are in fact different.

Theorem 2.8. *(Simon's factorization forest theorem) For all monoids M and all words on the monoid $u \in M^+$, there exists a Ramsey factorization tree for u with f of height at most $k = 3|M| - 1$.*

The $k = 3|M| - 1$ is an overestimation but for our purposes this does not matter. The fact that there exists a bound on the height of the Ramsey factorization tree that is not dependent of the length of the word provides the notion of repetition of behavior. Given a regular language $L \in \Sigma^*$, we know by Theorem 2.5 that there exists a monoid M and a function $f : \Sigma \to M$ such that the monoid M with morphism $f^* : \Sigma^* \to M$ recognizes exactly the language L. Every word $a_1 a_2 \ldots a_k \in L$ can be represented as a word $m_1 \ldots m_k \in M^*$ by applying the function f that is for every $1 \leq i \leq n$, $m_i = f(a_i)$. Now using the pigeon hole principle and the factorization forest theorem we can conclude that for every $n \in \mathbb{N}$ there exists a length $m \in \mathbb{N}$ such that for any larger word $u \in L,\ |u| > m$ there exists a Ramsey factorization tree for u that has a node with at least n children.

For $n \geq 3$ This means all the children of the node have the same idempotent element in the monoid and thus the value of the children are a factor in the word that have idempotent behavior in the monoid.

2.2 Transducers

Extensions of automata that produce output are called *transducers*. The definition is obtained by extending the transition functions of normal automata so that it does not only consume a word, but also produce output on every transition. This way an automaton will not recognize a language of words but recognize a relation between input and output words, this relation realised by a transducer is called a *transduction*.

Definition 2.9. *A two-way non-deterministic transducer (2NT) is given by $T = (Q, \Sigma, \Gamma, \delta, q_0, F)$ a 6-tuple consisting of:*

- *Q a finite set of states*
- *Σ, Γ the finite input and output alphabet respectively*
- *$\delta : Q \times (\Sigma \cup \{\vdash, \dashv, \varepsilon\}) \times Q \times \{L, R\} \times \Gamma^*$ the transition relation, where $\vdash, \dashv \notin \Sigma$ are markers indicating the beginning and end of the input word, and L, R are for the directions* left *and* right
- *q_0 the initial*
- *$F \subseteq Q$ the set of final states.*

Every *transition* $(q, a, q', D, v) \in \delta$ means that the automaton is in state $q \in Q$ reads an $a \in \Sigma \cup \{\vdash, \dashv, \varepsilon\}$ outputs $v \in \Gamma^*$, transitions to state $q' \in Q$ and moves the reading head to the left if $D = L$ or to the right if $D = R$. As a diagram this transition is given as

$$\begin{tikzcd} q \arrow[r, "a|v, D"] & q' \end{tikzcd}$$

Transition $(q, a, q', D, v) \in \delta$

A *run* of a transducer T on the input word $w \in \Sigma^*$ is obtained by applying the transition relation of the automaton on the word $\vdash w \dashv$ where $\vdash, \dashv$ are the markers preventing the automaton of running out of bounds. We also don't allow the automata to produce output on the end markers

$\vdash, \dashv$. So for every state $q, q' \in Q$ if $(q, \vdash, q', D, v) \in \delta$ then direction $D = R$ and the output $v = \varepsilon$. If $(q, \dashv, q', D, v) \in \delta$ then $D = L$ And $v = \varepsilon$. This makes sure the automaton does not go out of bounds or produce output on the end markers.

The first step in the run of the automaton is in $(q_0, 1)$ meaning that the reading head is on the first letter a_1 of the input word $w = a_1 \dots a_n$ and the automaton is in the initial state q_0. A possible step from any configuration (q, i) to the next configuration (q', i') if it is in the transition relation $(q, a_i, q', D, v) \in \delta$ where a_i is the ith letter in the input w, $v \in \Gamma^*$ is the output produced by the transition, and i' is decided by the direction D from the direction. If the direction $D = L$ the reading head will go one position to the left so $i' = i - 1$ if $D = R$ the reading head will go one place to the right and $i' = i + 1$. We will write these transitions as $(q, i) \rightarrow_b (q', i')$ where b is the output produced $\delta \ni (q, a_i, q', d \in \{L, R\}, b)$. A run of a transducer M on the input word w is given as $(q_0, 1) \rightarrow_{b_0} \cdots \rightarrow_{b_n} (q_f, i)$ for some $0 \leq i \leq |w|$. A run is valid if $q' \in F$ a final state.

The semantics of a given two-way transducer M, $[\![M]\!] : \Sigma^* \times \Gamma^*$ is a relation of input and output words given by $[\![M]\!] = \{(w, v) \mid w \in \Sigma^* \text{ and } v \in \Gamma^*\}$ if on input word w there exists a valid run $(q_0, 1) \rightarrow_{v_1} \cdots \rightarrow_{v_n} (q_f, i)$ where $v = v_1 v_2 \dots v_n \in \Gamma^*$ is the concatenation of all the outputs on the transitions that is: $\delta \ni (q_i, a_i, D, q_{i+1}, d \in \{L, R\}, v_i)$.

A transducer is two-way deterministic $2DT$ if there are no ε transitions and for every input letter $a \in \Sigma \cup \{\vdash, \dashv\}$ and state $q \in Q$ there is at most 1 transition rule (q, a, D, q', v) for some $q' \in Q, D \in \{L, R\}, v \in \Gamma^*$. The relation that a deterministic transducer describes is functional as there is at most one run valid on each input word.

A transducers is called one-way non-deterministic $1NT$ if every transition is right moving, this means that every in δ has directen R. Additionally like in normal automata we require these $1NTs$ to stop at the end of the word. More precisely run of a $1NT$ on a word $w \in \Sigma^*$ given by $(q_0, 1) \rightarrow_{v_1} \dots (q_f, i)$ is only valid if $q_f \in F$ is a final state and $i = |w|$ the final position in the word. As an exception we have that ε transitions do not move the tape head. Since $1NTs$ only move in one way the direction D on transitions is omitted.

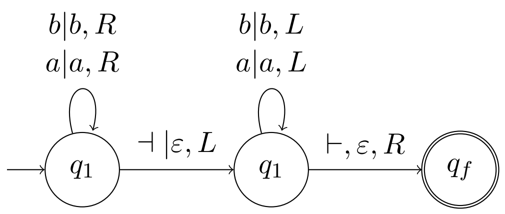
A one-way deterministic transducer $1DT$ is a transducer that is both one-way and deterministic. In contrast with the normal automata, the extension of the model with left transitions and the addition of non-determinism does change the class of transductions that is recognized.

$$1DT \neq 1NT \neq 2DT \neq 2NT$$

The class which in literature is called the *regular* transductions is the class exactly recognized by $2DTs$ as this is also precisely the class accepted by functional $MSOT$ transductions [EH01] and streaming string transducers

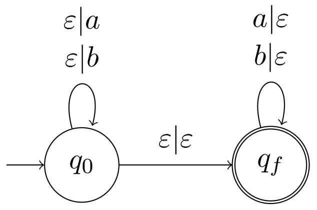
[AČ10]. More on the expressiveness and classifications off these different models are discussed in [BDGP17].

Example 2.10. *The 2DT $T_1 = (Q, \Sigma, \Gamma, \delta, q_0, \{q_f\})$ on input and output alphabet $\Sigma = \Gamma = \{a, b\}$ that computes the transduction $w \mapsto ww^r$. This transducer is given by the diagram:*



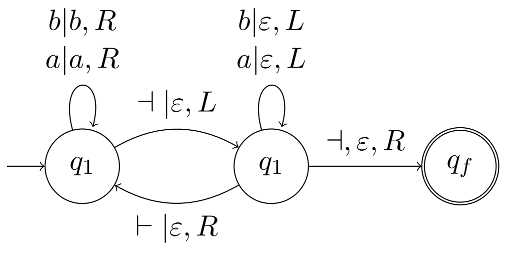
Transducer T_1

Example 2.11. *The 1NT $T_{full} = (Q, \Sigma, \Gamma, \delta, q_0, \{q_f\})$ on input and output alphabet $\Sigma = \Gamma = \{a, b\}$ that computes the full relation $[\![T_{full}]\!] = \Sigma^* \times \Gamma^*$. Given by the diagram:*



Transducer T_{full}

Example 2.12. *The 2NT $T_3 = (Q, \Sigma, \Gamma, \delta, q_0, \{q_f\})$ on input and output alphabet $\Sigma = \Gamma = \{a, b\}$ that computes the relation $[\![T_3]\!] = \{(w, w^n) \mid w \in \Sigma^*$ and $n \in \mathbb{N}\}$*



Transducer T_3

Chapter 3

Origin semantics

3.1 Introduction

The notion of *origin semantics* was introduced by Bojańczyk in [Boj14]. The main idea is that transducers not only produce a relation on input and output word, but also have information on where in the input word an output letter was produced: the origin. While in the classic semantics this information is thrown away, in the origin setting this information is kept. A run of the transducer now not only results in a pair of input word and output word, but also the origin of each output position. Such an object as in Figure 3.1 with a pair of input word, output word and the corresponding origin of where each output position originates in the input word is called an origin graph.

$$\begin{tikzcd}
\text{Input:} & a & b & c \\
\text{Origin:} & & & \\
\text{Output:} & c \arrow[uurr] & b \arrow[uu] & a \arrow[uull]
\end{tikzcd}$$

Figure 3.1: Possible origin graph that has input *abc* and output *cba*

The origin semantics of a transducer is the set of origin graphs that it produces. Origin semantics is more fine grained because origin graphs also contain the origin of every output position.

It is important to note that this is indeed a different semantics than the classical semantics. For example a run on a transducer $abc \mapsto cba$ can be computed in many different ways as seen in Figure 3.2, all producing different origin graphs while being the same in the classical sense. However if one would regard the reverse transduction $w \mapsto w^R$ the most natural solution in all frameworks describe the same origin semantics.

These new semantics for the transducers prove to be interesting since they have some promising results indicating a more robust class than trans-

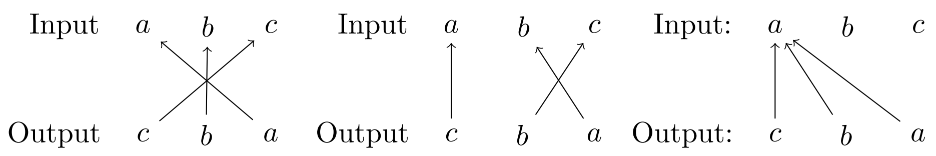


Figure 3.2: Origin graphs have input *abc* and output *cba*

ductions under classical semantics. Origin semantics allow a machine independent characterization of the regular transducers and a natural characterization of some of the natural subclasses including FO-definable and order-preserving [Boj14]. Furthermore equivalence of the origin semantics of $2NTs$ is decidable in PSPACE [BMPP18] whereas equivalence of the classical semantics is known to be undecidable.

3.2 Definitions

In this section we will give a more formal definition of origin graphs and origin semantics. In literature there exists a lot of slightly different definitions for origin graph, in this thesis we follow a slightly modified definition from [DFL18].

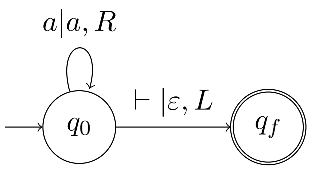
Definition 3.1. *On input alphabet* Σ *and output alphabet* Γ *an origin graph* $\sigma = (u, v, orig)$ *is a triple where*

- $(u, v) \in \Sigma^* \times \Gamma^*$ *the pair of input and output word.*
- $orig : \{1, 2, \ldots, |v|\} \to \{1, 2, \ldots, |u|\}$ *the origin function mapping every output position to an input position.*

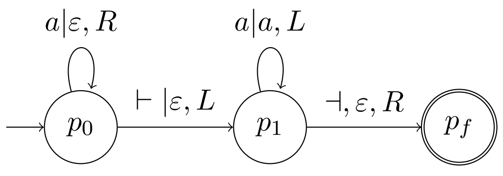
For an origin graph $\sigma = (u, v, orig)$ we introduce the functions $in(\sigma) = u$ and $out(\sigma) = v$ to regain the input word and the output word of the origin graph.

In this thesis we only discuss finite state transducers, however for different models there also exists natural ways of defining origin information. For the finite state transducers as discussed in the preliminaries the origin semantics are defined in the natural way as follows.

- **finite state transducers**
  The origin of an output position is the input position in which the reading head of the automaton is when that output position is produced. Given a run of T on input word $u = a_0 a_1 \ldots a_n$ with m transitions $(q_{i_1}, i_1) \to_{v_1} (q_{i_2}, i_2) \to_{v_2} \cdots \to_{v_m} (q_f, i_m)$, then the origin graph produced by this run $\sigma = (u, v, orig)$ has input u output $v = v_1 v_2 \ldots v_m$ where $v_j = \delta(q_j, a_{i_j})$ the output produced by the transitions. For every position in the subwords $x \in v_j$ the origin map $orig(x) = i_j$ the position of the tape head in which it was produced.



(a) Transducer T_{id}



(b) Transducer T_{id}^R

Figure 3.3: Definitions of T_{id}, T_{id^R}

(a) Origin graphs of T_{id^R}

Figure 3.4: Origin semantics of T_{id}, T_{id^R}

Definition 3.2. ***The origin semantics*** *of a transducer T from Σ to Γ denoted by $[\![T]\!]_o$ is the set of origin graphs produced by T.*

Two transducers are called *origin equivalent* when they produce the same origin graphs. Intuitively the equivalence relation of these new semantics are smaller than the classical semantics. Where transducers are equal under the classical equivalence if they compute the same relation, they are only equal under origin semantics if they also have the same origins.

For transducers classical equivalence does not imply origin equivalence.

Lemma 3.3. *For two transducers T_1, T_2 :*

$$[\![T_1]\!] = [\![T_2]\!] \not\Rightarrow [\![T_1]\!]_o = [\![T_2]\!]_o$$

Proof. Follows from Example 3.5. □

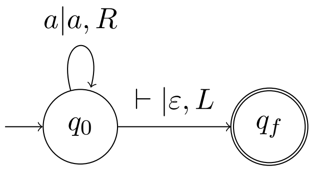
The reverse of this implication is true. One can see that origin semantics really are a refinement of the classical semantics.

Lemma 3.4. *For two transducers T_1, T_2:*

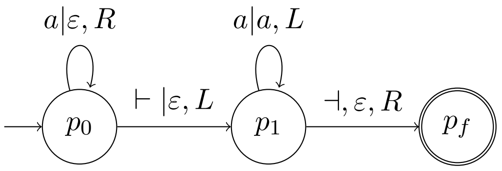
$$[\![T_1]\!]_o = [\![T_2]\!]_o \Rightarrow [\![T_1]\!] = [\![T_2]\!]_o$$

Proof. By definition if two origin graphs are equal, they have the same input and output. □

Example 3.5. *Two transducers T_{id} and T_{id^R} on the singleton input and output alphabet $\Sigma = \Gamma = \{a\}$. Where T_{id} computes the identity from left to right, and T_{id^R} computes the identity from right to left. Both transducers are given by the following diagrams*

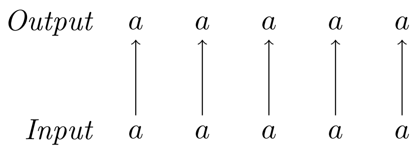


Transducer T_{id}

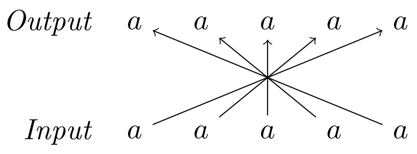


Transducer T_{id^R}

These two transductions both describe the relation $[\![T_{id}]\!] = [\![T_{id^R}]\!] = \{(a^n, a^n) \mid n \in \mathbb{N}\}$. However the origin graphs produced are different. For example on the input a^5:



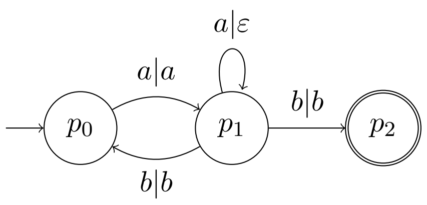
Origin graph by T_{id}



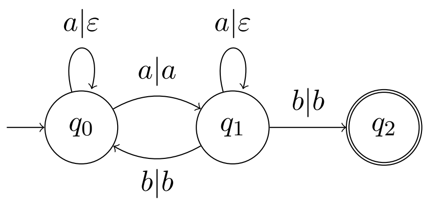
Origin graph by T_{id^R}

To demonstrate multiple interesting origin behaviors, we give three transducers $T_{first}, T_{random}, T_{last}$ which on alphabets $\Sigma = \Gamma = \{a, b\}$ all compute the function $(a^+b)^n \mapsto (ab)^n$.

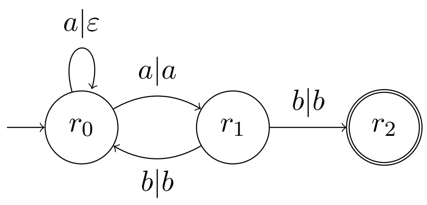
Example 3.6. *Given three 1NTs as $T_{first}, T_{random}, T_{last}$ on alphabets $\Sigma = \Gamma = \{a, b\}$ given in Figure 3.5. Possible origin graphs are given in Figure 3.6.*



(a) Transducer T_{first}



(b) Transducer T_{random}



(c) Transducer T_{last}

Figure 3.5: Transition diagrams for $T_{first}, T_{random}, T_{first}$

.

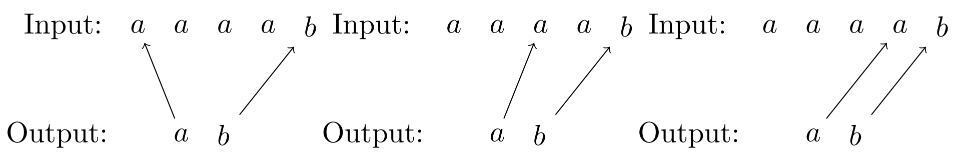


Figure 3.6: Possible origin graph produced by $T_{first}, T_{random}, T_{last}$ respectively.

While these examples are all equivalent $[\![T_{first}]\!] = [\![T_{last}]\!] = [\![T_{random}]\!]$ they are all different in origin semantics. The only origin containment relations that hold are $[\![T_{first}]\!]_o \subseteq [\![T_{random}]\!]_o$ and $[\![T_{last}]\!]_o \subseteq [\![T_{random}]\!]_o$ since the first and last are also one of the random cases.

Chapter 4

Resynchronizations

4.1 Resynchronizations

Since a transduction can have different origin information while maintaining the same originless semantics, one can be interested in a relation between origin graphs with different origins. We call a relation that relates origin graphs with different origin information a resynchronization. In recent work a framework for resynchronizations and the containment problem for regular transducers up to resynchronization is discussed [BMPP18]. A framework is proposed which consists of regular resynchronizations defined with MSO formulas. This is based on the idea of resynchronizations for rational transducers as described in [FJLW16]. Because of the regular character of MSO definable languages these resynchronizers are sometimes also called *regular* resynchronizers. In this section we will discuss these regular resynchronizations.

Resynchronizers describe a relation on origin graphs that share the same input and output but the origin information can be different in a described way. The main part of regular resynchronizers is an MSO sentence that describes if an origin is allowed to 'move' from a position to another position. The rest is useful to deal with non-determinism and to add some expressiveness.

The resynchronizer is equipped with output parameters $\bar{O}$ to express regular properties on the output word. Then there is an MSO formula γ over the input word that is responsible for moving origin mappings. This formula can not access the output types $\bar{O}$ directly but uses output types to know the output parameters for a single output positions. If x is an output position labelled with $a \in \Gamma$ and $b_1, b_2, \ldots, b_m \in \mathbb{B}$ where $b_i = 1 \iff x \in O_i$ then we note the output type $\tau \in \Gamma \times \mathbb{B}^m$ to be $\tau = (a, b_1, b_2, \ldots, b_m)$.

Definition 4.1. *An MSO resynchronizer R is a tuple $(\alpha, \beta, \gamma, \delta)$ with n input parameters $\bar{I} = (I_1, \ldots, I_n)$ and m output parameters $\bar{O} = (O_1, \ldots, O_m)$:*

- *$\alpha(\bar{I})$ an MSO formula over the input word with input parameters $\bar{I}$.*

- $\beta(\bar{O})$ *an MSO formula over the output word with output parameters* $\bar{O}$.

- $\gamma(\tau)(\bar{I}, x, y)$ *an MSO formula over the input word for every output-type* $\tau \in \Gamma \times \mathbb{B}^n$ *for* n *output parameters, and free variables* x, y *indicate the source origin and target origin.*

- $\delta(\tau, \tau')(\bar{I}, z, z')$ *an MSO formula over the input that constrains the origins* z, z' *of two consecutive output positions with type* τ, τ'.

The input and output parameters $\bar{I}$ and $\bar{O}$ are used to express regular properties of the input word and output word, using the MSO formula α and β. They also add expressiveness to deal with non-determinism to keep a consistent pick between the different input positions, as demonstrated in example 4.6.

The γ and δ formulas describe how origins are allowed to change. They both only are over the input part of the origin graph, and can only access fixed regular properties of the output word via the type of the output parameters which are encoded in the type of the output position.

Given an origin graph σ input parameters and output parameters are a subset of the input or output positions respectively. Thus on σ an interpretation on n input parameters are given by $\bar{I} = I_1, \dots, I_n$ where all $i \in [1, n], I_i \subseteq dom(in(\sigma))$ and the same for m output variables $\bar{O} = O_1, \dots, O_m \subseteq dom(out(\sigma))$. For every position $x \in dom(out(\sigma))$ we say the output type of x is $\tau = (a, b_1, b_2, \dots, b_m) \in \Gamma \times \mathbb{B}^m$. Where a is the label of the output position x and for $1 \leq i \leq m$ if $x \in O_i$ then $b_i = 1$ otherwise $b_i = 0$.

Definition 4.2. *An MSO resynchronizer* R *describes a relation* $[\![R]\!]$ *on origin graphs, defining the relation on origin graphs* $\sigma = (u, v, orig)$ *and* $\sigma' = (u', v', orig'))$ *defined by* $(\sigma, \sigma') \in [\![R]\!]$ *if and only if* $u = u'$ *and* $v = v'$*, there exists input parameters* $\bar{I} = I_1, \dots, I_n \subseteq dom(u)$, $\bar{O} = O_1, \dots, O_n \subseteq dom(v)$, *and the following requirements hold.*

- $(u, \bar{I}) \models \alpha$

- $(v, \bar{O}) \models \beta$

- *For every output position* $x \in dom(v)$ *with the type of* x $type(x) = \tau$ *and the origin of* x *in* σ *is* $orig(x) = y$ *while in* σ' *the origin is* $orig'(x) = z$*, we have* $(u, \bar{I}, y, z) \models \gamma(\tau)$

- *For all consecutive pairs of output positions* $x, x' \in v$ *with output type* τ, τ' $(u, \bar{I}, orig'(x), orig'(x')) \models \delta(\tau, \tau')$.

The δ formula is used to constrain the relation, which is of little use in this thesis so for the most part we will assume $\delta = \top$ so that the relation is only restricted on the movement of the origins by γ.

4.2 Examples

To illustrate the functionalities of the relation these resynchronizers define we show some examples. Since we don't use the δ function in this thesis all examples have $\delta = \top$, for an example illustrating the usefulness of the δ function see [BMPP18]. In the example origin graphs arrows will represent the origins in σ, and dotted lines the origins in σ'.

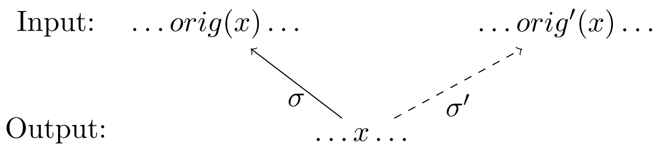
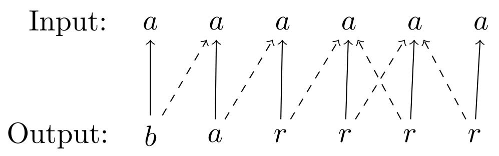


Figure 4.1: Notation to express $(\sigma, \sigma') \in [\![R]\!]$

Example 4.3. *The resynchronizer without parameters $R_{univ} = (\alpha, \beta, \gamma, \delta)$ with $\alpha = \beta = \top$ and for every output type $\tau, \tau' \in \Gamma$, $\gamma(\tau) = \top$ and $\delta(\tau, \tau') = \top$. This is called the universal resynchronizer and will resynchronize any two pair of origin graphs that share the same input and output.*

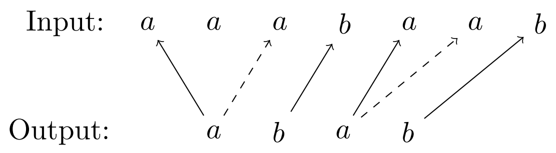
Example 4.4. *The resynchronizer without parameters $R_{\pm 1} = (\top, \top, \gamma, \top)$ that only use γ to shift origin by 1 place. For every output type $\tau \in \Gamma$ the formula $\gamma(\tau)(y, z) = s(y, z) \ \vee \ s(z, y)$*



Example 4.5. *A resynchronizer without parameters that moves the origin of the first letter of a sequence of a's to the last letter. $R = (\top, \top, \gamma, \top)$. Where for every output type τ the formula*

$$
\begin{aligned}
\gamma(\tau)(y, z) = \ & (y = z) \wedge b(y) \\
& \vee \\
& (\ y \leq z \\
& \quad \wedge (\forall x \in [y, z]. a(x)) \\
& \quad \wedge (\forall x. s(x, y) \rightarrow \neg a(x)) \\
& \quad \wedge (\forall x. s(z, x) \rightarrow \neg a(x)) \\
&)
\end{aligned}
$$

Note that in this resynchronization on the identity function over $(a^* b)^*$ can have arbitrary many output positions for which the origin moves arbitrary many places.



Example 4.6. *The resynchronizer from Example 4.5 but with one input parameter that chooses a random a in the sequence of a's to redirect to.* $R_{random} = (\alpha, \top, \gamma, \top)$*, where* α *ensure that* I_1 *only has one input position in every sequence of a's and* γ *redirects the first a to this chosen a.*

- $\alpha(I_1) = \forall x.\ (a(x) \land x \in I_1) \to \forall y.\ (a(y) \land \forall z \in [x, y].a(z)) \to \neg y \in I_1 \lor y = x$
- $\gamma(\tau)(y, z) = a(y) \land \neg a(y-1) \to z \in I_1 \land (\forall w.y \leq w \leq z \lor z \leq w \leq y \to a(w))$

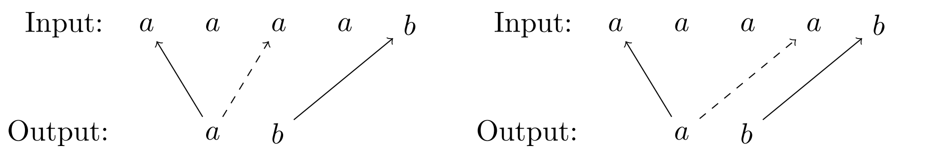


Figure 4.2: Multiple origin graphs in R_{random}

4.3 Containment up to resynchronization

Using these resynchronizers one can expand the notion of origin containment of transducers, by talking about the origin containment up-to some resynchronizer R. In contrast to containment of the classic semantics this new notion for containment for $2NTs$ is decidable in PSPACE as shown in [BMPP18].

Definition 4.7. *For a given resynchronizer R and two transducers* T_1, T_2 *we say* $T_1 \subseteq_o R(T_2)$ *if for every origin graph* $\sigma' \in [\![T_1]\!]_o$ *there exists* $\sigma \in [\![T_2]\!]_o$ *such that* $(\sigma, \sigma') \in [\![R]\!]$*.*

In other words this means that T_1 is contained in the resynchronization expansion of T_2. Verbally we say T_1 is contained in T_2 up to R.

Example 4.8. *Let* $T_{first}, T_{random}, T_{last}$ *be the transducers from Example 3.6 and* $R_{first\text{-}to\text{-}last}, R_{random}$ *be the resynchronizers from Examples 4.5,4.6. Then we have* $T_{last} \subseteq R_{first\text{-}to\text{-}last}(T_{first})$ *and* $T_{random} \subseteq R_{random}(T_{first})$*. All transducers compute the same relation, one only needs to observe the replacements done by the resynchronizers.*

We will show that this new notion of containment is not transitive and not reflexive. For a fixed resynchronizer R, it might not be the case that for every transducer T, we have that $T \subseteq_o R(T)$. Also it is not transitive. Given transducers T_1, T_2, T_3 and it is the case that $T_1 \subseteq_o R(T_2)$ and $T_2 \subseteq_o R(T_3)$ it might not be the case that $T_1 \subseteq_o R(T_3)$. This can be seen by the example resynchronizer $R_{\pm 1}$ from Example 4.4, that shifts the origin with one position.

4.4 Bounded resynchronizers

Note that the universal resynchronizer from Example 4.3 will relate any two graphs that share the same input and output. This will cause the containment relation to boil down to originless equivalence. Which will render containment up to resynchronization undecidable. Since $T_1 \subseteq_o R_{univ}(T_2)$ if and only if $[\![T_1]\!] \subseteq [\![T_2]\!]$ and this classical containment problem is known to be undecidable.

If we restrict the containment problem to bounded resynchronization the problem becomes decidable [BMPP18]. The intuition of bounded resynchronization is that for every input position z only a bounded number of positions get redirected to z.

Definition 4.9. *(Boundedness) A regular resynchronizer $R = (\alpha, \beta, \gamma, \delta)$ has bound k if for all inputs u, input parameters $\bar{I}$, output types $\tau \in \Gamma \times \mathbb{B}^n$, and target positions $z \in dom(u)$, there are at most k distinct valid source positions $y_1, \ldots y_k \in dom(u)$ formally $(u, \bar{I}, y_i, z) \models \gamma(\tau)$ for all $i = 1, \ldots, k$. A regular resynchronizer is bounded if it is k-bounded for some $k \in \mathbb{N}$ and unbounded if there is no such $k \in \mathbb{N}$.*

Of the examples from Section 4.2 only the first one is not bounded. Example 4.6 is interesting since there is no bounded resynchronizer without parameters that describes this relation.

Since our research focuses primarily on bounded regular resynchronizers when not specified by saying resynchronizer we mean bounded regular resynchronizer. The question whether a given regular resynchronizer R is bounded is decidable [BMPP18].

Chapter 5

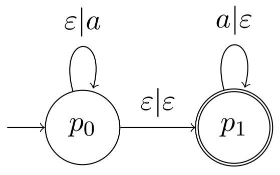
Resynchronizability relation

In this chapter we will be interested in the containment up to an unknown bounded resynchronizer. Resynchronizers form a fixed relation between origin graphs, but in this chapter we are interested whether there exists a bounded resynchronizer that relates the origin graphs of two transducers. This is what we call resynchronizability, transducers are resynchronizable if the origin graphs they produce can be related by a bounded resynchronizer. The way the computation is done might be different, but not too incompatible.

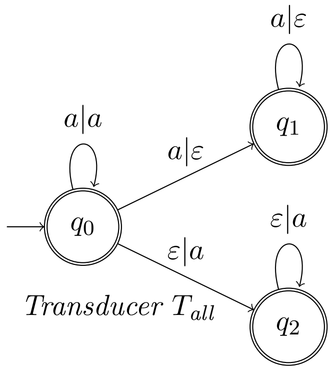
Definition 5.1. *(Resynchronizability) We define $\preceq : T \times T$ where $T_1 \preceq T_2$ if and only if there exists a bounded resynchronizer R such that $T_1 \subseteq_o R(T_2)$.*

This relation is an interesting one since it expresses something intermediate of the classical and origin containment. From the definition it follows that for two transducers T_1 and T_2, if $T_1 \preceq T_2$ then they are also contained in the classical sense $[\![T_1]\!] \subseteq [\![T_2]\!]$. The opposite is not the case as can be seen by Example 5.2. Let us recall that for $2NTs$ classic containment was undecidable and for origin semantics it is decidable. It could be the case that this new containment relation forms a decidable relation that is bigger than origin containment.

Example 5.2. *Let us give an example of two transducers T_{first}, T_{all} with $[\![T_{\mathit{first}}]\!] = [\![T_{\mathit{all}}]\!] = \{(a^n, a^m) \mid n, m \in \mathbb{N}\}$, and $T_{\mathit{all}} \preceq T_{\mathit{first}}$ but $T_{\mathit{first}} \not\preceq T_{\mathit{all}}$.*

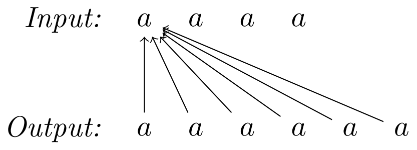


Transducer $T_{\textit{first}}$

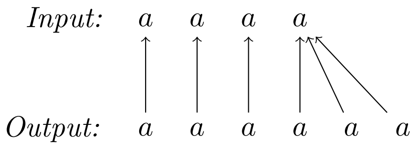


Transducer $T_{\textit{all}}$

Indeed, we have $T_{\textit{all}} \subseteq R(T_{\textit{first}})$ *where* R *uses only* $\gamma(x, y) = (x = \textit{first})$*, which is bounded. However, if we for some* R *had* $T_{\textit{first}} \subseteq R(T_{\textit{all}})$*, then* R *must redirect arbitrary many positions to the first one, and therefore it cannot be bounded. As can be seen in the origins graphs to be related:*



$(a^4, a^6, \textit{orig}) \in [\![T_{\textit{first}}]\!]_o$



$(a^4, a^6, \textit{orig}') \in [\![T_{\textit{all}}]\!]_o$

The argument above is a bit informal but later with Lemma 5.10 we provide a method to formally prove this statement.

The problem of finding an unknown resynchronization is also studied in [BKM+19] where they call it the synthesis problem for resynchronizers. A $2NT$ T is called *unambigious* if for every input word there is at most one possible run of T.

Theorem 5.3. *[BKM+19] Given two unambigious 2NTs* T_1, T_2 *the problem whether* $T_1 \preceq T_2$ *is decidable.*

For general $2NT$ this problem is left open, we show in Theorem 5.20 that it is in fact undecidable. Before showing the undecidability result we will show this relation forms a preorder, and introduce a tool to witness the non-existence of a bounded resynchronizer to satisfy a relation.

5.1 Relational properties

In this section we will prove some relational properties for the resynchronization relation. We will prove the relation forms a preorder, and we will prove that it is different from classical equivalence.

For many properties of this relation, we don't need to regard the δ function in the resynchronizer, since this only make the relation smaller.

Lemma 5.4. *Given a resynchronizer $R = (\alpha, \beta, \gamma, \delta)$ and two transducers T_1, T_2 such that $T_1 \subseteq_o R(T_2)$, then the containment $T_1 \subseteq_o R'(T_2)$ also holds with resynchronizer $R' = (\top, \top, \gamma, \top)$.*

Proof. Note that the obtained relation R' is bigger: $[\![R]\!] \subseteq [\![R']\!]$. Now for every pair $\sigma' \in [\![T_1]\!]_o$ there exists a corresponding pair $(\sigma, \sigma') \in R$ such that $\sigma \in [\![T_2]\!]_o$, we know that $(\sigma', \sigma) \in R$ so also $(\sigma, \sigma') \in R'$ which implies $T_1 \subseteq_o R'(T_2)$. □

Given we can leave out the functionality of the δ filter function we can show that the class of bounded resynchronizers is closed under composition. This will be a key ingredient in proving the transitivity of our $\preceq$ relation.

Lemma 5.5. *Given two bounded resynchronizers $R_1 = (\alpha_1, \beta_1, \gamma_1, \top)$, $R_2 = (\alpha_2, \beta_2, \gamma_2, \top)$ it is possible to construct a bounded resynchronizer R_3 such that $[\![R_3]\!] = [\![R_1]\!] \circ [\![R_2]\!]$*

Proof. Given $R_1 = (\alpha_1, \beta_1, \gamma_1, \top)$ and $R_2 = (\alpha_2, \beta_2, \gamma_2, \top)$ we construct $R_3 = (\alpha_3, \beta_3, \gamma_3, \top)$ such that $[\![R_3]\!] = [\![R_1]\!] \circ [\![R_2]\!]$.

Both α_1, α_2 have some input parameters $\bar{I}_1 = (I_1, \ldots, I_n), \bar{I}_2 = (I'_1, \ldots, I'_n)$. These act as MSO predicates over the input string $in(\sigma)$, since in these resynchronizers the input part of the origin graph does not change at all we can simply combine these input parameters $\bar{I}_3 = (I_1, \ldots, I_n, I'_1, \ldots I'_n)$. Now construct α_3 such that $(in(\sigma), \bar{I}_3) \models \alpha_3$ implies that $(in(\sigma), \bar{I}_1) \models \alpha_1$ and $(in(\sigma), \bar{I}_2) \models \alpha_3$

$$\alpha_3(\bar{I}_3) = \alpha_1(I_1, \ldots, I_n) \wedge \alpha_2(I'_1, \ldots, I'_n)$$

Similar to α_3 construct β_3 based on the truth values on β_1, β_2. For output parameters $\bar{O}_1 = (O_1, \ldots O_n), \bar{O}_2 = O'_1, \ldots O'_m$ we construct output parameters $\bar{O}_3 = (O_1, \ldots, O_n,\ O'_1, \ldots O'_m)$.

$$\beta_3(\bar{O}_3) = \beta_1(\bar{O}_1) \wedge \beta_2(\bar{O}_2)$$

We can define γ_3 directly by quantifying of input positions and finding an *intermediate* position. Since we want to track an interpretation of the output parameters $\bar{O}_3$ in the type τ So we need to massage that a bit. For given $\tau = \Gamma \times \mathbb{B}^{n+m}$ where τ gives a valuation for both $\bar{O}_1, \bar{O}_2$ we call τ_1 the projection on the first part containing the valuation of $\bar{O}_1$ and τ_2 the equivalent for $\bar{O}_2$

$$\gamma_3(\tau)(\bar{I}_3, x, z) = \exists y. \gamma_1(\tau_1)(\bar{I}_1, x, y) \wedge \gamma_2(\tau_2)(\bar{I}_2, y, z)$$

This completes the construction, we now verify that $[\![R_3]\!] = [\![R_2]\!] \circ [\![R_1]\!]$ and that it is bounded.

$[\![R_3]\!] \subseteq [\![R_2]\!] \circ [\![R_1]\!]$: for any (σ_1, σ_3) by definition for some input parameters $\bar{I} = (I_1, \ldots, I_n, I'_1, \ldots, I'_m)$ and output parameters $\bar{O} = (O_1, \ldots, O_n, O'_1, \ldots, O'_m)$ the following holds

- $(in(\sigma_1), \bar{I}) \models \alpha_3$ equivalently $(in(\sigma_1), \bar{I}) \models \alpha_1(I_1, \dots, I_n) \wedge \alpha_2(I'_1, \dots, I'_m)$ which implies $(in(\sigma_1), (I_1, \dots I_n)) \models \alpha_1$ and $(in(\sigma_1), I'_1, \dots, I'_m) \models \alpha_2$
- $(out(\sigma_1), \bar{O}) \models \beta_3$ equivalently to previous case $(out(\sigma_1), (O_1, \dots O_n) \models \beta_1$ and $(out(\sigma_1), (O'_1, O'_m) \models \beta_2$
- We know for pairs of output type τ and output position z there was some souce x for this target which satisfy $(in(\sigma_1), \bar{I}, x, z) \models \gamma_3(\tau)$. Rewritten with the definition of γ_3 we know there exists a y such that $(in(\sigma_1), (I_1, \dots, I_n), x, y) \models \gamma_1(\tau)$ and $(in(\sigma_1), (I'_1, \dots, I'_m), y, z) \models \gamma_2(\tau)$

Combining this we know that there exists a σ_2 such that $(\sigma_1, \sigma_2) \in [\![R_2]\!]$ and $(\sigma_2, \sigma_3) \in [\![R_2]\!]$ thus $(\sigma_1, \sigma_3) \in [\![R_2]\!] \circ [\![R_1]\!]$.

$[\![R_1]\!] \circ [\![R_2]\!] \subseteq [\![R_3]\!]$: if there exists $(\sigma_1, \sigma_2) \in [\![R_1]\!]$ and $(\sigma_2, \sigma_3) \in [\![R_2]\!]$

- $(in(\sigma_1), (I_1, \dots I_n)) \models \alpha_1$ and $(in(\sigma_1), I'_1, \dots, I'_m) \models \alpha_2$ implies that $(in(\sigma_1), \bar{I}) \models \alpha_1(I_1, \dots, I_n) \wedge \alpha_2(I'_1, \dots, I'_m)$ or $(in(\sigma_1), \bar{I}_3) \models \alpha_3$ equivalently
- Assuming $(out(\sigma_1), (O_1, \dots O_n) \models \beta_1$ and $(out(\sigma_1), (O'_1, O'_m) \models \beta_2$ we know $(out(\sigma_1), \bar{O}_3) \models \beta_3$ similar to the α case.
- For every output position of type $\tau = \Gamma \times \mathbb{B}^{n+m}$ if this output position in origin graph σ_1 has origin x then $\gamma_1(\tau_1)(\bar{I}_1, x, y)$ for some y that is the origin of this output in σ_2, by assumption of R_2 there exists some position z such that $\gamma_2(\tau_2)(\bar{I}_2, y, z)$ where z is the origin in σ_3. Which are exactly the two conditions for γ_3,so conclude $(in(\sigma_1), \bar{I}_3, x, z) \models \gamma_3(\tau)$

Now we know that $(\sigma_1, \sigma_3) \in [\![R_3]\!]$ thus $[\![R_1]\!] \circ [\![R_2]\!] \subseteq [\![R_3]\!]$, so also $[\![R_3]\!] = [\![R_2]\!] \circ [\![R_1]\!]$.

This relation is also bounded since R_1 and R_2 are also bounded. So we know that for some $n, m \in \mathbb{N}$ R_1 is n-bounded and R_2 is m-bounded. These bounds imply that for every origin graph σ, output type τ, input parameters $\bar{I}$ and target position z, there are at most m sources $y_1, \dots, y_m \in in(\sigma)$ such that $(in(\sigma), \bar{I}, y_i, z) \models \gamma_2(\tau_2)$ for all $1 \leq i \leq m$. For each of these positions y_i there are at most n sources x_i such that $(in(\sigma), \bar{I}, x_i, y_i) \models \gamma_1(\tau_1)$ so for every output position z there are at most bounded $n * m$ input positions x_i such that there exists an y_i for which $\gamma_1(x_i, y_i) \wedge \gamma_2(y_i, z)$ which implies there are only bounded number of sources x_i such that $(in(\sigma), \bar{I}, x_i, z) \models \gamma_3(\tau)$. $\square$

Lemma 5.6. *$\preceq$ is transitive, if $T_1 \preceq T_2$ and $T_2 \preceq T_3$, then $T_1 \preceq T_3$.*

Proof. Assume $T_1 \subseteq_o R_1(T_2)$ and $T_2 \subseteq_o R_2(T_3)$ for bounded resynchronizers R_1, R_2 that do not use the δ part, this does not lose generality by lemma 5.4. Then $T_1 \subseteq_o (R_1 \circ R_2)(T_3)$. We know for every $\sigma_1 \in [\![T_1]\!]_o$ there is a $\sigma_2 \in [\![T_2]\!]_o$ such that $(\sigma_2, \sigma_1) \in [\![R_1]\!]$ and for every $\sigma_2 \in [\![T_2]\!]_o$ there is a

$\sigma_3 \in \llbracket T_3 \rrbracket_o$ such that $(\sigma_3, \sigma_2) \in \llbracket R_1 \rrbracket$. So also for every $\sigma_1 \in \llbracket T_1 \rrbracket_o$ there is a $\sigma_3 \in \llbracket T_3 \rrbracket_o$ such that $(\sigma_3, \sigma_1) \in \llbracket R_1 \rrbracket \circ \llbracket R_2 \rrbracket$ □

Note that this transitivity property is a slightly different than the question whether regular resynchronizations are closed under composition, that is given two regular resynchronizations R_1, R_2 is it possible to construct R_3 such that $\llbracket R_3 \rrbracket = \llbracket R_1 \rrbracket \circ \llbracket R_2 \rrbracket$. In order to prove this one would need to drop the assumption that $\delta = \top$ in both R_1, R_2. This problem is still open [BMPP18] and might be interesting for further work.

We define $T_1 \sim_D T_2$ if $T_1 \preceq T_2$ and $T_2 \preceq T_1$, which is the equivalence closure of resynchronizability. This equivalence relation is not the same as the originless semantic equivalence $\llbracket T_1 \rrbracket = \llbracket T_2 \rrbracket$, since it is possible to construct a pair of transductions T_1, T_2 such that $\llbracket T_1 \rrbracket = \llbracket T_2 \rrbracket$ but $T_1 \not\preceq T_2$. To show this, we make use of Example 3.5. On a one letter input and output alphabet $\Sigma = \Gamma = \{a\}$ take the transducer T_{id} that outputs the identity from left to right, and T_{id^R} that outputs the identity from right to left. By this definition we know that $\llbracket T_{id} \rrbracket = \llbracket T_{id^R} \rrbracket$ but we will see that $\llbracket T_{id} \rrbracket_o \neq \llbracket T_{id^R} \rrbracket_o$

Lemma 5.7. *For the transducers T_{id} and T_{id^R} there is no bounded regular resynchronizer R such that $T_{id} \subseteq_o R(T_{id^R})$.*

Proof.

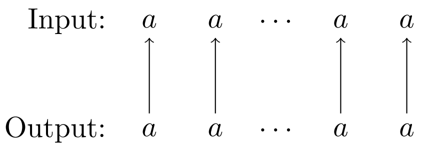
1. Fix an arbitrary $R = (\alpha, \beta, \gamma, \delta)$, with n_i input parameters and m_o output parameters such that $T_{id} \subseteq_o R(T_{id^R})$. We can assume $\alpha = \beta = \delta = \top$ using lemma 5.4.

2. For an arbitrary $k \in \mathbb{N}$ we show that there is a length $N \in \mathbb{N}$, input parameters $\bar{I} \in (\mathbb{B}^{n_i})^N$, and an output-type $\tau \in \Gamma \times \mathbb{B}^{m_o}$. Such that there are k distinct sources $x_1, \ldots x_k \in [1, N]$, and one target position $y \in [1, N]$ such that $\forall i \in [1, k]$, $(a^N, \bar{I}, x_i, y) \models \gamma(\tau)$. This would show that R is unbounded.

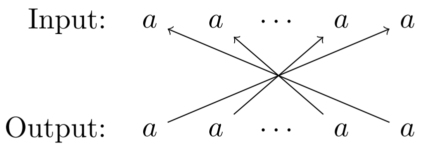
3. For all output types τ the formula $\gamma(\tau)$ is a MSO sentence with variables $\bar{I}$ and first order variables x, y modeling source and target. For every type τ this MSO-sentence recognizes a regular language over the alphabet $\Sigma \times \mathbb{B}^{n_i+2}$, and therefore there is a monoid M_τ and a morphism $f_\tau : \Sigma \times \mathbb{B}^{n_i+2} \to M_\tau$ recognizing the language of $\gamma(\tau)$, where the booleans encode the n input parameters $\bar{I}$ and the source and target x and y.

   Pick the type τ_i with the largest corresponding monoid M_{τ_i}, and let $\pi : M_{\tau_i}^* \to M_{\tau_i}$ be the evaluation morphism in M_{τ_i}. Using the Simon's factorization forest theorem[Sim90], we know that there is $H \in \mathbb{N}$ such that for any word $u = m_1 m_2 \ldots m_H \in M^H$, there is an idempotent $e \in M_{\tau_i}$ and a factorization of u into $u = m e_1 e_2 \ldots e_{k+2} m'$ with $m = m_1 \ldots m_{i_1}, e_j = m_{i_j+1} m_{i_j+2} \ldots m_{i_{j+1}}$ and $m' = m_{i_{k+2}+1} \ldots m_H$, such that for all $j \in [1, k+2]$, $\pi(e_j) = e$. This property also holds for all output types τ on the monoid M_τ, as we chose M_{τ_i} of maximal size, and the bound H increases with the size of the monoid.

4. We consider the word a^N with $N = 2^{m_o+1}(H+1)$. Let σ, σ' be the origin graphs produced by T_{id} and T_{id^R} on a^N.


Input: a a $\cdots$ a a
Output: a a $\cdots$ a a

(a) Shape of $\sigma \in [\![T_{id}]\!]_o$.


Input: a a $\cdots$ a a
Output: a a $\cdots$ a a

(b) Shape of $\sigma' \in [\![T_{id^R}]\!]_o$.

5. Since σ' is the only graph in $[\![T_{id^R}]\!]_o$ with same input as σ, and we assumed $T_{id} \subseteq_o R(T_{id^R})$, then we must have $(\sigma', \sigma) \in R$. Thus there exists input parameters $\bar{I}$ and output parameters $\bar{O}$ witnessing that $(\sigma', \sigma) \in R$.

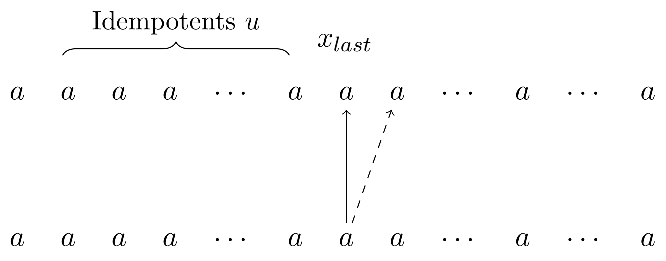
6. Pick the output type τ which occurs the most in the first half of the word. The types τ are element of $\Gamma \times \mathbb{B}^{m_o}$ for the fixed m_o output parameters, so since $|\Gamma| = 1$ there are 2^{m_o} output types τ_i. The τ we pick that occurs the most in the first half occurs at least $\frac{\frac{N}{2}}{2^{m_o}} = \frac{2^{m_o}(H+1)}{2^{m_o}} = H+1$ times in the first half of the word. In all the positions x where the output type is τ, we have by the shape of the reversal of the origin arrows that $(a^N, \bar{I}, x, N-x) \models \gamma(\tau)$.

7. In the first half of the word pick the last element x_{last} with type τ we know that $(a^N, \bar{I}, x_{last}, N - x_{last}) \models \gamma(\tau)$. Let $w \in \Sigma \times \mathbb{B}^{n_i+2}$ be the encoding of $(a^N, \bar{I}, x_{last}, N - x_{last})$. This word w will be accepted by the monoid M_τ with morphism $\hat{f}_\tau$.

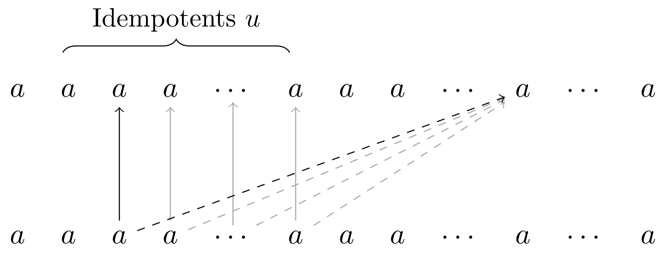
8. Split the word w in parts $w = w_1 \ldots w_l$ where all the blocks have exactly one position with output type τ. In the first half of the word there are at least $H+1$ blocks since there were at least that much occurrences of output positions with type τ. Before the position x_{last} there are at least H blocks $w_1, w_2, \ldots, w_H$. Which under the morphism f_τ form a word on the monoid M_τ.

9. Using Simon's factorization theorem, there is an idempotent $e \in M_\tau$ a factor $u = u_1 u_2 \ldots u_{k+2}$ in the first half of w, and a strictly increasing sequence $i_1, i_2 \ldots i_{k+2}$ such that each u_j is of the form $w_{i_j+1} \ldots w_{i_{j+1}}$, and $f^*_\tau(u_j) = e$.

   The following steps will modify the word w the idea is demonstrated in this picture where the solid arrow represents the source and the dashed arrow the target:



initial w



editted w' with possible new sources

10. By construction there is a position x' with type τ in u_2, there is a corresponding element $y = N - x'$ in the second half of the word such that $(a^N, \bar{I}, x', y) \models \gamma(\tau)$. We pick the corresponding word $w' \in (\Sigma \times \mathbb{B}^{n_i+2})^*$ that is accepted by monoid M_τ. The subword $u = u_1u_2 \dots u_{k+2}$ is now changed to $u_1u_2'u_3 \dots u_{k+2}$, where u_2' is the same as u_2 only the boolean representing the source will be 1 in position x' and for $i \neq 2$ we still have $\hat{f}_\tau(u_i) = e$ the idempotent element of M.

11. Change $u_1u_2'u_3 \dots u_{k+2}$ to $u' = u_2u_2'u_2^k$. this word is still accepted since it does not change the element of the monoid M_τ under the morphism. It can change the length of the word and the input parameters $\bar{I}$, but since it is projected on the same element in the monoid M, it is accepted. This new word w'' possibly changes in length and contents. Let the new length be $|w''| = N'$ and input parameters $\bar{I}'$.

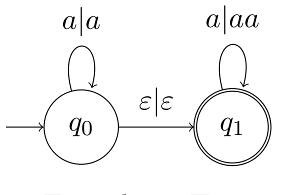
12. The word $a^{N'}$ with input parameters $\bar{I}'$ and target y has k distinct sources x for which $(a^{N'}, \bar{I}', x, y) \models \gamma(\tau)$. This can be done by swapping the block with the source position x in the sequence of idempotents without changing the element of the monoid under the morphism. Change the subword u' of w'' by $u_i = u_1u_2^iu_2'u_2^{k-i}$ for $0 \leq i \leq k$. All these different subwords u_i have the same length and input parameters, and a different source positions x_i so for all these positions $(a^{N'}, \bar{I}', x_i, y) \models \gamma(\tau)$.

Since for every $k \in N$, R is not k-bounded there exists no bounded resynchronization R such that $T_{id} \subseteq_o R(T_{id^R})$. $\square$

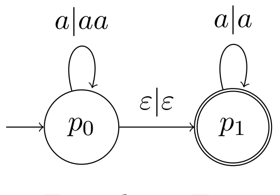
This proof demonstrates that there exists two transducers T_1, T_2 that are contained in the classical semantics $[\![T_1]\!] \subseteq [\![T_2]\!]$ but not in our resynchronizability relation $T_1 \preceq T_2$. This is interesting because it indicates that our new equivalence relation $\preceq$ is strictly between classic containment and origin containment, which could make it a decidable relation. However we show in Theorem 5.20 it is not.

The example transducer used in Lemma 5.7 use the property that the identity on the single letter alphabet is the same in reverse order. One might question if it is possible to construct a pair of classically equal transducers T_1, T_2 with $T_1 \not\sim_D T_2$ which do not use this two-way property. It turns out that it is possible to construct a pair of one-way non-deterministic transducers with these requirements as seen in example 5.8.

Example 5.8. *Consider the 1NT's $T_{one\text{-}two}, T_{two\text{-}one}$ on a unary alphabet $\Sigma = \Gamma = \{a\}$ which both will describe the relation $[\![T_{one\text{-}two}]\!] = [\![T_{two\text{-}one}]\!] = \{(a^n, b^m) \mid n, m \in \mathbb{N}$ and $n \leq m \leq 2 * n\}$*

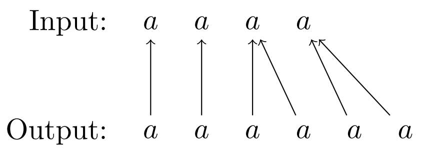


Transducer $T_{one\text{-}two}$

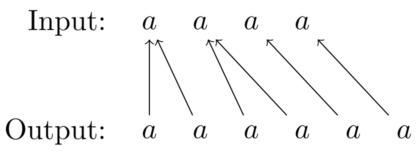


Transducer $T_{two\text{-}one}$

Transducer $T_{one\text{-}two}$ will first output single a's and then at some point non-deterministically go to a state from which it will only output aa for each a read from the input. The transducer $T_{two\text{-}one}$ will do the same but first outputting aa's and after that outputting a's. For any pair in the $(a^n, a^m) \in [\![T_1]\!]$ there is exactly one origin graph in $\sigma \in [\![T_{one\text{-}two}]\!]_o$ with $in(\sigma) = a^n, out(\sigma) = a^m$ and one origin graph $\sigma' \in [\![T_{two\text{-}one}]\!]_o$ with $in(\sigma') = a^n, out(\sigma') = a^m$. Examples of these origin graphs are given in Figure 5.2.



(a) Example $\sigma' \in [\![T_{\text{one-two}}]\!]_o$



(b) Example $\sigma \in [\![T_{\text{two-one}}]\!]_o$

Figure 5.2: Example origin graphs of $T_{\text{one-two}}, T_{\text{two-one}}$.

It can be proven in the way as Lemma 5.7 that $T_{\text{two-one}} \not\sim_D T_{\text{one-two}}$. We are interested in what we need in order to use this proof structure and see if we can generalize this.

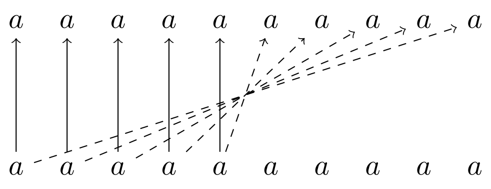
5.2 Desynchronized blocks

We have seen an example of a relation between origin graphs which can't be generated by a bounded resynchronizer. In order to classify when resynchronization is not possible we introduce a tool to recognize a pattern which implies the non-existence of a bounded resynchronizer. The proof of this tool is a generalization of the proof of Lemma 5.7.

Definition 5.9. *Given two origin graphs σ, σ' with $in(\sigma) = in(\sigma'), out(\sigma) = out(\sigma')$, a factor u of $in(\sigma)$ is called a desynchronized block, if the subgraph of u in σ has no shared output positions with the subgraph induced by u in σ'.*

In other words this means that in a pair of origin graphs σ, σ' if the input positions u form a desynchronized block, that the output produced by u in σ is produced by different input positions in σ'. We call the size of the desynchronized block the number of input positions in u that produce output.

Now an observation is that the resynchronization realizing Example 3.5 and 5.8 contain desynchronized blocks of arbitrary length. In the following picture a desynchronized block of size 5 is demonstrated in the origin graph relating the words (a^{10}, a^{10}), where the solid arrows are the origins in T_{id}, and the dashed arrows the origins in T_{id^R}.



desynchronized block T_{id}, T_{id^R}

Lemma 5.10. *Given a resynchronizer R, if for any number $d \in \mathbb{N}$ there is a pair of two origin graphs $(\sigma, \sigma') \in [\![R]\!]$ that have a desynchronized block of size greater than d. Then R is not bounded.*

Proof. This proof has the same style as the proof of Lemma 5.7, but in a more generalized way, for simplicity we don't deal with input and output parameters, a full proof is given in the appendix. Given an $R = (\alpha, \beta, \gamma, \delta)$ such that the first property holds. Assume $\alpha = \beta = \delta = \top$. Now we show that for any $k \in \mathbb{N}$ there exists $(\sigma, \sigma') \in R$, a target position y, and k distinct source positions $x_i \in \{x_1, x_2, \ldots, x_k\}$ such that $(x_i, y) \models \gamma$.

Take a monoid M,a morphism $f : \Sigma \times \mathbb{B}^2 \to M$ describing the regular language of γ. Let $\pi : M^* \to M$ be the evaluation morphism in M, there is a $H \in \mathbb{N}$ such that for any word $u = m_1 m_2 \ldots m_H \in M^H$, there is a idempotent element $e \in M$ and a factorization of u into $u = m e_1 e_2 \ldots e_k m'$

with $m = m_1 \dots m_{i_1}, e_j = m_{i_j+1} m_{i_j+2} \dots m_{i_{j+1}}$ and $m' = m_{i_{k+2}+1} \dots m_H$, such that for all $j \in [1, k+2]$, $\pi(e_j) = e$.

By assumption there exists a pair of origin graphs $(\sigma, \sigma') \in [\![R]\!]$ that have a desynchronized block u of size $H+1$. Thus there are $H+1$ positions in u that produce output and none of this output is produced by u in σ'. Split this word $u = u_1 u_2 \dots u_{H+1}$ with every subword u_i containing exactly one input position that has output. We choose as source input $x \in u_{H+1}$ the input position that produced output in u_{H+1}. Since u_{H+1} is a part of a desynchronised block we know that the corresponding target input position y for which $(in(\sigma), x, y) \models \gamma$ is not part of u. The word that is formed $w \in (\Sigma \times \mathbb{B}^2)^*$ has zeroes everywhere except in the input positions x marked as the source and y marked as the target position. So the sequence $u_1 u_2 \dots u_H$ in w is padded with all zeroes.

Given this word $w \in \Sigma \times \mathbb{B}^2$, since the sequence $u_1 \dots u_H$ contains at least H times a positions that produces output the evaluation under the morphism M there is a sequence of $v_1 \dots v_k \in (\Sigma \times B^2)^*$ for which the image $\hat{f}(v_i) = e_i$ are idempotent elements $e_1 e_2 \dots e_k \in M^*$, for which $\pi(e_i) = e$ and every word corresponding v_j contain at least one position producing output.

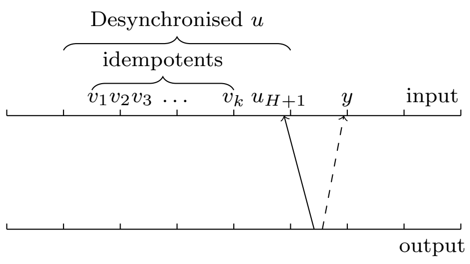
Now change w to w' by removing the original source and target position x, y. Placing the new source in v_2 replacing the sequence $v_1 v_2 \dots v_k$ by the sequence $v_1 v_2' v_3 \dots v_k$, where v_2' contains the new source input position x_0 , this position is in the desynchronised block, so there is a target positions y' such that $in(\sigma), x_0, y' \models \gamma$ pick this as target in the new word w'. This new word w' is accepted by the monoid M, since it is also a model of γ. In the next step change the sequence $v_1 v_2' v_3 \dots v_k$ in the sequence $v_2 v_2' v_2^{k-2}$ this changes the word w' in a word w'' but not the evaluation so this w'' is still accepted by M and thus a valid model for γ. Since $\hat{f}(v_2) = e$ the idempotent element you can move blocks of v_2 around v_2' producing k different positions $\pi_1(w''), x_i, y \models \gamma$, where $\pi_1(w'') \in \Sigma^*$ is projection on the first component. This construction is shown in Figure 5.3 where the arrows represent the source, and dashed arrows the target. Since k is arbitrary chosen this proves that this resynchronization is not bounded.

□

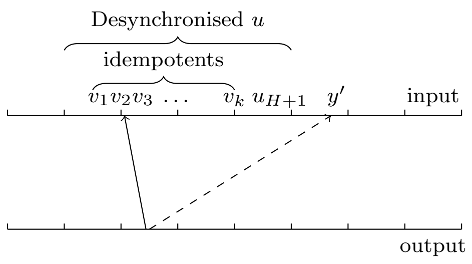
This tool for resynchronizers translate to a property for transducers. When we now there must exist pairs of origin graphs with unbounded desynchronized blocks in the resynchronization, it can not be bounded.

Lemma 5.11. *Given two transducers T_1, T_2 if for all $d \in \mathbb{N}$ there is an origin graph $\sigma \in [\![T_1]\!]_o$ pair such that for all graphs $\sigma' \in [\![T_2]\!]_o$ such that $in(\sigma) = in(\sigma')$, $out(\sigma) = out(\sigma')$ and (σ', σ) have a desynchronized block larger than d then $T_1 \not\preceq T_2$.*

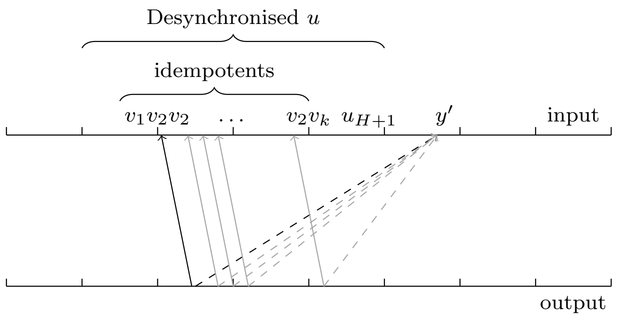
Proof. Assume there would exists a bounded resynchroniser R witnessing $T_1 \preceq T_2$. For any d there is an origin graph $\sigma \in [\![T_1]\!]_o$ such that for all



(a) word with source u_{H+1} target y



(b) word with source u_{H+1} target y'



(c) All possible sources x_i

Figure 5.3: Edits of w with desynchronized block u

$\sigma' \in [\![T_2]\!]_o$ that have the same in put output pair have a desynchronized block larger than d. Since $T_1 \subseteq_o R(T_2)$, for this origin graph σ there must be a corresponding $\sigma' \in [\![T_2]\!]_o$ and $(\sigma', \sigma) \in [\![R]\!]$. Thus any choice of σ' would result in a desynchronized block larger than d. By lemma 5.9 this resynchroniser R can't be bounded, so we reach a contradiction and know that $T_1 \not\preceq T_2$. $\square$

5.2.1 Classification of unresynchronizable origin graphs

The definition of desynchronized block from Lemma 5.9, is only one direction. While the pattern is enough to show unboundedness of a resynchronizer the reverse is not implied. For our main result the implication of desynchronized blocks is enough, but one might be interested to classify resynchronization that can't be bounded. In this section we will refine the notion of desynchronized to obtain a classification for resynchronizers that are not bounded.

Definition 5.12. *(Desynchronized position) Let $\sigma = (u, v, orig), \sigma' = (u, v, orig')$ be two origin graphs with the same input and ouput word. Given an interval of input positions $x_i x_{i+1} \dots x_j$, we say position x_n, $i \leq n \leq j$ is desynchronized for interval $[i, j]$ if there exists an output position y for which $orig(y) = x_n$ and $orig'(y) \notin \{x_i, x_{i+1}, \dots, x_j\}$.*

Informally that says an input position in the interval which has an output that is redirected outside the interval. A pair of origin graphs (σ, σ') is said to have k-bounded desynchronization $k \in \mathbb{N}$ if for every interval $[i, j], i, j \in dom(in(\sigma))$ there are at most k distinct input positions $x_{c_1}, \ldots, x_{c_k}$ with $i \leq c_1, \ldots, c_n \leq j$ which are desynchronized.

Definition 5.13. *(Bounded desynchronization) Let $[\![R]\!]$ be a relation on origin graphs, then $[\![R]\!]$ is said to have bounded desynchronization if there exists some $k \in \mathbb{N}$ such that for every pair of graphs $(\sigma, \sigma') \in [\![R]\!]$ these graphs have k-bounded desynchronization.*

Note that this is not a symmetric notion so the desynchronization from σ to σ' might be different than the reverse. A good example of a relation that has bounded desynchronization one way and not the other way is the relation relating the origin graphs of the transducers from Example 5.2.

Now we claim a resynchronizer is bounded if and only if the relation has bounded desynchronization. Before we introduce this lemma we first show that it is possible to construct a bounded resynchronizer R_{d_1} that computes all possible related origin graphs with desynchronization at most 1.

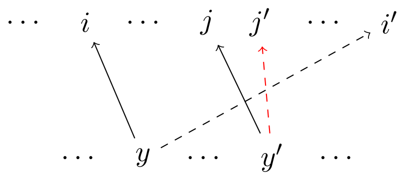
The construction of R_{d_1} is based on the following observation on a resynchronization with 1 desynchronization bound. Let R_1 be a resynchronization with desynchronization 1. This means that for every interval there is at most 1 input position which gets redirected outside this interval. Now for input word $u = x_1 \ldots x_n$ let $\sigma = (u, v, orig), \sigma' = (u, v, orig')$ be origin graphs such that $(\sigma, \sigma') \in R_1$ and let $y \in dom(v)$ be an output position with $orig(y) = i$ and $orig'(y) = i'$ we can claim the following:

If the origin moves right $i \leq i'$ then for all $j \in [i, i']$, if an output positions $y' \in dom(v)$ has origin $orig(y') = j$ then:

1. it does not move right: $orig'(y') \leq orig(y')$, if $orig'(y') > orig(y')$ then on the interval $[i, j]$ both i and j would be desynchronized which can't be the case.

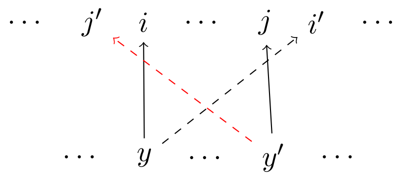
2. does not move left of i: $i \leq orig'(y')$, if $i > orig'(y')$ then on the interval $[i, j]$ both i and j would be desynchronized which can't be the case.

This behaviour is demonstrated in this picture in which the red arrow would make the resynchronization 2-bounded instead of 1-bounded, and thus is impossible.



$\cdots$ i $\cdots$ j j' $\cdots$ i'
$\cdots$ y $\cdots$ y' $\cdots$

First case



$\cdots$ j' i $\cdots$ j i' $\cdots$
$\cdots$ y $\cdots$ y' $\cdots$

Second case

Lemma 5.14. *Let σ, σ' have desynchronization-bound of 1, then for every input position $x \in dom(u)$ and output-position $y \in dom(v)$ for which $orig(y) = x$ then for every target $t \in [x, orig'(y)]$, $y' \in dom(v)$ with $orig(y') = t$ then for the new target $orig'(y') = t'$:*

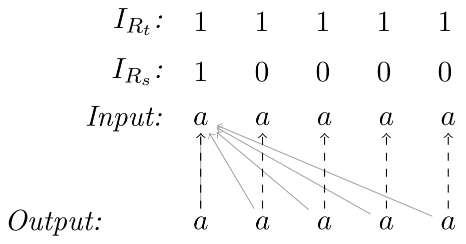
$$|t - x| \geq |t' - x| \wedge t \geq x \rightarrow t' \geq x \wedge t \leq x \rightarrow t' \leq x$$

If the origin moves left the same kind of analogy holds. By these requirements we construct a resynchronization which guesses sources and target for right and left moving distortions. For both right and left moving distortions according the requirements the resynchronization will only distort a guessed source to a guessed target if there is not another guessed source between it. In addition with MSO we make sure a left moving distortion does not overlap with a right moving distortion.

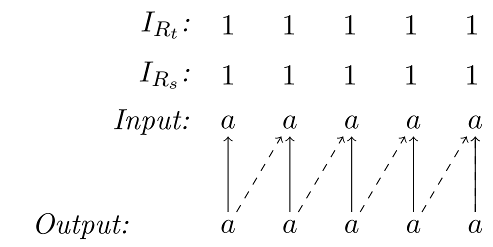
Definition 5.15. *The resynchronizer R_{d_1} that guesses right and left moving distortions with the input parameters $I_{R_s}, I_{L_s}, I_{R_t}, I_{L_t}$. The input parameters I_{R_s} (resp. I_{L_s})are used to indicate the source of the right (resp. left) moving distortions. The input parameters I_{R_t} (resp. I_{L_t}) are used to indicate the targets of these right (resp. left) moving distortions. Now construct our γ to ensure the above described requirements.*

$$\begin{aligned}\gamma(x, y) = & x = y \\ & \vee (x \in I_{R_s} \wedge y \in I_{R_t} \wedge x \leq y \wedge (\forall z \in (x, y).z \notin I_{R_s}) \\ & \vee (x \in I_{L_s} \wedge y \in I_{L_t} \wedge y \leq x \wedge (\forall z \in (x, y).z \notin I_{L_s})\end{aligned}$$

Example 5.16. *To demonstrate this behavior we show how R_{d_1} relates two graphs of T_{all} to T_{first}. Since it only has right moving distortions we only show the parameters I_{R_s}, I_{L_s} and assume $I_{L_s} = I_{L_t} = \emptyset$.*



Example 5.17. *Another example shows how this resynchronizer R_{d_1} relates two graphs in which every origin is moved right with exactly 1 position.*



By construction of our γ and the way it interacts with the input parameters we see that at most 3 distinct sources (the input position closest in I_{L_s}, I_{R_s} and the position itself) are valid for a every target position. So this resynchronizer is in fact bounded. Now we proof that it calculates every possible graph of desynchronization at most 1. It does in fact compute more since it does not take into account the second requirement for 1 bounded relations, however it relates at least the pairs that we require.

Lemma 5.18. *If $\sigma = (u, v, orig)$ and $\sigma' = (u, v, orig')$ are origin graphs, and the desynchronization of these graphs has 1-bound on desynchronization then it is related by R_{d_1}: $(\sigma, \sigma') \in [\![R_{d_1}]\!]$.*

Proof. Let $\sigma = (u, v, orig), \sigma' = (u, v, orig')$ be such arbitrary graphs with 1-bounded desynchronization. We derive the input parameters $I_{R_s}, I_{L_s}, I_{R_t}, I_{L_t}$ for R_{d_1} that witness that $(\sigma, \sigma') \in [\![R_{d_1}]\!]$. For every $y \in dom(v)$:

- origin stays equal $orig(y) = orig'(y)$, then we do nothing
- origin moves left $orig(y) < orig'(y)$ then $orig(y) \in I_{R_s}$ and $orig'(y) \in I_{R_t}$
- origin moves right $orig(y) > orig'(y)$ then $orig(y) \in I_{L_s}$ and $orig'(y) \in I_{L_t}$

Now for input positions $orig(y), orig'(y)$ we can derive $(u, \bar{I}, orig(y), orig'(y)) \models \gamma$ since for the three cases:

- $orig(y) = orig'(y)$ thus $(u, \bar{I}, orig(y), orig'(y)) \models \gamma$
- We know that $orig(y) \in I_{R_s}$ and $orig'(y) \in I_{L_s}$ and there can't be another input positions $z \in dom(u)$ with $orig(y) < z < orig'(y)$ for which $z \in I_{R_s}$ since this only happens if there exists an output position that has origin z and moves right which can't happen by by Lemma 5.14
- Analogous to the right moving distortions

□

Since the δ function can be chosen $\top$ by Lemma 5.4 we see that the resynchronizer is compositional. So we can freely construct a resynchronizer that $R_{d_1} \circ R_{d_1}$ or for any $n \in \mathbb{N}$ compute the resynchronization $R^n_{d_1}$.

Lemma 5.19. *For a set on pairs of origin graphs S there exists a bounded resynchronizer R such that $S \subseteq [\![R]\!]$ if and only if S has bounded desynchronization.*

Proof. ($\Rightarrow$) Assume S has no bounded desynchronization then for arbitrary $k \in \mathbb{N}$ there exists an interval $[i, j]$ with desynchronization k. This is a desynchronized block of size k, since k is chosen arbitrarily R is not bounded.

($\Leftarrow$) Assume there is a bound $n \in \mathbb{N}$ for desynchronization in S. Then $S \subseteq [\![R^n_{d1}]\!]$. □

5.3 Undecidability of resynchronizability

Theorem 5.20. *Given two non-deterministic one-way transducers T_1, T_2 the question whether $T_1 \preceq T_2$ is undecidable.*

In order to prove this we make a reduction inspired from the proof that Post Correspondence Problem (PCP) is undecidable as described in for example [Sip06]. To show indecidablity of the resynchronizability relation we use the problem that asks if a Turing machine M uses a bounded amount of tape. The reduction shares a lot of ingredients from the reduction from HALT to PCP [Pos46]. Instead of the halting problem (HALT) we use a modified version of this problem, but the main parts are the same.

5.3.1 The infinite tape problem

We will reduce from the problem called INFTAPE. Which is version of the halting problem that detects whether the Turing machine uses an infinite amount of tape instead of whether it stops.

Definition 5.21. *The decision problem* INFTAPE *decides if a Turing machine uses infinite tapecells on the empty input.*

$$\text{INFTAPE} = \{n \in \mathbb{N} \mid M_n \textit{ uses unbounded part of tape on the empty input}\}$$

For this problem it might not be too surprising that it is undecidable, here we prove that INFTAPE is undecidable by a reduction from the halting problem on the empty input string.

Lemma 5.22. *For a deterministic Turing Machine M it is undecidable whether $M \in$ INFTAPE.*

Proof. Proof this by reducing the halting problem on an empty tape $\varepsilon-$HALT,which we know is undecidable, to INFTAPE. Consider deterministic Turing machine M, we construct a new Turing machine M' which simulates M by writing the full configurations of M (state, contents of tape, and the position of the tape head) on the tape. This new machine M' will stop if and only if the computation of M stops. If M does not stop it will keep writing configurations and thus means M' will use infinite amount of tape cells regardless of the tape usage of M. So $M \in \varepsilon-$HALT $\iff M' \notin$ INFTAPE. Since $\varepsilon-$HALT is undecidable so is INFTAPE. □

5.3.2 The PCP domino game

Before introducing the reduction let us first introduce the Post correspondence problem, which is usually done with dominos. Say you have a set of dominos D where each domino has a top text and bottom text, and $\mathcal{I}$ is the finite set of indices of the tiles.

$$D = \{(u_1, v_1), \dots, (u_n, u_n) \mid u_i, v_i \in \Sigma^*, i \in \mathcal{I}\}$$

Now PCP asks the following question. Is there some sequence of tiles with indices $i_1 i_2 \dots i_n$ such that the dominos top and bottom part produce the same word?

$$u_{i_1} u_{i_2} \dots u_{i_n} = v_{i_1} v_{i_2} \dots v_{i_n}$$

Example 5.23. *Let us have a set of dominos D which are*

Index :	1	2	3
Top :	*a*	*ab*	*bba*
Bottom :	*baa*	*aa*	*bb*

Now we see this instance is a valid instance of PCP since we can construct a sequence of indices (3231*) such that the top and bottom part of the corresponding tiles are the same since $bba \cdot ab \cdot bba \cdot a = bb \cdot a \cdot bb \cdot baa$:*

Index :	3	2	3	1
Top	*bba*	*ab*	*bba*	*a*
Bottom :	*bb*	*aa*	*bb*	*baa*

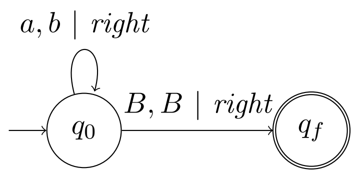
5.3.3 Turing machine to tiles

We can use these tiles to simulate a Turing machine on the dominos. In order to keep track of the computation history of the Turing machine M we model the full configuration of the Turing machine in strings. To capture a full configuration of a Turing machine we need the current contents of the tape, position of the tape head and state of the finite control automata. For a given Turing machine $M = (Q, \Sigma, q_0, q_f, \delta)$ The configuration can now be represented by $w \in \Sigma^* Q \Sigma^*$. On an alphabet $\Sigma = \{a, b\}$ a configuration might look like abq_6ba which represents a tape with $abba$ where the tape head is on the second b and the control automata is in $q_6 \in Q$. A full computation history of Turing machine is now given as all consecutive configurations divided by a seperation token $\# \notin \Sigma$. So the maybe infinite sequence of configurations models the computation history of M. Let $\Delta = \Sigma \cup Q \cup \{\#\}$ be alphabet of this history, then we call $\mathrm{Hist}_M \in \Delta^* \cup \Delta^\omega$ the computation history of M.

The computation history of a Turing machine $M = (Q, \Sigma, q_0, q_f, \delta)$ can be build in an instance of the domino game with these full configurations. Based on M we can construct set of tiles D_M with indices $\mathcal{I} = \{1, \dots, |D_M|\}$, where foreach $(u, v) \in D_M$, u is the top part of the domino and v the bottom part. We make the construction such that the bottom and top parts combined both make the configuration history but the bottom part is one configuration delayed. Construct this set of tiles D_M to contain each of the following tile:

- A copy tile (a, a) for every $a \in \Sigma \cup \{\#\}$
- A right transition tile (pa, bq) for every right moving transition $\delta(p, a) = (q, b, \text{right})$
- A left transition tile (cpa, qcb) for every $c \in \Sigma \cup \{B\}$ and $\delta(p, a) = (q, c, \text{left})$
- A right expansion tile $(q\#, qB\#)$ for every state $q \in Q$
- a left expansion tile $(\#qa, \#Bqa)$ for every $q \in Q, a \in \Sigma$.

Example 5.24. *Let $M = (Q, \{a, b\}, q_0, q_f, \delta)$ be a Turing machine that will read a's write b's and stops when the whole word is read*



Then the computation of M can be expressed with the following dominos:

Index :	1	2	3	4	5	6
Top :	a	b	$\#$	$q_0\#$	q_0a	q_0B
Bottom :	a	b	$\#$	$q_0B\#$	bq_0	Bq_f

These are all the tiles we need in our construction, now if we want to simulate M on the input a^2 we add the tile which only has the initial state on the bottom tile:

Index :	7
Top :	
Bottom :	$q_0 aa\#$

Now there is only one sequence of dominos that starts with tile 7 in which the top part is a prefix of the bottom part:

Index :	7	5	1	3	2	5	3	2	2	4	2	2	6	3
Top :		$q_0 a$	a	$\#$	b	$q_0 a$	$\#$	b	b	$q_0\#$	b	b	$q_0 B$	$\#$
Bottom :	$q_0 aa\#$	bq_0	a	$\#$	b	bq_0	$\#$	b	b	$q_0 B\#$	b	b	Bq_f	$\#$

Now the computation the content on the dominos expresses exactly the computation history of M.

We skip some technical details from the original reduction since we don't need it for our proof. While in PCP the top part needs to be equal to the bottom part, in our construction this can't happen because the state q_f only occurs on the bottom part of a tile. This can be fixed by adding tiles that remove all input symbols when q_f is reached, but in our case this will not be a problem so we leave these tiles out.

Another technical detail is the first tile which we silently added in Example 5.24. This tile does not need to be part of the set D_M since we will force a transducer to start with this first tile.

For these reasons we don't have the full property of PCP that the top part is equal to the bottom part, but in our reduction we don't care and we only care that by this construction we have that if the top part is a prefix of the bottom part, then the bottom part is a prefix of the computation history of M.

Lemma 5.25. *Given a Turing machine M and the set of tiles D_M with indices as described above. Given a sequence of indices $i_1 \dots i_k \in$ let the top parts of the corresponding tiles be given by $\bar{u} = u_{i_1} \dots u_{i_k}$, the top parts starting with the initial configuration given by $\bar{v} = q_0\# v_{i_1} \dots v_{i_k}$, and the computation history of M on the empty input given by $Hist_M$. Then*

$$\bar{u} \sqsubseteq \bar{v} \Rightarrow \bar{v} \sqsubseteq Hist_M$$

Proof. This can be proved using induction, but this is also true by construction of which more can be read in [Pos46, Sip06]. □

5.3.4 Tiles back to transducers

We now build two $1NT$'s that will output the top and bottom parts of the dominos D_M. To do so let $\Delta = Q \cup \{\#, B\} \cup \Sigma$ be the alphabet to have all characters needed to build the configuration history of the Turing machine M. Now consider the set of domino's D_M to model the computation history of M with indices $\mathcal{I} = \{1, \dots, |D_M|\}$. We now build two non-deterministic one-way transducers T_{up}, T_{down} that will have input alphabet $\mathcal{I}$ the indices, and output alphabet Δ which contains all content on the domino's. Let $i_{s_1} \dots i_{s_n} \in \mathcal{I}^*$ be the sequence of tiles that describe Hist_M then the output of the two transducers roughly looks like this. The only difference might be that there are intermediate transitions to expand the tape with B's.

Example 5.26. *The behaviour of T_{up}, T_{down} outputting a prefix of $Hist_M$.*

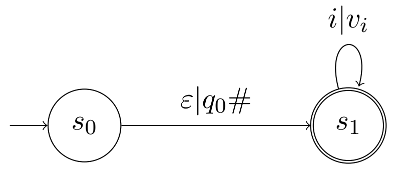
Input	i_{s_1}		i_{s_2}		i_{s_3}	$\dots$		i_{s_n}		$i_{s_{n+1}}$
Output T_{up}	q_0	$\#$	s_1	$\#$	s_2	$\dots$	$\#$	s_{n-1}	$\#$	$s_n\#s_{n+1}$
Output T_{down}	$q_0\#s_1$	$\#$	s_2	$\#$	s_3	$\dots$	$\#$	s_n	$\#$	s_{n+1}

On input $i_1 \dots i_k$ the transducer T_{down} will start by outputting $q_0\#$ and continue to output the top part of the tiles $v_{i_1} \dots v_{i_k}$ thus describing a function $i_1 \dots i_k \mapsto q_0\#v_{i_1} \dots v_{i_k}$.

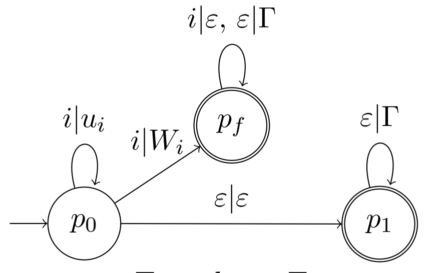
On the same input transducer T_{up} will output the bottom part of the tiles $u_{i_1} \dots u_{i_k}$. This transducer however is also allowed to non-deterministically output anything that is not a prefix of u_i and from that point output anything in Δ^*. The transducer T_{up} is also allowed to output anything at the end of the word, this is in order to catch up the one configuration it is delayed by construction.

In the figure both transducers T_{up}, T_{down} with the set W_i the set of valid non-prefixes of u_i:

$$W_i = \{u \in \Delta^* \mid |u| \leq |u_i| \text{ and } |u| \not\sqsubseteq |u_i|\}$$



Transducer T_{down}



$i|\varepsilon,\ \varepsilon|\Gamma$
p_f
$i|u_i$
$i|W_i$
$\varepsilon|\Gamma$
$\varepsilon|\varepsilon$
p_0
p_1

Transducer T_{up}

By construction on input word $i_1 \dots i_k$ the only way T_{down} outputs $q_0\#u_{i_1} \dots u_{i_k}$ is that the origins of u_{i_j} is exactly the input position i_j.

The same thing holds for transducer T_{up} outputting $v_{i_1} \dots v_{i_k}$. Additionally by construction of the tiles if on the same input word $i_1 \dots i_k$ the output $v = v_{i_1} \dots v_{i_k}$ generated by T_{up} is a prefix of the output $w = q_0\# u_{i_1} \dots u_{i_k}$ if v, w describe a prefix of the computation history of M.

Lemma 5.27. *$M \in$ INFTAPE $\implies T_{down} \not\preceq T_{up}$*

Proof. We assume $M \in$ INFTAPE so we know that uses an infinite amount of tape. For any $k \in \mathbb{N}$ let $\bar{v}$ be the prefix of the computation history $\bar{v} \subseteq \text{Hist}_M$ witnessing a configuration that uses $k + 2$ tapecells. Now by our construction there exists a sequence of indices $\bar{i} = i_0 i_1 \dots i_n$ that represents this computation history in the top part of the tiles $\bar{v} = v_{i_1} \dots v_{i_n}$. $q_0\# v_{i_0} \dots v_{i_N} = q_0\# s_1 \# \dots \# s_{n+1}$ is produced by T_{down} as origin graph $\sigma \in [\![T_{down}]\!]_o$.

The origin graph σ' of T_{up} that has the input $in(\sigma') = i_0 \dots i_N$ and outputs exactly the computation history $out(\sigma') = q_0\# s_0 \# \dots \# s_{n+1}$ is outputted only when the automata ends in p_1. Since every configuration of the Turing machine in particular s_n is outputted with delay in T_{up}, so the input positions producing s_n have no output positions in common, so the input positions that produce s_n are a desynchronized block. Since s_n uses at least $k + 2$ tapecells this has to take at least k input positions to produce with our tile construction. This block is a desynchronized block of size at least k and σ, σ' are the only origin graphs with this exact input and output so by Lemma 5.11 we can conclude $T_{down} \not\preceq T_{up}$. □

Lemma 5.28. *$M \notin$ INFTAPE $\implies T_{down} \preceq T_{up}$*

Proof. To show this we construct a bounded resynchronizer R such that $T_{down} \subseteq_o R(T_{up})$.

Since $M \notin$ INFTAPE there exists a bound $k \in \mathbb{N}$ on the tape M uses. We build a resynchronizer without input and output parameters:$R = (\top, \top, \gamma, \top)$ where γ is given by $\gamma(x, y) = \bigvee_{0 \le i \le k+2} x = y + i$. This is a bounded resynchronization since for any word w and target y there are only $k + 3$ distinct sources x_i such that $(w, x_i, y) \models \gamma$.

Now to show that $T_{down} \subseteq_o R(T_{up})$ we prove that for every $\sigma \in [\![T_{down}]\!]_o$ there exists a $\sigma' \in [\![T_{up}]\!]_o$ for which $(\sigma', \sigma) \in [\![R]\!]$. Let $\sigma \in [\![T_{down}]\!]_o$ be an arbitrary origin graph with input $\bar{i} = i_1 i_2 \dots i_n$, $\sigma = (\bar{i}, v, orig)$ then the construction of our transducers the output $v = q_0\# v_{i_1} \dots v_{i_n}$ the exact bottom parts of the sequence of tiles $\bar{i}$.

Now either the bottom part of the tiles $\bar{u} = u_{i_1} \dots u_{i_n}$ form a prefix of v or they don't:

- If $\bar{u} \sqsubseteq v$ then by Lemma 5.25 we have $v \sqsubseteq \text{Hist}_M$. The transducer T_{up} can output this word $v_{i_1} \dots v_{i_n}$ in state p_0 with exactly one configuration delay as seen in Example 5.24 and the table in Example

5.26. Let σ' be this origin graph with in addition the last configuration outputted at the end in state p_1. Since there is a bound k on tape usages of M and the encoding of this configurations takes at most $k+2$ positions we can conclude that $(\sigma', \sigma) \in [\![R]\!]$

- If $\bar{u} \not\sqsubseteq v$ then in order to output v. The transducer T_{up} has to go to the state p_f, at the point the tiles are not equal anymore. From this point T_{up} can mimic T_{down} as close as possible and catch up at the end. This run produces the origin graph $\sigma' \in T_{up}$ and since at any point before it goes to p_f the shift is at most $k+2$ and does not change after we can conclude $(\sigma', \sigma) \in [\![R]\!]$

□

Combining all lemma's gives us a proof of Theorem 5.20.

Proof. By Lemma 5.27 and Lemma 5.28 we know that $T_{down} \preceq T_{up} \iff M \notin$ INFTAPE. By Lemma 5.21 INFTAPE is an undecidable problem. The construction in these Lemmas is effective, thus deciding whether $T_{down} \preceq T_{up}$ is undecidable for non-deterministic one way transducer. □

This proof extends to any class of transducers that include the class of $1NT$s in particular it proves the open problem from [BKM+19] for $2NT$ case. Since the reduction does not use parameters the following collorary is the most general one.

Corollary 5.29. *Given T_1, T_2 two $1NT$'s, it is undecidable whether there exists a bounded regular resynchronizer R without parameters such that $T_1 \subseteq_o R(T_2)$.*

Chapter 6

Conclusions

In this thesis we recalled how regular resynchronizers can be used in order to expand the notion of equivalence for the origin semantics of a transducer with respect to the classical semantics. We used bounded regular resynchronizers to form a resynchronizability relation relation on two transducers $T_1 \preceq T_2$ that states whether there exists a bounded regular resynchronizer R such that T_1 is contained in T_2 up to R. We showed that this relation forms a preorder intermediate of the undecidable classical containment and the decidable origin containment.

In order to prove transducers are not resynchronizable we studied what behavior is possible to resynchronize using bounded regular resynchronizers. We introduced a notion of desynchronized and proved that a regular resynchronizer is bounded if and only if it has bounded desynchronization. This helped us to prove the non-existence of a bounded regular resynchronizer that witnesses the $\preceq$ relation.

With this tool witnessing unresynchronizability we could make a reduction from INFTAPE to the resynchronizability relation $\preceq$ for $1NT$s. A direct corollary is that this solves the open problem stated in [BKM$^+$19] asking whether this problem is decidable for the general case of $2NT$s.

A very natural follow-up question would be to extend this construction to the decidability of the equivalence closure $\sim_D$. Preliminary results indicate the same proof works to show this relation undecidable. In order for the same structure to work, one has to modify T_{up}, T_{down} to be equal in classic semantics. Another interesting question for further research would be if the approach of extending containment with the distortion relation would yield decidable results on the singleton alphabet. For the distortion relation it is also interesting to see if the δ-formula is compositional. That would solve the problem stated in [BMPP18] whether, given resynchronizations R_1, R_2, it is possible to (effectively) find a resynchronization R_3 such that $[\![R_3]\!] = [\![R_1]\!] \circ [\![R_2]\!]$.

Another interesting question is how this undecidability proof generalizes in different classes of transductions. This follows directly for classes in which the class of one-way non-deterministic transductions are contained. But there are some classes like NSST and MSOT which are incomparable with the class of non-deterministic one-way transducers and for which it would be interesting to see if the undecidability proof can be reproduced in that machine model, or that in fact this resynchronization relation is decidable.

Bibliography

[AČ10] Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–12, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BDGP17] Mikolaj Bojanczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. Which classes of origin graphs are generated by transducers. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[BKM$^+$19] Sougata Bose, Shankara Narayanan Krishna, Anca Muscholl, Vincent Penelle, and Gabriele Puppis. On Synthesis of Resynchronizers for Transducers. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 69:1–69:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BMPP18] Sougata Bose, Anca Muscholl, Vincent Penelle, and Gabriele Puppis. Origin-equivalence of two-way word transducers is in pspace. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[Boj14] Mikołaj Bojańczyk. Transducers with origin information. In *International Colloquium on Automata, Languages, and Programming*, pages 26–37. Springer, 2014.

[Büc60] J Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

[Col13] Thomas Colcombet. The factorisation forest theorem. *To appear in the handbook "Automata: from Mathematics to Applications"*, 2013.

[DFL18] Luc Dartois, Emmanuel Filiot, and Nathan Lhote. Logics for word transductions with synthesis. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 295–304. ACM, 2018.

[EH01] Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254, 2001.

[Elg61] Calvin C Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, 1961.

[FJLW16] Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[Iba78] Oscar H Ibarra. The unsolvability of the equivalence problem for ε-free ngsm's with unary input (output) alphabet and applications. *SIAM Journal on Computing*, 7(4):524–532, 1978.

[Pos46] Emil L. Post. A variant of a recursively unsolvable problem. *Bull. Amer. Math. Soc.*, 52(4):264–268, 04 1946.

[Sim90] Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72(1):65–94, 1990.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation*, volume 2, page 199–205. Thomson Course Technology Boston, 2006.

[Tra61] Boris A Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 140(326-329):122–123, 1961.

Appendix

Lemma 5.10 Given a resynchroniser R, if for any number $d \in \mathbb{N}$ there is a tuple of two origin graphs $(\sigma, \sigma') \in [\![R]\!]$ that have a desynchronised block of size greater than d. Than R is not bounded.

Proof.

1. Let $R = (\alpha, \beta, \gamma, \delta)$ with n input parameters and m output parameters. We can assume $\alpha = \beta = \delta = \top$, since the definition of bounded is only based on γ if $R' = (\top, \top, \gamma, \top)$ is not bounded than also R is not bounded.

2. For an arbitrary $k \in \mathbb{N}$ we show that there is a word $w \in \Sigma^*$, $\bar{I} \in (\mathbb{B}^n)^N$, an output-type τ, k distinct sources $x_1, \dots x_k \in [1, N]$, and one target position $y \in [1, |w|]$ such that $\forall i \in [1, k],\ (w, \bar{I}, x_i, y) \models \gamma(\tau)$. This shows that R is unbounded.

3. For all output types τ the formula $\gamma(\tau)$ is a MSO sentence with variables $\bar{I}$ and a first order variable x, y modeling source and target. For every type τ this MSO-sentence recognizes a regular language, and therefore there is a monoid M_τ and a morphism $f_\tau : \Sigma \times \mathbb{B}^{n+2} \to M_\tau$ recognizing the language of $\gamma(\tau)$, where the booleans encode the n input parameters $\bar{I}$ and the source and target x and y.

   Pick the type τ_i with the largest corresponding monoid M_{τ_i}, and let $\pi : M^* \to M$ be the evaluation morphism in M. Using the Simon's factorization forest theorem[Sim90], we know that there is $H \in \mathbb{N}$ such that for any word $u = m_1 m_2 \dots m_H \in M^H$, there is an idempotent $e \in M$ and a factorization of u into $u = m e_1 e_2 \dots e_{k+2} m'$ with $m = m_1 \dots m_{i_1}, e_j = m_{i_j+1} m_{i_j+2} \dots m_{i_{j+1}}$ and $m' = m_{i_{k+2}+1} \dots m_H$, such that for all $j \in [1, k+2]$, $\pi(e_j) = e$. This property also holds for $M = M_\tau$ for any output-type τ, as we chose M_{τ_i} of maximal size, and the bound H increases with the size of the monoid

4. Pick the smallest graph $(\sigma, \sigma') \in [\![R]\!]$ that has a desynchronised block of size $N \geq |\Sigma| * 2^m(H + 1)$. By our assumption this pair of graphs exists.

5. Pick the output type τ which occurs the most in the positions of the desynchronised block that produce output $v = in(\sigma)$. The types τ

are element of $\Gamma \times \mathbb{B}^m$ for the fixed m output parameters, so there are $|\Sigma| * 2^m$ output types τ_i. The τ we pick that occurs the most in the desynchronised block occurs atleast $\frac{N}{|\Sigma|*2^m} = \frac{|\Sigma|*2^m(H+1)}{|\Sigma|*2^m} = H+1$ times in the desynchronised block.

6. In the desynchronised block of the word pick the last element that produces output x_{last} with type τ we know that $(w, \bar{I}, x_{last}, y) \models \gamma(\tau)$, where y is not a part of the desynchronised block. Let $\bar{w} \in \Sigma \times \mathbb{B}^{n+2}$ be the encoding of $(w, \bar{I}, x_{last}, y)$. This word w will be accepted by the monoid M_τ with morphism f^*_τ.

7. Split the word $\bar{w}$ in parts $\bar{w} = w_1 \dots w_l$ where all the blocks have exactly one letter that produce output with output type τ. There are at least $H+1$ blocks in the desynchronised block for since at least that much occurrences of output positions with type τ that produce output. Strictly before the position x_{last} there are at least H blocks $w_i, w_{i+1}, \dots, w_{i+H}$, within the desynchronised block.

8. Using the result from the Simon's factorization, there is an idempotent $e \in M_\tau$ a factor $u = u_1 u_2 \dots u_{k+2}$ in the desynchronised block of $\bar{w}$, and a strictly increasing sequence $i_1, i_2 \dots i_{k+2}$ such that each u_j is of the form $w_{i_j+1} \dots w_{i_{j+1}}$, and $\hat{f}_\tau(u_j) = e$.

9. By construction there is an element x' with type τ in u_2, there is a corresponding element y', that is not part of the desynchronised block such that $(w, \bar{I}, x', y') \models \gamma(\tau)$. We pick the word $\bar{w'} \in \Sigma \times \mathbb{B}^{n+2}$ that is accepted by monoid M_τ. The subword $u = u_1 u_2 \dots u_{k+2}$ is now changed to $u_1 u'_2 u_3 \dots u_{k+2}$ where $u'_2 = (\pi_1(u_2), 1, 0, \pi_I(u_2))$. In this equation $\pi_1(u_2)$ is the left projection which gives the part of u_2 which is in Σ and π_I is the right projection representing the booleans for the input parameters $\mathbb{B}^n$. For all $i \neq 2$ the blocks u_i are still represented by the same element in the monoid $\hat{f}_\tau(u_i) = e$ the idempotent element of M.

10. Change $u_1 u'_2 u_3 \dots u_{k+2}$ to $u' = u_1 u'_2 u_2^k$. this word is still accepted since it does not change the element of the monoid M_τ under the morphism. It can change the length of the word and the input parameters $\bar{I}$, but since it is projected on the same element in the monoid M, it is accepted. This new word $w'' \in \Sigma \times \mathbb{B}^{n+2}$, input parameters $\bar{I}'$ source x' and the target y'.

11. The word $\pi_1(w'')$ with input parameters $\bar{I}'$ and target y has k distinct sources x for which $(\pi_1(w''), \bar{I}', x, y) \models \gamma(\tau)$. This can be done by swapping the block with the source position x in the sequence of idempotents without changing the element of the monoid under the morphism. Change the subword u' of w'' by $u_i = u_1 u_2^i u'_2 u_2^{k-i}$ for

$0 \leq i \leq k$. All these different subwords $u_i \in \Sigma \times \mathbb{B}^{n+2}$ have the same left projection in Σ, the same input parameters, and a different source positions x_i so for all these positions $(\pi_1(w''), \bar{I}', x_i, y) \models \gamma(\tau)$.

$\square$