Journal of Automata, Languages and Combinatorics **9** (2004) 1, 121–146
© Otto-von-Guericke-Universität Magdeburg

# BIMACHINES AND STRUCTURALLY-REVERSED AUTOMATA

Nicolae Santean
*Department of Computer Science, The University of Western Ontario*
*1151 Richmond Street, Suite 2, London, Ontario N6A 5B8, Canada*
*e-mail:* `nic@csd.uwo.ca`

ABSTRACT

Although bimachines are not widely used in practice, they represent a central concept in the study of rational functions. Indeed, they are finite state machines specifically designed to implement rational word functions. Their modelling power is equal to that of single-valued finite transducers. From the theoretical point of view, bimachines reflect the decomposition of a rational function into a left and a right sequential function. In this paper we define three new types of bimachines, classified according to the scanning direction of their reading heads. Then we prove that these types of bimachines are equivalent to the classical one and for doing so, we define and use a new concept, of structurally-reversed automaton. Consequently, we prove that the scanning directions of bimachines are irrelevant from the point of view of their modelling power. This leads to a method of simulating a bimachine by a left sequential transducer (or generalized sequential machines - *GSM* for short). Indeed, a preprocessing of the input word allows sequential transducers to realize the full range of rational functions. Remarkably enough, we basically show that the so versatile functional transducers - nondeterministic and with $\lambda$-input transitions - can successfully be replaced by a simple deterministic setup: a "trimmer" coupled with a *GSM*. Intuitively, this fact proves that sequential functions are not much weaker than rational functions.

*Keywords:* structurally-reversed automata, rational functions, bimachines, *GSM*

## 1. Introduction

The interest in providing finite and effective descriptions of sets in certain algebraic structures dates as early as 40's. Formal machines, initially designed as nets of formalized neurons (McCulloch-Pitts nets, comprising of synchronized elements, each capable of some boolean function) were introduced by McCulloch and Pitts in 1943 ([14]) in order to carry out the control operations of a Turing Machine ([25]). The idea was further refined by Kleene in 1956 ([11]), who interrelated regular sets (or regular events in nerve nets), regular expressions and finite automata.

In parallel, a special interest in formal and natural language processing was developing. Indeed, in addition to classification ([3, 4] - Chomsky, 1956–59), recognition and generation of languages, a growing interest for the study of devices with output emerged. In [16](Mealy, 1955), [18](Moore, 1956) and [22](Raney, 1958) we find the design of finite sequential machines which both decide whether some input word belongs to a given language and record a "trace" of their computation during the decision process.

These initial attempts to implement language transductions were followed in 60's and 70' by a systematic study of rational and regular families of sets, in particular of rational word relations and functions. Bimachines ([23] - Schutzenberger, 1961), generalized/complete sequential machines ([7] - Ginsburg, 1966) and subsequential transducers ([24] - Schutzenberger, 1977) were added to the portfolio of finite sequential machines with output. Soon, sequential and subsequential transducers gained momentum, being extensively studied in [5](Eilenberg, 1974) and [2](Choffrut, 1978).

The past 15 years have witnessed a revival of the topic, due to an increased practical interest. Indeed, applications of rational relations and functions in Code Theory and Communications ([9] - Head and Weber, 1993; [12] - Konstatinidis, 2002), in Natural Language Processing ([17] - Mohri, 1997), as well as in DNA Computing ([20] - Paun et al., 1998; [13] - Manca et al., 1999) have undoubtedly proven the high degree of applicability of this endeavour.

The present paper tackles aspects of bimachine design. These machines, which realize rational word functions, have two reading heads which scan the input in opposite directions and in multiple passes. One may ask why these machines need more than one reading head and why the scanning direction of their two reading heads is the way Schutzenberger designed it to be. We addressed these questions and found that a simple preprocessing of the input word can lead to the disuse of one reading head and that the scanning directions do not actually matter.

In Section 2 we give a few basic notions related to automata, equivalences and rational sets and we review the relationship between recognizable and rational sets in monoids. In Section 3 we describe bimachines, we present two characterizations of rational functions and we define three new types of bimachines, classified based on the scanning direction of their reading heads. In order to prove their equivalence we first introduce the concept of structurally-reversed automaton and study its properties in Section 4. In Section 5 we prove that indeed, all types of bimachines are equivalent, by essentially using the properties of structurally-reversed automata. Finally, in Section 6 we use the equivalence of a classical bimachine with a left sequential bimachine in order to prove that $GSM$ (generalized sequential machines, also referred to as left sequential transducers) can eventually replace bimachines. This application conveys the fact that rational functions and sequential functions are not too far apart and that we can successfully implement rational functions by means of sequential transducers, hence obsoleting bimachines and single-valued transducers. The advantage of this approach becomes apparent when we notice that both bimachines and single-valued transducers are quite difficult to design and manipulate (for example, to minimize).

## 2. Basic Notions

Let $X$ be an alphabet, i.e., a nonempty, finite set of symbols. By $X^*$ we denote the set of all finite words (strings of symbols) over $X$ and by $\lambda$ we denote the empty word (a word having zero symbols). The operation of concatenation (juxtaposition) of two words $u$ and $v$ is denoted by $u \cdot v$, or simply $uv$. Notation wise, if $u$ is a word, then $u^R$ is the word obtained by reversing the order of symbols in $u$ ($u^R$ is the reverse of $u$). A language is a subset of $X^*$.

A `deterministic finite automaton` over $X$, $DFA$ for short, is a tuple $A = (Q, X, \delta, q_0, F)$ where

- $Q$ is a finite set of states and $X$ is an input alphabet;
- $\delta : Q \times X \to Q$ is a next state function;
- $q_0$ is an initial state and $F \subseteq Q$ is a set of final states.

The next state (or transition) function is extended to work on words as following: $\delta(q, \lambda) = q, \forall q \in Q$ and $\delta(q, aw) = \delta(\delta(q, a), w), \forall a \in X, w \in X^* and q \in Q$. The language recognized by $A$ is $\mathcal{L}(A) = \{w \in X^* \mid \delta(q_0, w) \in F\}$ (a `regular language` over $X$ is any language recognized by some $DFA$ over $X$). A $DFA$ can be viewed as a machine with a reading head, an internal current state and a finite table governing the change of its state with respect to the symbols read from an input tape. By a computation in $A$ we understand an expression $qw_1w_2 \vdash q'w_2$, which denotes that $A$ has advanced from state $q$ to state $q'$ while reading(consuming) the prefix $w_1$ of the input $w_1w_2$. If $\delta$ is a total function, we say that $A$ is complete, otherwise $A$ is incomplete. A complete $DFA$ rejects a word if the reading of that word leads to a non-final state. An incomplete $DFA$ rejects a word also when it blocks - i.e. when the next state function is not defined on the initial state and that word. A state is accessible in $A$ if there exists a computation from $q_0$ to that state. A state is coaccessible, if there exists a computation form that state to some final state. A state is useful if it is both accessible and coaccessible.

Let $A, B$ be two arbitrary sets. The Cartesian product of $A$ and $B$ is denoted by $A \times B := \{(a, b) \mid a \in A, b \in B\}$. A binary relation over $A$ and $B$ is a subset $R$ of $A \times B$. The inverse relation of $R$ is $R^{-1} = \{(b, a) \mid (a, b) \in R\}$. The identity of $A$ is the relation $id_A = \{(x, x) \mid x \in A\}$. The composition of two relations $R_1 \subseteq A \times B$ and $R_2 \subseteq B \times C$ is the relation $R_2 \circ R_1 = \{(a, c) \mid \exists b \in B : (a, b) \in R_1 \text{ and } (b, c) \in R_2\}$. We say that a relation $R_1$ is coarser than another relation $R_2$ if $R_2 \subseteq R_1$. $R \in A \times A$ is an equivalence over $A$ if it is reflexive ($id_A \subseteq R$), symmetric ($R^{-1} = R$) and transitive ($R \circ R \subseteq R$). A binary operation over $A$ is a function $\phi : A \times A \to A$. We use the infix notation to denote binary operations: $a\phi b := \phi(a, b)$. Let $\phi$ be a binary operation and $R$ be an equivalence, over $A$. Then $R$ is a right invariant equivalence with respect to $\phi$ if $(a, b) \in R \Rightarrow (a\phi c, b\phi c) \in R, \forall c \in A$, and is a left invariant equivalence if $(a, b) \in R \Rightarrow (c\phi a, c\phi b) \in R, \forall c \in A$. Given an equivalence $R$ over $A$ and an element $a \in A$, the equivalence class of $a$ with respect to $R$ is the set $\hat{a} := \{b \in A \mid (a, b) \in R\}$. All possible equivalence classes of $R$ represent a partition of $A$, i.e. they do not overlap and they cover $A$.

A `monoid` is a tuple $(M, \circ, 1_M)$, where $M$ is a nonempty carrier set, $\circ$ an associative binary operation over $M$ ($\forall a, b, c \in M : a \circ (b \circ c) = (a \circ b) \circ c$) and $1_M$ a zero-ary operation denoting the unity of $M$ ($\forall a \in M : 1_M \circ a = a \circ 1_M = a$). A monoid morphism is a total function from one monoid to another, which maps unity to unity and is compatible with monoid's operations. If $A, B$ are subsets of $M$, then $A \circ B =$

$\{a \circ b \mid a \in A, b \in B\}$. Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. Then $A^0 := \{1_M\}$ and $A^n := A \circ ... \circ A$ ($n$ times), for all $n \in \mathbb{N}^+$. In addition, $A^+ := \bigcup_{n \in \mathbb{N}^+} A^n$ and $A^* := A^0 \cup A^+$.

**Definition 1** *The family of rational subsets of the monoid $M$, denoted by $\mathcal{RAT}(M)$, is the least family of subsets of $M$ satisfying the following conditions:*

*(i)* $\emptyset \in \mathcal{RAT}(M)$;

*(ii)* $\forall a \in M : \{a\} \in \mathcal{RAT}(M)$ ;

*(iii)* $\forall A, B \in \mathcal{RAT}(M) : A \cup B \in \mathcal{RAT}(M)$ *and* $A \circ B \in \mathcal{RAT}(M)$;

*(iv)* $\forall A \in \mathcal{RAT}(M) : A^+ \in \mathcal{RAT}(M)$.

Consequently, if $A \in \mathcal{RAT}(M)$ then $A^* \in \mathcal{RAT}(M)$ (see [1, p. 55] for a discussion on rational sets).

Given $X$ and $Y$ two alphabets, we consider the monoid $(X^* \times Y^*, \circ, 1_{X^* \times Y^*})$, where:

- $(u_1, v_1) \circ (u_2, v_2) := (u_1 u_2, v_1 v_2)$;
- $1_{X^* \times Y^*} := (\lambda, \lambda)$.

Notice that the monoid $X^* \times Y^*$ defined above is finitely generated, in the sense that there exists a finite subset $B$, called a set of generators, such that $B^* = X^* \times Y^*$ (indeed, take $B = (X \times \{\lambda\}) \cup (\{\lambda\} \times Y)$). Notice also that $X^* \times Y^*$ is not necessarily a free monoid, in the sense that does not exist a set of generators which generate each element of the monoid in a unique way (for example, $B$ - above - can generate an element in more than one way : $(x, y) = (x, \lambda) \circ (\lambda, y) = (\lambda, y) \circ (x, \lambda)$; notice also that $X^* \times Y^*$ is not a commutative monoid). The fact that $X^* \times Y^*$ is not a finitely generated free monoid is of major importance: it implies that finite automata can not always recognize sets in $\mathcal{RAT}(X^* \times Y^*)$. In order to state this fact clearly, let us review the definition of recognizable sets and emphasize the difference between rational and recognizable sets in arbitrary monoids.

By recognizable sets in a monoid $M$, denoted by $\mathcal{REC}(M)$, we understand the family of all inverse images of subsets of arbitrary finite monoids through monoid morphisms. For a formal definition and study of recognizable sets consult [1, p. 52] or [21, p. 689]. The following facts are worth recalling:

1. ([15] - McKnight, 1964). If $M$ is a `finitely generated monoid` then
$$\mathcal{RAT}(M) \supseteq \mathcal{REC}(M).$$
2. ([11] - Kleene, 1956). If $M$ is a `finitely generated free monoid` then
$$\mathcal{RAT}(M) = \mathcal{REC}(M),$$
case in which we refer to this family as the family of regular languages.
3. If $M$ is an arbitrary monoid, we can not relate $\mathcal{RAT}(M)$ and $\mathcal{REC}(M)$.

As mentioned above, the monoid $X^* \times Y^*$ is finitely generated but not free and Kleene's result does not hold here. Then we can clearly affirm that $\mathcal{RAT}(X^* \times Y^*) \supseteq \mathcal{REC}(X^* \times Y^*)$, the inclusion being strict in general. Hence finite automata over arbitrary monoids (as defined in [26, p. 8] or in [1, Ex. 1.2, p. 55]) can not always recognize sets in $\mathcal{RAT}(X^* \times Y^*)$ - they can solely recognize sets in $\mathcal{REC}(X^* \times Y^*)$. However, there exist finite machines - transducers - which exactly express the family of rational subsets of $X^* \times Y^*$, also called rational relations. Figure 1 gives an approximate hierarchy of rational relations together with the appropriate machines which represent each family. In this paper we focus on rational and sequential functions, realized by bimachines and sequential transducers, respectively. A detailed discussion on the families of this hierarchy can be found in [1, Chapters III and IV].

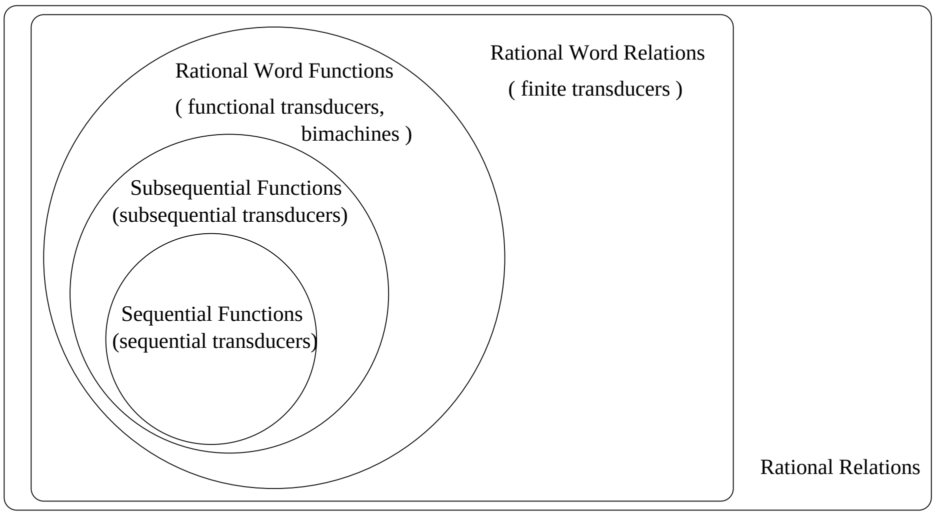


Figure 1: A hierarchy of rational relations.

## 3. Types of Bimachines

Rational word functions are partial functions from $X^*$ to $Y^*$ whose graphs are rational subsets of the monoid $X^* \times Y^*$. They are a particular case of rational relations, hence they are realized by functional transducers from $X^*$ to $Y^*$ - also called single-valued transducers. However, there exists a finite machine specially customized to express rational functions: the bimachine.

**Definition 2** *A* `bimachine` $B = (Q, P, X, Y, \delta_Q, \delta_P, q_0, p_0, \omega)$ *over* $X$ *and* $Y$ *is composed of*

*(i) two finite sets of states* $Q$ *and* $P$*;*

*(ii) a finite input alphabet* $X$ *and a finite output alphabet* $Y$*;*

(iii) *two partial next state functions* $\delta_Q : Q \times X \to Q$ *and* $\delta_P : X \times P \to P$*;*

(iv) *two initial states* $q_0 \in Q$ *and* $p_0 \in P$*;*

(v) *and a partial output function* $\omega : Q \times X \times P \to Y^*$*.*

The next state functions are extended to operate on words as following:

- $\forall q \in Q$ and $p \in P : \delta_Q(q, \lambda) = q$ and $\delta_P(\lambda, p) = p$;
- $\forall q \in Q, p \in P, a \in X$ and $w \in X^+$:

$$\delta_Q(q, wa) = \delta_Q(\delta_Q(q, w), a) \text{ and } \delta_P(aw, p) = \delta_P(a, \delta_P(w, p)).$$

Notice that function $\delta_P$ "reads" its word argument in reverse. We then consider a similar extension of the output function:

- $\forall q \in Q$ and $p \in P : \omega(q, \lambda, p) = \lambda$;
- $\forall q \in Q, p \in P, a \in X$ and $w \in X^+$:

$$\omega(q, wa, p) = \omega(q, w, \delta_P(a, p))\omega(\delta_Q(q, w), a, p).$$

The partial word function realized by $B$ is a function $f_B : X^* \to Y^*$, defined by

$$f_B(w) = \omega(q_o, w, p_0) \text{ if } \omega \text{ is defined in } (q_0, w, p_0) \text{ and is undefined otherwise.}$$

Notice that in essence, a bimachine is composed of two partial automata without final states (more precisely, all states act as final) and an output function. Indeed, $(Q, X, \delta_Q, q_0)$ will denote the `left automaton` of $B$ and $(P, X, \delta_P, p_0)$ its `right automaton`. The bimachine $B$ operates as illustrated in Figure 2.

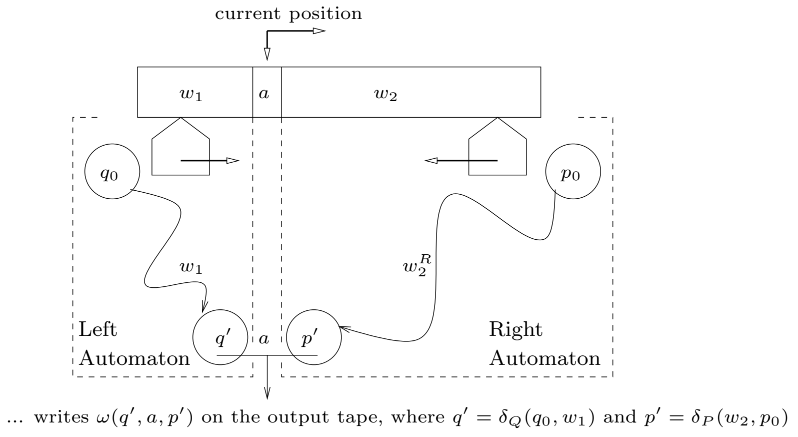

current position
$w_1$
$a$
$w_2$
$q_0$
$p_0$
$w_1$
$w_2^R$
Left Automaton
$q'$
$a$
$p'$
Right Automaton
... writes $\omega(q', a, p')$ on the output tape, where $q' = \delta_Q(q_0, w_1)$ and $p' = \delta_P(w_2, p_0)$

Figure 2: Computations in a bimachine.

The symbols on the input tape are considered from left to right, starting with the leftmost one. For each considered symbol the bimachine performs a computation step yielding some output written on an output tape. In Figure 2, the current computation step considers some symbol $a$ as the current symbol and a factorization of the input word as $w_1aw_2$. First, both left and right automata are reset to their initial states. Then the left automaton scans $w_1$ from left to right, reaching an internal state $q'$. In the same time, the right automaton scans the subword $w_2$ from right to left, reaching an internal state $p'$. At this point, the bimachine applies the output function $\omega$ to the arguments $q'$, $a$ and $p'$ and writes the result on the output tape. Next, the current position advances one symbol to the right and the process is repeated. The final output is the concatenation of the output for each step, as sequentially written on the output tape. This process is formally expressed by

$$\forall w = a_1...a_n \in X^+ \text{ (where } a_i \in X, \forall i \in \{1,...,n\}) : \ \omega(q,w,p) =$$

$$\omega(q,a_1,\delta_P(a_2...a_n,p))\omega(\delta_Q(q,a_1),a_2,\delta_P(a_3...a_n,p))...\omega(\delta_Q(q,a_1...a_{n-1}),a_n,p).$$

As we mentioned before, bimachines are of great theoretical importance since they are specifically designed to characterize the family of rational word functions, as the following result shows:

**Theorem 3** *[5, Volume A, §11.7, Theorem 7.1, p. 321] Let $X,Y$ be finite alphabets and $f : X^* \to Y^*$ be a partial word function with $f(\lambda) = \lambda$. Then $f$ is rational if and only if it is realized by some bimachine over $X$ and $Y$.*

Taking a closer look at the control operations of a bimachine, one may pay attention to the scanning direction of its two reading heads: the left automaton scans always from left to right and the right automaton scans always from right to left. This behaviour is in line with the decomposition theorem of Elgot and Mezei:

**Theorem 4** *[6, §7, Theorem 7.8, p. 61] A partial function $f : X^* \to Y^*$ with $f(\lambda) = \lambda$ is rational if and only if there exist an alphabet $Z$, a left sequential function $\alpha : X^* \to Z^*$ and a right sequential function $\beta : Z^* \to Y^*$, such that $f = \beta \circ \alpha$.*

We briefly mention that left sequential functions are realized by left sequential transducers - which scan the input from left to right - and that right sequential functions are realized by right sequential transducers - which scan the input from right to left (in Section 6 we give more details about sequential transducers). It is now straightforward the parallel between bimachines and the above decomposition theorem.

What will happen if we change the scanning directions in a bimachine? Since the scanning directions of the reading heads of this "classical" bimachine are from the extremities of the input word toward each other we will call it a `convergent` bimachine. We can then define three new types of bimachines by considering different scanning directions, as shown in Figure 3: `left sequential`, `right sequential` and `divergent` bimachines.

For example, a right sequential bimachine would be defined as a tuple $B = (Q,P,X,Y,\delta_Q,\delta_P,q_0,p_0,\omega)$ where everything remains defined as for convergent bimachines, except for $\delta_Q : X \times Q \to Q$ which is extended to work on $X^*$ as following: $\delta_Q(\lambda,q) = q$, $\delta_Q(aw,q) = \delta_Q(a,\delta_Q(w,q))$, $\forall q \in Q, a \in X$ and $w \in X^+$. In other words, the left automaton scans the input from right to left this time. Notice that

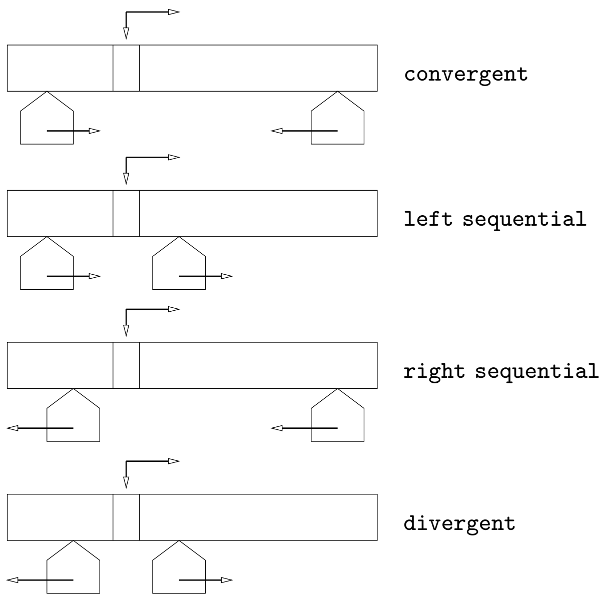


Figure 3: Types of bimachines.

the difference between these four types of bimachines essentially reside in the way the next state (transition) functions are extended. In the next sections we prove that this difference is irrelevant from the point of view of their modelling power.

## 4. Structurally-Reversed Automata

In order to prove the equivalence of these four types of bimachines we first take a closer look at the left and right automata of a convergent bimachine. As mentioned before, these automata can be viewed as deterministic finite automata with all states final and with partial next state functions. Let us consider the convergent bimachine defined in Section 3 and let $(Q, X, \delta_Q, q_0)$ be its left automaton. For an easier formalization we choose to work with complete $DFA$ with useful states (except a *sink* state which - if present - is accessible only), hence consider $A_L := (Q', X, \delta'_Q, q_0, F)$, where

- $Q' := Q \cup \{sink\}$, *sink* being a new state;
- $\delta'_Q : Q' \times X^* \to Q'$ is a total function defined as:

$$\delta'_Q(q, w) = \begin{cases} \delta_Q(q, w), & \text{if } \delta_Q \text{ is defined in } (q, w); \\ sink, & \text{otherwise;} \end{cases}$$

- $F := Q$.

If $\delta_Q$ is a total function in the first place, the above construction is not necessary (in this case, a *sink* state may not even exist).

It is easy to observe that $A_L$ is simply the complete version of the left automaton of $B$. As mentioned, we assume that all states are useful, except possibly a *sink* state. Notice also that the bimachine definition can easily be adapted to operate with complete left and right $DFA$, by changing the domain of the output function.

In order to prove the equivalence of convergent and - for example - right sequential bimachines, one must find a way to modify $A_L$ to scan the input from right to left. This has been proven to be nontrivial, since along with changing the left automaton, one must also adjust the output function of the bimachine in order to preserve its global behaviour.

Let us first prove a result which can very well be viewed as a general automata-theoretic result. Recall that by $\mathcal{L}(A_L)$ we understand the language accepted by $A_L$, and denote by $L^R$ the language obtained from language $L$ by reversing all its words.

**Theorem 5** *Given a complete $DFA$ $A_L = (Q', X, \delta'_Q, q_0, F)$ there exists a complete $DFA$ $A'_L = (Q'', X, \delta''_Q, q'_0, F')$ verifying the following relations:*

*(i)* $\mathcal{L}(A_L) = \mathcal{L}(A'_L)^R$;

*(ii)* $\forall u, v \in X^* : \delta'_Q(q_0, u) \neq \delta'_Q(q_0, v) \Rightarrow \delta''_Q(q'_0, u^R) \neq \delta''_Q(q'_0, v^R)$.

*Proof.* In order to prove this theorem we first prove two interim results. Let $Q'$ consist of $n$ states, $Q' = \{q_0, ..., q_{n-1}\}$. For each $i \in \{0, ..., n-1\}$, let $\equiv_i$ be the equivalence defined as

$$\forall u, v \in X^* : \qquad (u \equiv_i v) \Leftrightarrow \delta'_Q(s_i, u) = \delta'_Q(s_i, v).$$

It is easy to see that $\{\equiv_i\}_{i \in \{0,...,n-1\}}$ is a family of right invariant equivalences of finite index. Then denote by $\equiv_L$ the coarsest equivalence included in all $\equiv_i$, i.e., $\equiv_L := \bigcap_{i=0}^{n-1} \equiv_i$, or in other words

$$\forall u, v \in X^* : \qquad (u \equiv_L v) \Leftrightarrow (u \equiv_i v, \forall i \in \{0, ..., n-1\}).$$

**Lemma 6** *The equivalence $\equiv_L$ is both a left and a right invariant equivalence of finite index.*

*Proof.* $\equiv_L$ is a right invariant equivalence of finite index since it is an intersection of right invariant equivalences of finite index. In order to prove that it is also left invariant, let $u, v \in X^*$ be two equivalent words with respect to $\equiv_L$ (i.e. $u \equiv_L v$) and let us fix an arbitrary word $z \in X^*$. Take now an arbitrary $i \in \{0, ..., n-1\}$ and denote $s_j := \delta'_Q(s_i, z)$. Then $\delta'_Q(s_i, zu) = \delta'_Q(s_j, u)$ and $\delta'_Q(s_i, zv) = \delta'_Q(s_j, v)$. But since $u \equiv_L v$ then $u \equiv_j v$, and from the definition of $\equiv_j$ we have that $\delta'_Q(s_j, u) = \delta'_Q(s_j, v)$ and therefore $\delta'_Q(s_i, zu) = \delta'_Q(s_i, zv)$. In other words we proved that $zu \equiv_i zv$. Since $i$ has been chosen arbitrary from the set $\{0, ..., n-1\}$, it follows that $zu \equiv_L zv$. This proves that $\equiv_L$ is left invariant. □

Let us further define a "reversed equivalence", $\equiv_R$, as following:

$$\forall u, v \in X^* : \qquad (u \equiv_R v) \Leftrightarrow (u^R \equiv_L v^R).$$

**Lemma 7** *The equivalence $\equiv_R$ is both a right and a left invariant equivalence of finite index. Moreover, for any $i \in \{0, ..., n-1\}$ and $u, v \in X^*$, if $\delta'_Q(s_i, u) \neq \delta'_Q(s_i, v)$ then $u^R \not\equiv_R v^R$.*

*Proof.* Let $u, v, z \in X^*$, such that $u \equiv_R v$. Then:

$$u \equiv_R v \Rightarrow u^R \equiv_L v^R \Rightarrow^r u^R z^R \equiv_L v^R z^R \Rightarrow (zu)^R \equiv_L (zv)^R \Rightarrow zu \equiv_R zv,$$

and

$$u \equiv_R v \Rightarrow u^R \equiv_L v^R \Rightarrow^l z^R u^R \equiv_L z^R v^R \Rightarrow (uz)^R \equiv_L (vz)^R \Rightarrow uz \equiv_R vz.$$

The inferences $\Rightarrow^r$ and $\Rightarrow^l$ denote places where we used the property of $\equiv_L$ of being right, respectively left invariant. We then proved that $\equiv_R$ is also right and left invariant. Next, notice that $\delta'_Q(s_i, u) \neq \delta'_Q(s_i, v)$ implies that $u \not\equiv_L v$, hence $u^R \not\equiv_R v^R$. Finally, $\equiv_R$ is of finite index, since its index equals that of $\equiv_L$. □

It is well known that any regular language - hence any $DFA$ - has associated with it a right invariant equivalence of finite index (see Myhill-Nerode Theorem, as in [10, §3.4, Theorem 3.9, p. 65]). Let us consider $\equiv_R$ and construct a corresponding $DFA$ as following. Denote by $\hat{u}$ the equivalence class of $u$ with respect to $\equiv_R$. There exists a finite number of equivalence classes since $\equiv_R$ is of finite index. Denote by $Q''$ the set of all these classes, i.e. $Q'' := \{\hat{u} \mid u \in X^*\}$. Let $\delta''_Q : Q'' \times X^* \to Q''$ be a function defined as $\delta''_Q(\hat{u}, w) := \hat{uw}$. Since $\equiv_R$ is right invariant, the function $\delta''_Q$ is well defined. Consider now the set $F' = \{\hat{u} \mid \delta'_Q(q_0, u^R) \in F\}$ and denote $q'_0 := \hat{\lambda}$. Let now prove that the $DFA$ $A'_L := (Q'', X, \delta''_Q, q'_0, F')$ verifies the conditions of our theorem.

**I.** We first prove that $\mathcal{L}(A_L) = \mathcal{L}(A'_L)^R$. Let $w$ be a word in $\mathcal{L}(A_L)$. Then $\delta'_Q(q_0, w) \in F$, hence $\hat{w^R} \in F'$. This further implies that $\delta''_Q(\hat{\lambda}, w^R) \in F'$, in other words that $w^R \in \mathcal{L}(A'_L)$. This proves that $\mathcal{L}(A_L) \subseteq \mathcal{L}(A'_L)^R$. Conversely, let $w$ be a word of $\mathcal{L}(A'_L)$. Then $\delta''_Q(\hat{\lambda}, w) \in F'$, hence $\hat{w} \in F'$. This implies that $\delta'_Q(q_0, w^R) \in F$, in other words that $w^R \in \mathcal{L}(A_L)$. This proves that $\mathcal{L}(A'_L)^R \subseteq \mathcal{L}(A_L)$, hence the conclusion.

**II.** Next we prove that $\forall u, v \in X^* : \delta'_Q(q_0, u) \neq \delta'_Q(q_0, v) \Rightarrow \delta''_Q(q'_0, u^R) \neq \delta''_Q(q'_0, v^R)$. Notice that this property is not necessarily implied by (i) and that the implication in the opposite direction is not true in general (i.e. we can not interchange $\delta'_Q$ and $\delta''_Q$ in (ii)). We prove this property by contradiction. Assume that there exist two words $u, v \in X^*$ such that $\delta'_Q(q_0, u) \neq \delta'_Q(q_0, v)$ and yet, that $\delta''_Q(q'_0, u^R) = \delta''_Q(q'_0, v^R)$. From the later we derive that $\hat{u^R} = \hat{v^R}$, in other words that $u^R \equiv_R v^R$. This means that $u \equiv_L v$, hence that $\delta'_Q(q_i, u) = \delta'_Q(q_i, v), \forall i \in \{0, ..., n-1\}$. In particular for $i = 0$, this implies that $\delta'_Q(q_0, u) = \delta'_Q(q_0, v)$, contradicting the initial assumption.

Then the above defined automaton $A'_L$ verifies the conditions of Theorem 5. Notice that this proof is constructive, giving a base for an algorithm which finds $A'_L$ for any given $A_L$. □

**Example 8** Let $A_L := (Q', X, \delta'_Q, q_0, F)$, where $Q' = \{q_0, q_1, q_2\}, X = \{a, b\}, F = \{q_1\}$ and $\delta'_Q$ is given by the transition graph in Figure 4 (A). Then the family of equivalences $\{\equiv_i\}_{i \in \{0,1,2\}}$ is given by:

$$(\equiv_0) : X^*/{\equiv_0} = \{\{b^*\}, \{b^*a\}, \{b^*aX^+\}\};$$
$$(\equiv_1) : X^*/{\equiv_1} = \{\{\lambda\}, X^+\};$$
$$(\equiv_2) : X^*/{\equiv_2} = \{\{X^*\}\}.$$

Then, since $\equiv_L = \bigcap_{i=0}^{2} \equiv_i$, we obtain: $X^*/{\equiv_L} = \{\{\lambda\}, \{b^+\}, \{b^*a\}, \{b^*aX^+\}\}$, from which we directly derive $\equiv_R$: $X^*/{\equiv_R} = \{\{\lambda\}, \{b^+\}, \{ab^*\}, \{X^+ab^*\}\}$.

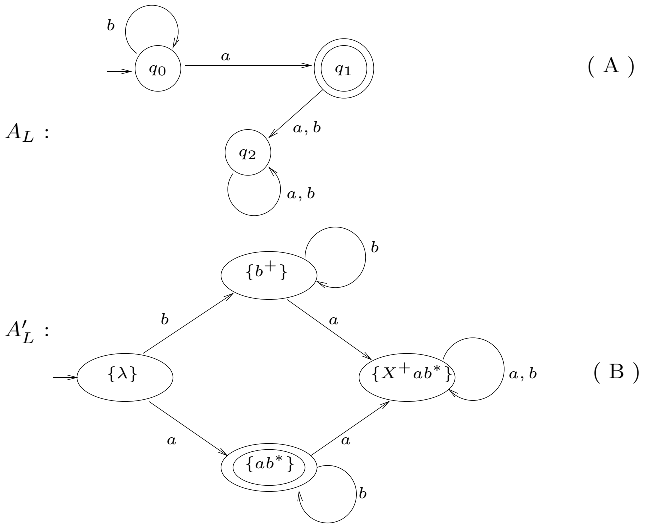


Figure 4: A complete $DFA$ and its corresponding "reversed" automaton.

Then, the automaton $A'_L$ will have $Q'' = \{\{\lambda\}, \{b^+\}, \{ab^*\}, \{X^+ab^*\}\}$ - set of states, $q'_0 = \hat{\lambda}$ - initial state, $F' = \{\{ab^*\}\}$ - set of final states, and the transition function $\delta''_Q$ given by Figure 4 (B). Take for example the words $b$, $ba$ and $bab$. Then $\delta'_Q(q_0, b) \neq \delta'_Q(q_0, ba) \neq \delta'_Q(q_0, bab)$. It is easily verifiable that $\delta''_Q(q'_0, b) \neq \delta''_Q(q'_0, ab) \neq \delta''_Q(q'_0, bab)$. Also, $ab \in \mathcal{L}(A'_L)$ since $ba \in \mathcal{L}(A_L)$.

In the following we give one important property of the automaton constructed in Theorem 5.

**Proposition 9** *Giving an arbitrary* $DFA$ $A_L$*, the corresponding* $DFA$ $A'_L$ *as constructed in the proof of Theorem 5 is a minimal (with respect to the number of states)* $DFA$ *verifying the conditions (i) and (ii) of the theorem.*

*Proof.* Recall that we consider only complete automata with all states accessible (except eventually a *sink* state). Let $A_L$ be an arbitrary complete $DFA$ and $A'_L$ the

$DFA$ constructed in Theorem 5. Proceed by contradiction assuming that there exists a complete $DFA$ $B$ with fewer states than $A'_L$, which verifies the conditions (i) and (ii) of the theorem. Denote by $\delta_B$ the transition function of $B$ and by $q_B$ the initial state of $B$. Since $B$ is smaller than $A'_L$ and since all the states are accessible, there exist two words $u$ and $v$ such that $\delta_B(q_B, u) = \delta_B(q_B, v)$ and $\delta''_Q(q'_0, u) \neq \delta''_Q(q'_0, v)$. The later implies that $u \not\equiv_R v$, hence $u^R \not\equiv_L v^R$ (the notations $\equiv_L$, $\equiv_R$ and $\equiv_i$ have the same meaning as in Theorem 5). Furthermore, we infer that there exists $i \in \{0, ..., n-1\}$ such that $u^R \not\equiv_i v^R$, hence that $\delta'_Q(q_i, u^R) \neq \delta'_Q(q_i, v^R)$ in $A'_l$. But since $q_i$ is accessible, there exists a word $z$ such that $\delta'_Q(q_0, z) = q_i$. Take now the words $zu^R$ and $zv^R$. We have $\delta'_Q(q_0, zu^R) \neq \delta'_Q(q_0, zv^R)$ and $\delta_B(q_B, (zu^R)^R) = \delta_B(q_B, uz^R) = \delta_B(\delta_B(q_B, u), z^R) = \delta_B(\delta_B(q_B, v), z^R) = \delta_B(q_B, vz^R) = \delta_B(q_B, (zv^R)^R)$. We found that $\delta'_Q(q_0, zu^R) \neq \delta'_Q(q_0, zv^R)$ and $\delta_B(q_B, (zu^R)^R) = \delta_B(q_B, (zv^R)^R)$. Since these relations contradict property (ii) of Theorem 5, we proved the inexistence of $B$. □

**Example 10** The example shown in Figure 5 proves that the reciprocal of property (ii) of Theorem 5 does not hold for $A'_L$ - as constructed in Theorem 5. Given the automaton $A_L$ as in Figure 5 (A) we obtain the automaton $A'_L$ as shown in Figure 5 (B).

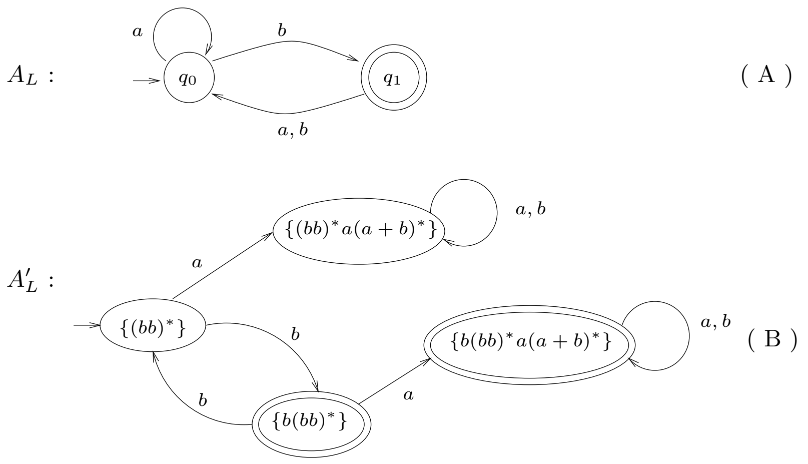


Figure 5: Counter-example for the reciprocal of property (ii), Theorem 5.

Consider the words $bb$ and $ab$. In $A'_L$, $\delta''_Q(q'_0, bb) \neq \delta''_Q(q'_0, ab)$. However, in $A_L$, $\delta'_L(q_0, bb) = \delta'_Q(q_0, ba) = q_0$. A similar situation can be observed in Example 8, when we consider the words $\lambda$ and $b$.

**Definition 11** *We call $A'_L$ a minimal* `structurally-reversed` *automaton of $A_L$.*

One may naturally ask whether there exist more than one minimal structurally-reversed automaton for a given a $DFA$. The following result answers this rather nontrivial question.

**Proposition 12** *There exists a unique (up to an isomorphism) minimal structurally-reversed automaton for a given $DFA$.*

*Proof.* We prove this result by showing that any minimal structurally-reversed automaton of a given $DFA$ is isomorphic with the automaton constructed in Theorem 5. For doing so, let us import the notations used in the mentioned theorem. Consider an arbitrary complete $DFA$ $A_L = (Q', X, \delta', q_0, F)$ having all states useful (except possibly a sink state) and let $A'_L = (Q'', X, \delta'', q'_0, F')$ be the structurally-reversed automaton as previously constructed. We have already proven the minimality of $A'_L$. Assume that there exists another minimal structurally-reversed automaton for $A_L$: $B = (Q_B, X, \delta_B, s_0, F_B)$. Notice that $\mid Q'' \mid = \mid Q_B \mid$ and let $Q'' = \{q'_0, q_1, ..., q_{n-1}\}$, $Q_B = \{s_0, ..., s_{n-1}\}$. We will use letter *"p"* to denote states in $A_L$, *"q"* for states in $A'_L$ and *"s"* for states in $B$.

For each state $q \in Q''$ choose a smallest word $x_q \in X^*$ such that $\delta''(q'_0, x_q) = q$ (actually, the condition of being "a smallest" word is not critical - however, it helps the formalization). Then $x_{q'_0} = \lambda$, and let us define a function $\psi : Q'' \to Q_B$ given by

$$\psi(q) = \delta_B(s_0, x_q).$$

Consequently, $\psi(q'_0) = s_0$. We next prove that $\psi$ is a bijection and in order to do so it suffices to show that $\psi$ is injective (since both $Q''$ and $Q_B$ are finite and have the same number of elements).

Assume that $\psi(q_i) = \psi(q_j)$ for some different states $q_i, q_j \in Q''$. Then $\psi(q_i) = \delta_B(s_0, x_{q_i}) = \psi(q_j) = \delta_B(s_0, x_{q_j})$. But $q_i \neq q_j$ implies that $x_{q_i} \not\equiv_R x_{q_j}$ and $\delta_B(s_0, x_{q_i}) = \delta_B(s_0, x_{q_j})$ implies that $x^R_{q_i} \equiv_0 x^R_{q_j}$ (recall the notations $\equiv_R$ and $\equiv_i$ from Theorem 5). Since $x_{q_i} \not\equiv_R x_{q_j}$, there exists $t \in \{0, ..., n-1\}$ such that $x^R_{q_i} \not\equiv_t x^R_{q_j}$. In other words, there exists a state $p_t$ in automaton $A_L$ such that $\delta'(p_t, x^R_{q_i}) \neq \delta'(p_t, x_{q^R_j})$. Since $p_t$ is accessible, there exists a word $z$ such that $\delta'(q_0, z) = p_t$. It follows that $\delta'(q_0, zx^R_{q_i}) \neq \delta'(q_0, zx^R_{q_j})$. However, notice that this is in contradiction with the fact that $\delta_B(s_0, x_{q_i}z^R) = \delta_B(s_0, x_{q_j}z^R)$ (which holds since $\delta_B(s_0, x_{q_i}) = \delta_B(s_0, x_{q_j})$). Since we have reached a contradiction, we conclude that $\psi$ is injective, hence a bijection.

It remains to prove that $\psi$ is an automata homomorphism (i.e. that it maps initial state into initial state, final states into final states and is compatible with the transition table). Figure 6 is an useful companion of this proof. By the definition of $\psi$, we have that $\psi(q'_0) = s_0$.

Let us now prove that $\psi(\delta''(q, w)) = \delta_B(\psi(q), w)$, for all $q \in Q''$ and $w \in X^*$. Let $q \in Q''$ and $w \in X^*$ arbitrarily taken and denote $q' := \delta''(q, w)$. Then $\psi(\delta''(q, w)) = \psi(q') = \delta_B(s_0, x_{q'})$ and $\delta_B(\psi(q), w) = \delta_B(\delta_B(s_0, x_q), w) = \delta_B(s_0, x_q w)$. It then remains to prove that $\delta_B(s_0, x_{q'}) = \delta_B(s_0, x_q w)$. Assume by contradiction that $s' := \delta_B(s_0, x_{q'})$ is different from $s'' := \delta_B(s_0, x_q w)$. Denote $q'' := \psi^{-1}(s'')$. Since $\psi^{-1}$ is injective, $q' \neq q''$ hence $x_q w \not\equiv_R x_{q''}$. Then $w^R x^R_q \not\equiv_k x^R_{q''}$ for some $k \in \{0, ..., n-1\}$, hence there exists a word $z$ such that $\delta'(q_0, zw^R x^R_q) \neq \delta'(q_0, zx^R_{q''})$ in $A_L$. But this would mean that $\delta_B(s_0, x_q w z^R) \neq \delta_B(s_0, x_{q''} z^R)$ in $B$ which is a contradiction with the fact that $\delta_B(s_0, x_q w) = \delta_B(s_0, x_{q''})$. We reached this contradiction by assuming that $\delta_B(s_0, x_{q'}) \neq \delta_B(s_0, x_q w)$; hence the equality holds. We conclude that $\psi(\delta''(q, w)) = \delta_B(\psi(q), w), \forall q \in Q''$ and $w \in X^*$.

Finally, notice that $q \in F' \Rightarrow \psi(q) \in F_B$, from the fact that $A'_L$ and $B$ recognize the same language. This completes the proof, that $\psi$ is an automata homomorphism. Hence $A'_L$ and $B$ are isomorphic. □

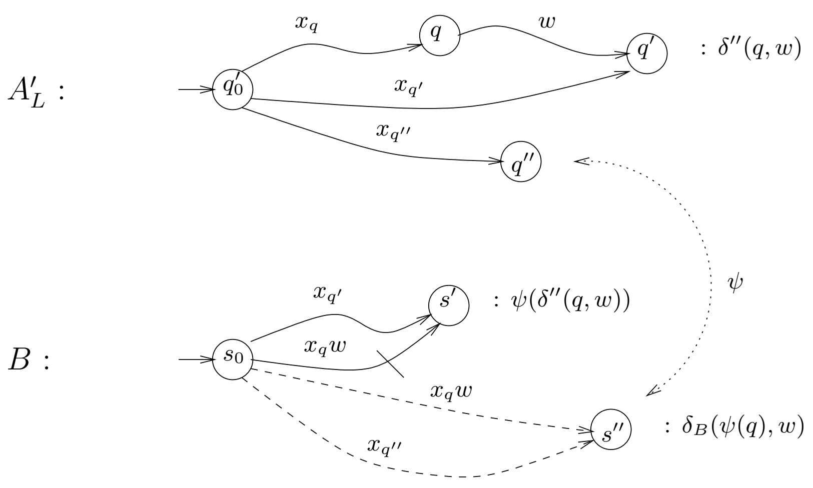


Figure 6: Companion for the proof of Proposition 12.

Notice that the structurally reversed automaton $A'_L$ can be modified to scan the input from right to left, hence accepting the same language as $A_L$. Moreover, if two input words lead to different states in $A_L$, then the same input words lead to different states in this modified version of $A'_L$. This construction will be detailed in the next section.

Observe also that the structurally-reversed automaton of a given $DFA$ is more "powerful" than a plain reversed automaton - which simply accepts the reverse of the given language. Indeed, if two words are "state-discriminated" by the given $DFA$, then the reversed words are state-discriminated by its structurally-reversed automaton. This observation is central to the proof of bimachine equivalence.

**Definition 13** *Let* $A_L = (Q', X, \delta'_Q, q_0, F)$ *be a* $DFA$ *and let* $A'_L = (Q'', X, \delta''_Q, q'_0, F')$ *be its minimal structurally-reversed automaton (as previously defined). The* `structural connection` *between* $A_L$ *and* $A'_L$ *is the function* $\nu : Q' \to \mathcal{P}(Q'')$ *given by:*

$$\nu(q) = \{q' \in Q'' \mid \exists u \in X^* : \delta'_Q(q_0, u) = q \text{ and } \delta''_Q(q'_0, u^R) = q'\}.$$

In this definition we denoted by $\mathcal{P}(Q'')$ the powerset (set of all subsets) of $Q''$.

**Proposition 14** *The image of* $\nu$ *is a partition of* $Q''$.

*Proof.* It is clear that the image of $\nu$ covers $Q''$. Indeed, given a state $q' \in Q''$, choose an arbitrary word $u$ such that $\delta''_Q(q'_0, u) = q'$ (such word always exists, since the construction of $A'_L$ ensures that all its states are accessible). Then clearly $q' \in \nu(\delta'_Q(q_0, u^R))$. Next, let us prove that $\nu(q_1) \cap \nu(q_2) = \emptyset$ for any two different states $q_1, q_2 \in Q'$. Suppose (by contradiction) that there exists $q' \in \nu(q_1) \cap \nu(q_2)$. Then by the definition of $\nu$ there exist two different words $u_1$ and $u_2$ such that $\delta'_Q(q_0, u_1) = q_1$, $\delta'_Q(q_0, u_2) = q_2$ and $\delta''_Q(q'_0, u_1^R) = \delta''_Q(q'_0, u_2^R) = q'$. However, one can easily observe that these relations contradict the definition of a structurally-reversed automaton (condition (ii) of Theorem 5). □

**Example 15** Considering the automata described in Example 8, we obtain the following structural connection:

$$\nu_8(\{q_0\}) = \{\{\lambda\}, \{b^+\}\}, \nu_8(\{q_1\}) = \{\{ab^*\}\}, \nu_8(\{q_2\}) = \{\{X^+ab^*\}\},$$

and considering the automata described in Example 10, we obtain:

$$\nu_{10}(\{q_0\}) = \{\{(bb)^*\}, \{(bb)^*a(a+b)^*\}\},$$

$$\nu_{10}(\{q_1\}) = \{\{b(bb)^*\}, \{b(bb)^*a(a+b)^*\}\}.$$

The structural connection can actually be defined for any $DFA$ and any associated structurally-reversed automaton(hence not necessarily minimal), and yet the property of Proposition 14 will still hold.

## 5. Bimachine Equivalence

We now have all ingredients for proving one of the main results of this paper, namely that all types of bimachines defined in Section 3 are equivalent (two bimachines are equivalent if they realize the same rational function).

**Theorem 16** *For any bimachine of type $\mathcal{A}$ there exists an equivalent bimachine of type $\mathcal{B}$, where*

$$\mathcal{A}, \mathcal{B} \in \{\text{“convergent”}, \text{“left sequential”}, \text{“right sequential”}, \text{“divergent”}\}.$$

*Proof.* Notice that this theorem essentially says that the scanning directions of the reading heads of a bimachine are irrelevant. Let $f_B : X^* \to Y^*$ be a rational function realized by a convergent bimachine $B = (Q, P, X, Y, \delta_Q, \delta_P, q_0, p_0, \omega)$. In the following we prove that there exists a right sequential bimachine $B' = (Q'', P, X, Y, \delta^R, \delta_P, q'_0, p_0, \omega^R)$ realizing the same function $f_B$. The reciprocal of this property as well as the equivalence among other types of bimachines are proved in a similar way and will be omitted. Consider the left automaton $(Q, X, \delta_Q, q_0)$ of $B$ together with its complete version, $A_L = (Q', X, \delta'_Q, q_0, F)$. We first construct the minimal structurally-reversed automaton of $A_L$, namely $A'_L = (Q'', X, \delta''_Q, q'_0, F')$ (as detailed in Theorem 5). The minimality of $A'_L$ is not crucial; however, it makes the formalization easier. We noticed earlier that $A'_L$ can be modified to scan the input from right to left and accept exactly $\mathcal{L}(A_L)$. Indeed, construct the automaton $A_L^R := (Q'', X, \delta^R, q_0', F')$, where $\delta^R : X^* \times Q'' \to Q''$ is defined (extended) as following:

$$\delta^R(\lambda, q) = q,\ \delta^R(w, q) := \delta_Q''(q, w^R),\ \delta^R(aw, q) := \delta^R(a, \delta^R(w, q)).$$

Notice that the extension of $\delta^R$ implies a right to left direction of scanning for the reading head of automaton $A_L^R$.

**Fact** $\mathcal{L}(A_L^R) = \mathcal{L}(A_L) = \mathcal{L}(A_L')^R$. *Moreover, if* $\delta_Q'(q_0, u) \neq \delta_Q'(q_0, v)$ *for two words* $u, v$, *then subsequently* $\delta^R(u, q_0') \neq \delta^R(v, q_0')$.

*Proof.* The proof of this fact is straightforward and is left to the reader. □

Consider now the left sequential bimachine $B' = (Q'', P, X, Y, q_0', p_0, \delta^R, \delta_P, \omega^R)$, where $\omega^R : Q'' \times X \times P \to Y^*$ is given by:

$$\omega^R(q, a, p) = \begin{cases} \omega(q', a, p), & \text{if } q \in \nu(q') \text{ and } \omega(q', a, p) \text{ is defined ;} \\ undefined, & \text{otherwise.} \end{cases}$$

In the above definition we have used $\nu$, the structural connection between $A_L$ and $A_L'$. Then $\omega^R$ is extended to work on $Q'' \times X^* \times P$ in the usual way. It is clear that bimachine $B'$ is right sequential and well defined. Let us prove now that the function $f_{B'}$ realized by $B'$ is the same as function $f_B$. Take an arbitrary word $w \in X^*$. We distinguish the following three relevant cases:

**Case I.** There exists a factorization $w = w_1 a w_2$ with $w_1$ a proper prefix of $w$, such that $\delta_Q(q_0, w_1)$ is undefined (hence $f_B$ is undefined in $w$). This implies that $\delta_Q'(q_0, w_1) = sink$. Then , since $\delta^R(w_1, q_0') = \delta_Q''(q_0', w_1^R) \in \nu(\delta_Q'(q_0, w_1)) = \nu(sink)$ and since $\omega$ is not defined in $(sink, a, \delta_P(w_2, p_0))$, it follows that $\omega^R(\delta^R(w_1, q_0'), a, \delta_P(w_2, p_0))$ is undefined, hence that $f_{B'}$ is undefined in $w$ as well.

**Case II.** There exists a factorization $w = w_1 a w_2$ with $w_1$ a proper prefix of $w$, such that $\delta_Q(q_0, w_1)$ and $\delta_P(w_2, p_0)$ are both defined and $\omega(\delta_Q(q_0, w_1), a, \delta_P(w_2, p_0))$ is undefined. Then $\delta_Q'(q_0, w_1) = \delta_Q(q_0, w_1) \neq sink$, hence $\omega^R(\delta^R(w_1, q_0'), a, \delta_P(w_2, p_0)) = \omega(\delta_Q(q_0, w_1), a, \delta_P(w_2, p_0))$, which is undefined.

**Case III.** $f_B$ is defined in $w$. By the definition of $B$, $f_B(w) = \omega(q_0, w, p_0)$. Consider $w = a_1 a_2 ... a_k$, with $a_i \in X, \forall i \in \{1, ...k\}$. Then

$$f_B(w) = \omega(q_0, a_1, \delta_P(a_2 ... a_k, p_0))$$

$$\omega(\delta_Q(q_0, a_1), a_2, \delta_P(a_3 ... a_k, p_0)) ... \omega(\delta_Q(q_0, a_1 ... a_{k-1}), a_k, p_0),$$

by the definition of $\omega$. Notice now that for any $i \in \{1, ..., k-1\}$:

$$\delta^R(a_1 ... a_i, q_0') = \delta_Q''(q_0, a_i ... a_1) \in \nu(\delta(q_0, a_1 ... a_i)),$$

hence for $k \geq 3$:

$$\omega^R(\delta^R(a_1...a_i, q'_0), a_{i+1}, \delta_P(a_{i+2}...a_k, p_0)) =$$

$$= \omega(\delta(q_0, a_1...a_i), a_{i+1}, \delta_P(a_{i+2}...a_k, p_0)),$$

from the definition of $\omega^R$. It is now easy to check that $\omega(q_0, w, p_0) = \omega^R(q'_0, w.p_0)$, hence that $f_B(w) = f_{B'}(w)$.

All other cases are either similar to the above ones or they can easily be proven. Concluding, we found a right sequential bimachine $B'$ equivalent to the convergent bimachine $B$, i.e. such that $f_{B'} = f_B$. Similar constructions lead to the conversion of bimachines of a given type to a bimachine of any other type. Notice that the core of this conversion is the construction of a structurally-reversed automaton of a given $DFA$ and the use of the structural connection between the automaton and its structurally-reversed counterpart.

Finally notice that if we want to convert for example a convergent bimachine into a left sequential bimachine, we need to "structurally reverse" a right automaton - which is a $DFA$ which scans the input from right to the left. With care, one can adapt the construction of structurally-reversed automata to this situation as well. In this case a corresponding structurally-reversed automaton will scan the input in the common way, from left to right. $\square$

## 6. Simulating Bimachines by *GSM*

In this section we give a representation of rational functions which leads to a method of simulating bimachines by means of left sequential transducers (or $GSM$).

**Definition 17** *[1, p. 96] A* `left sequential transducer` *is a tuple* $L = (Q, X, Y, \delta, q_0, \gamma)$ *where*

*(i)* $Q$ *is a finite set of states,* $q_0$ *is an initial state and* $X, Y$ *are input and output alphabets;*

*(ii)* $\delta : Q \times X \to Q$ *is a partial next state function;*

*(iii)* $\gamma : Q \times X \to Y^*$ *is a partial output function with the same domain as* $\delta$ *(notation wise,* $dom(\gamma) = dom(\delta)$*).*

The functions $\delta$ and $\gamma$ are extended in the usual way to operate on words. Accordingly, for the output function we have: $\gamma(q, \lambda) = \lambda$ and $\gamma(q, wa) = \gamma(q, w)\gamma(\delta(q, w), a)$ where $a \in X$, $w \in X^+$ and $q \in Q$. In essence, the left sequential transducer is a $DFA$(incomplete, with all its states final) with output words associated to its transitions. While scanning the input word, this automaton sequentially writes on the output tape all the output words associated to those transitions triggered by the input. Formally, the partial function realized by $L$ is $f_L : X^* \to Y^*$ given by

$$f_L(w) := \gamma(q_0, w).$$

Notice that we can relax the above definition - without loss of generality - by allowing $dom(\gamma) \subseteq dom(\delta)$. Indeed, the transitions where $\gamma$ is not defined can eventually be ignored/discarded. In this section we construct a left sequential transducer in which this situation occurs and where the corresponding adjustments are omitted.

A left sequential transducer is also called a generalized sequential machine ([7, 5]) or $GSM$ for short.

The family of all partial functions realized by left sequential transducers is called the family of `left sequential functions`(or sequential functions - if no confusion arises) and can be proved that this family is strictly included in the family of rational functions. In other words, bimachines are strictly more powerful than $GSM$ (for example, the rational function used in Example 20 is neither sequential nor subsequential; also recall the hierarchy in Figure 1).

Let $w = a_1...a_k \in X^*$ with $k \geq 1$ and $a_i \in X, \forall i \in \{1, ..., k\}$. By the `trimming` of $w$ we understand the ordered sequence $(a_1...a_{k-1}a_k,\ a_2...a_{k-1}a_k,\ ...,\ a_{k-1}a_k,\ a_k)$.

**Definition 18** *By a* `trimming` *over* $X$ *we understand a total function* $\mu_\$$*, given by*

$$\mu_\$ : X^* \to (X \cup \{\$\})^*,$$

$$\forall k \geq 1 : \ \mu_\$(a_1...a_k) = a_1...a_k\$a_2...a_k\$...\$a_{k-1}a_k\$a_k\$,$$

*where* \$ *is a symbol not in* $X$*. By convention,* $\mu_\$(\lambda) = \lambda$*.*

Notice that a trimming over $X$ is simply a global description of the trimming of all words of $X^*$.

**Theorem 19** *If* $f : X^* \to Y^*$ *is a rational function such that* $f(\lambda) = \lambda$*, then there exists a left sequential function* $f_L : (X \cup \{\$\})^* \to Y^*$ *such that* $f = f_L \circ \mu_\$$*.*

*Proof.* In proving this result we make use of bimachine equivalence presented in the previous section. Accordingly, any rational function can be realized by a left sequential bimachine. Let $B = (Q, P, X, Y, \delta_Q, \delta_P, q_0, p_0, \omega)$ be a left sequential bimachine realizing $f$. As previously mentioned, we can assume that both the left and the right automata of $B$ are complete (we can always enforce this situation, by tweaking the domain of the output function). Then $\delta_Q$ and $\delta_P$ are total functions, $\omega$ is a partial function and the automata composing this bimachine can be viewed as complete $DFA$ with all states final (i.e. themselves alone do not reject any input).

Let us first focus on the left $DFA$, $A_L = (Q, X, \delta_Q, q_0)$. We modify this automaton in order to process \$. Notice that given $(a_1...a_k, a_2...a_k, ..., a_k)$ a trimming of a word $w$, if we consider only the first symbol of each of its components we obtain a "trace" of $w$: $(a_1, a_2, ..., a_k)$. Based on this observation we construct a new $DFA$ $A'_L = (Q', X \cup \{\$\}, \delta'_Q, q_0)$ where

- $Q' = Q \cup (Q \times X)$;

- $\delta'_Q : Q' \times (X \cup \{\$\}) \to Q'$, given by:

$$\delta'_Q(r,x) = \begin{cases} (r,x), & \text{if } x \in X \text{ and } r \in Q; \\ r, & \text{if } x \in X \text{ and } r \in Q \times X; \\ \delta_Q(q,a), & \text{if } x = \$ \text{ and } r = (q,a) \in Q \times X; \\ undefined, & \text{otherwise.} \end{cases}$$

Notice that $\delta'_Q$ is designed in such way that the behaviour of $A'_L$ when scanning $\mu_\$(w)$ simulates the behaviour of $A_L$ when scanning $w$ (in $B$). Indeed if for some input $w = a_1a_2$ for example, $A_L$ executes the following computation:

$$q_1a_1a_2 \vdash q_2a_2 \vdash q_3,$$

then $A'_L$ will execute the following computation for the input $\mu_\$(w) = a_1a_2\$a_2\$$:

$$q_1a_1a_2\$a_2\$ \vdash (q_1,a_1)a_2\$a_2\$ \vdash (q_1,a_1)\$a_2\$ \vdash q_2a_2\$ \vdash (q_2,a_2)\$ \vdash q_3.$$

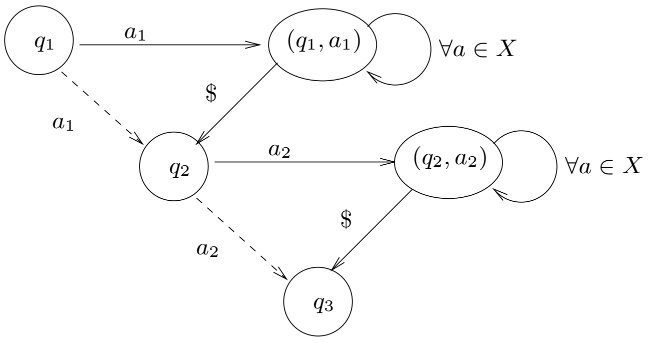


Figure 7: Relationship between $\delta_Q$ and $\delta'_Q$.

Notice that $A'_L$ memorizes the "trace" of $w$, performs "useful" transitions only triggered by the symbol \$ and "skips" the other symbols. Figure 7 illustrates the relationship between $\delta_Q$ and $\delta'_Q$: dotted lines represent old transitions of $A_L$ and solid lines represent the new corresponding transitions of $A'_L$. We let $A'_L$ be an incomplete $DFA$.

Next, let us modify the right $DFA$ $A_R$ to allow it to process \$. Unlike $A'_L$ which was designed to simulate $A_L$ on a single scan, $A'_R$ will simulate computations of $A_R$ on multiple scans. More specifically, let $A_R = (P, X, \delta_P, p_0)$ be the right automaton of $B$. Since $B$ is a left sequential bimachine, $A_R$ is an "usual" $DFA$, scanning the input from left to right. Consider the automaton $A'_R = (P', X \cup \{\$\}, \delta'_P, p'_0)$ where

- $P' = P \cup \{p'_0\}$, with $p'_0$ a new, initial state;

- $\delta'_P : P' \times (X \cup \{\$\}) \to P'$ given by:

$$\delta'_P(r,x) = \begin{cases} p_0, & \text{if } x \in X \text{ and } r = p'_0; \\ \delta_P(r,x), & \text{if } x \in X \text{ and } r \in P; \\ p'_0, & \text{if } x = \$ \text{ and } r \in P. \\ undefined, & otherwise. \end{cases}$$

The automaton $A'_R$ "skips" the symbols immediately following a \$ (and the first symbol of the input). For all the other symbols up to another \$, $A'_R$ simulates the computations of $A_R$. Each scanned \$ resets the automaton to its new initial state. Figure 8 illustrates the design of $A'_R$. The dotted rectangle is a replica of the transition graph of $A_R$; in addition each state in $P$ has an \$-transition into $p'_0$. As in the case of $A'_L$, $A'_R$ is an incomplete $DFA$.

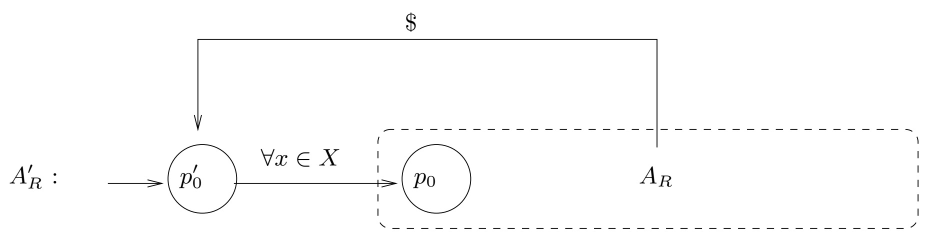


Figure 8: The construction of $A'_R$.

We now construct a left sequential transducer corresponding to the left sequential function required by the theorem. We do this by basically constructing a machine which "runs in parallel" $A'_L$ and $A'_R$, and to which we augment a simple output function. Indeed, consider the left sequential transducer $L = (Q_L, X \cup \{\$\}, Y, \delta_L, q_0^L, \gamma)$ detailed as following:

- $Q_L = Q' \times P'$;
- $q_0^L = (q_0, p'_0)$;
- $\delta_L : Q_L \times (X \cup \{\$\}) \to Q_L$, a partial next state function given by:

$$\delta_L((q,p),a) = \begin{cases} (\delta'_Q(q,a), \delta'_P(p,a)), & \text{if } \delta'_Q \text{ and } \delta'_P \text{ are defined in } (q,a); \\ undefined, & \text{otherwise.} \end{cases}$$

- $\gamma : Q_L \times (X \cup \{\$\}) \to Y^*$, a partial output function given by:

$$\gamma((r,p),x) = \begin{cases} \lambda, & \text{if } x \in X; \\ \omega(q,a,p), & \text{if } x = \$, r = (q,a) \in Q \times X, \\ & p \in P \text{ and } \omega \text{ is defined in } (q,a,p); \\ undefined, & \text{otherwise.} \end{cases}$$

Recall that we allow the domain of $\gamma$ to be strictly included in the domain of $\delta_L$ - in practice, all transitions for which $\gamma$ is undefined are discarded.

It remains to prove that if $f_L$ is the function realized by $L$ then $f_L \circ \mu_\$ = f$. Arbitrarily choosing a word $w = a_1a_2....a_k \in X^*$, we distinguish the following cases:

(i) if $k = 0$, then $f(\lambda) = \lambda$ and $f_L(\mu_\$(\lambda)) = f_L(\lambda) = \gamma(q_0^L, \lambda) = \lambda$;

(ii) if $k = 1$, then $w = a \in X$ and $f_L(\mu_\$(a)) = f_L(a\$) = \gamma(q_0^L, a\$)$, which is equal to $\omega(q_0, a, p_0)$ if $\omega$ is defined in $(q_0, a, p_0)$ or is undefined otherwise - in both cases being equal to $f(a)$;

(iii) the case when $k \geq 2$ is discussed in the following.

Let us define a "cut" of $w$ to be a factorization of $w$ as $w_1a_iw_2$. Corresponding to this cut we consider the following factorization of $\mu_\$(w)$:

$$\mu_\$(w) = u_1vu_2, \text{ where } u_1 \text{ ends in a \$ and } v = a_i...a_k\$.$$

These cut and factorization are illustrated in Figure 9.

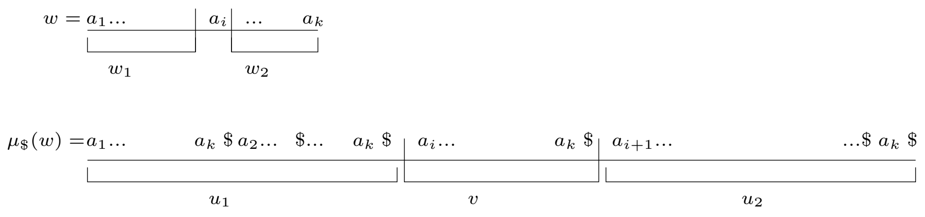


Figure 9: A "cut" of $w$, and the corresponding factorization of $\mu_\$(w)$.

**Case 1.** $f$ is undefined in $w$. Then there exists a cut as above such that $\omega(\delta_Q(q_0, w_1), a_i, \delta_P(p_0, w_2))$ is undefined (recall that $\delta_Q$ and $\delta_P$ are complete functions). Consider $w_1$ to be the smallest prefix of $w$ such that the above holds. Then also $\gamma(q_0^L, u_1v)$ is undefined since otherwise it should have $\omega(\delta_Q(q_0, w_1), a_i, \delta_P(p_0, w_2))$ as a suffix - which is undefined. Then $\gamma(q_0^L, \mu_\$(w))$ is undefined as well, hence $f_L$ is undefined in $\mu_\$(w)$.

**Case 2.** If $f$ is defined in $w$ then it is clear than $f_L$ is defined in $\mu_\$(w)$ as well. We next prove that in this case $f(w) = f_L(\mu_\$(w))$. We expand the function $\gamma$ in $\mu_\$(w)$:

$$\gamma(q_0^L, \mu_\$(w)) = \gamma(q_0^L, a_1...a_k\$)\gamma(\delta_L(q_0^L, a_1...a_k\$), a_2...a_k\$)...$$
$$...\gamma(\delta_L(q_0^L, a_1...a_k\$...\$a_{k-1}a_k\$), a_k\$).$$

Notice that if a word $u$ which ends in a \$ is a prefix of $\mu_\$(w)$, then $\delta_L(q_0^L, u)$ is a state of the form $(q, p_0')$ where $q \in Q$. Consider the situation when the transducer $L$ reaches such state $(q, p_0')$, and assume that the remaining of the input has the prefix $a_j...a_k\$$ for some $j < k$. In this situation, $L$ will next execute the following computation (we ignore the output for the time being):

$$(q, p_0')a_j...a_k\$ \vdash ((q, a_j), \delta_p(p_0, a_{j+1}...a_k))\$ \vdash (\delta_Q(q, a_j), p_0').$$

The only output of this computation is written in the last step and is exactly $\omega(q, a_j, \delta_P(p_0, a_{j+1}...a_k))$. In other words,

$$\gamma((q, p'_0), a_j...a_k\$) = \omega(q, a_j, \delta_P(p_0, a_{j+1}...a_k)), \forall q \in Q.$$

Since $q_0^L = (q_0, p'_0)$, $\delta_L(q_0^L, a_1...a_k\$) = (\delta_Q(q_0, a_1), p'_0)$ and so forth up to $\delta_L(q_0^L, a_1...a_k\$...\$a_{k-1}a_k\$) = (\delta_Q(q_0, a_1...a_{k-1}), p'_0)$, it is an easy exercise to apply the above relation to the expansion of $\gamma(q_0^L, \mu_\$(w))$, hence yielding

$$\gamma(q_0^L, \mu_\$(w)) = \omega(q_0, a_1, \delta_P(p_0, a_2...a_k))\omega(\delta_Q(q_0, a_1), a_2, \delta_P(p_0, a_3...a_k))...$$
$$...\omega(\delta_Q(q_0, a_1...a_{k-1}), a_k, p_0),$$

this being exactly $\omega(q_0, w, p_0)$, i.e. $f(w)$. We then proved that $f(w) = f_L(\mu_\$(w))$. □

This theorem basically says that bimachines can successfully be replaced/simulated by *GSM*, at a small cost. The cost is given by a simple preprocessing of the input words, namely a trimming.

**Remark** A trimming $\mu_\$$ over some alphabet $X$ is itself a word function which can trivially be implemented in practice. However, $\mu_\$$ can not be realized by any finite or push-down transducer (defined in [8], for example) - in other words it is neither a rational nor an algebraic function (see [1, p. 71] for a definition of algebraic transductions). In order to support this remark, we recall two properties of such functions.

1. Each rational function preserves regular languages. This property can be derived from Nivat's characterization of rational relations ([19], [1, Theorem 4.1, p. 66]).
2. The image of a regular language through a push-down transducer is context-free ([8, Theorem 3.3, p. 170]).

Now it suffices to observe that $\mu_\$(X^*)$ is neither regular, nor context-free, fact easily proven using the pumping lemma for either regular or context-free languages.

In the following we give an example of simulating bimachines by *GSM*.

**Example 20** Let us consider a classical example of a rational function which is not sequential. Consider $f : \{x\}^* \to \{a, b\}^*$, given by

$$f(x^n) = \begin{cases} a^n, & \text{if } n \text{ is an even natural number;} \\ b^n, & \text{if } n \text{ is odd.} \end{cases}$$

The fact that $f$ is rational is proven by the single-valued transducer which realizes $f$, shown in Figure 10 . We follow the usual convention, that a label "$x/w$" of a transition implies that the transition is triggered by a symbol $x$ and it writes $w$ on an output tape. For a general discussion on finite transducers consult [1, p. 77].

Function $f$ is not sequential since it is not prefix-preserving, i.e. given two words $u$ and $v$ such that $f$ is defined in both $u$ and $uv$, then $f(u)$ is not necessarily a prefix of $f(uv)$. However, notice that sequential functions are always prefix-preserving. This means that although there exists a single-valued transducer which realizes $f$, it surely

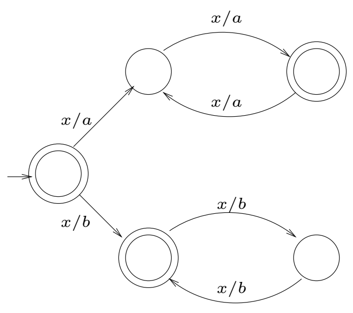
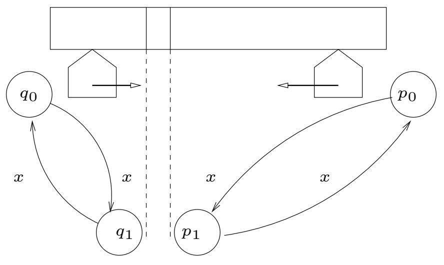


Figure 10: A transducer for function $f$.

does not exist a (left) sequential transducer for $f$. However, by Theorem 19 we can construct a sequential transducer which simulates $f$, i.e. in our case it realizes $f$ modulo a preprocessing.

Since $f$ is rational, it means that there also exists a bimachine realizing $f$. Indeed, the convergent bimachine in Figure 11 realizes $f$. Notice that due to its symmetry(one-letter input alphabet), this machine can readily be converted into a left sequential bimachine. Indeed, for this conversion, one needs to simply change the scanning direction of its right automaton without modifying anything else.



| $\omega$ | $p_0$ | $p_1$ |
|---|---|---|
| $q_0$ | $b$ | $a$ |
| $q_1$ | $a$ | $b$ |

Figure 11: A bimachine for function $f$.

Following the method presented in Theorem 19 we construct the sequential transducer shown in Figure 12. It is an easy exercise to verify that indeed, this *GSM* realizes $f$ when applied on the trimming of the input. Then we can use this machine and a preprocessing (trimming) of the input in order to implement $f$.

Let $\gamma$ be the output function of the sequential transducer shown in Figure 12 and consider the words $u = x^3$ and $v = x^4$. Then, $f(u) = b^3$ and $f(v) = a^4$ - as also computed by the transducer shown in Figure 10 or by the bimachine shown in Figure 11. Since $\mu_\$(u) = xxx\$xx\$x\$$ and $\mu_\$(v) = xxxx\$xxx\$xx\$x\$$, the sequential transducer applied to each of these two trimmings issues the following output:

$$\gamma([q_0,p_0'],\mu_\$(u)) = \gamma([q_0,p_0'],xxx\$)\gamma([q_1,p_0'],xx\$)\gamma([q_0,p_0'],x\$) =$$
$$= \gamma([(q_0,x),p_0],\$)\gamma([(q_1,x),p_1],\$)\gamma([(q_0,x),p_0],\$) = bbb = f(u),$$

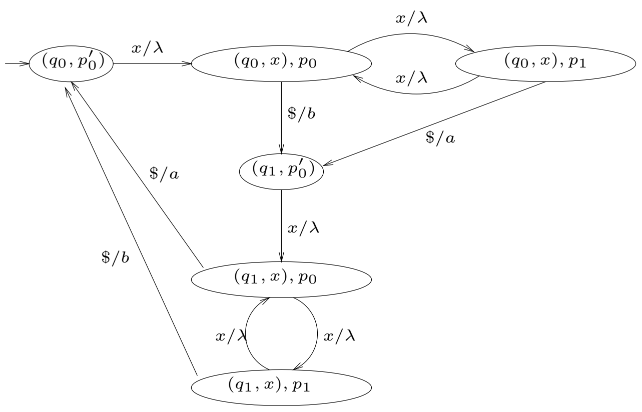


Figure 12: A left sequential transducer which simulates function $f$.

$$\begin{aligned}
&\gamma([(q_0, p'_0)], \mu_\$(v)) = \\
&\quad = \gamma([q_0, p'_0], xxxx\$)\gamma([q_1, p'_0], xxx\$)\gamma([q_0, p'_0], xx\$)\gamma([q_1, p'_0], x\$) = \\
&\quad = \gamma([(q_0, x), p_1], \$)\gamma([(q_1, x), p_0], \$)\gamma([(q_0, x), p_1], \$)\gamma([(q_1, x), p_0], \$) = \\
&\quad = aaaa = f(v)
\end{aligned}$$

## 7. Conclusions and Further Research

Bimachines have been designed in [23] for the purpose of implementing rational functions. Their initial goal was to model in a concise manner the computations performed by a cascade of a left and a right sequential transducer. Indeed, the decomposition theorem for rational functions is easily proved using classical bimachines. This was probably the main reason why Schutzenberger chose the scanning directions of the reading heads in a bimachine to be the way they are known today. In this paper we proved that in fact these scanning directions are irrelevant. Furthermore, we give a method of changing the scanning direction of any reading head of a bimachine. To this respect, we introduced the new concept of structurally-reversed automaton. Such an automaton realizes more than a language reversal: it preserves the state-discrimination among words. Although a few properties of this type of automata have been studied, there is more to be found. Moreover these automata may have other useful applications (for example, in reversing Moore machines). Left for future work is to find an efficient algorithm for computing the minimal structurally-reversed automaton of a given $DFA$ (in this paper we gave a basis for such algorithm).

The equivalence between convergent and left sequential bimachines helped us give a simple method of simulating a bimachine by means of only one sequential transducer (fact which may seem surprising if we consider that the decomposition of rational functions necessarily requires two sequential transducers: one left sequential, the other right sequential). In our method, the input word is first preprocessed in a simple way (we perform a trimming), then is passed to a sequential transducer which would normally realize a sequential function. However, we found that such setup can realize any given rational function.

In parallel with the technical aspect of these results, we convey a good intuition about the gap between sequential and rational functions. To the author in particular, it appears that these two families of functions are closer than they may seem at the first glance. This fact may explain why the hierarchy of rational functions is so succinct and indeed tight (see Figure 1). However, it remains to investigate whether there exist deterministic, sequential ( , retrospective) and finitary automata with output more powerful than subsequential transducers (notice that neither bimachines nor single-valued transducers match this profile). In a broader sense, it may be worth looking at ways to refine the hierarchy of rational functions.

**Acknowledgements**

I thank Gabriel Thierrin and Sheng Yu for reading the material and drawing useful observations. I also thank Christian Choffrut for making available his These de Doctorat d'Etat.

## References

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

[2] C. Choffrut, *Contribution a l'Etude de quelques Familles Remarquables de Fonctions Rationnelles.* These d'Etat, Universite Paris VII, 1978.

[3] N. Chomsky, Three Models for the Description of Languages. *IRE Transaction on Information Theory* **2** (1956), 113–124.

[4] N. Chomsky, On Certain Formal Properties of Grammars. *Information and Control* **2** (1959), 137–167.

[5] S. Eilenberg, *Automata, Languages and Machines.* Vol. A, Academic Press, New York and London, 1974.

[6] C. C. Elgot, J. E. Mezei, On Relations Defined by Generalized Finite Automata. *IBM Journal of Research and Development* **9** (1965), 47–65.

[7] S. Ginsburg, *The Mathematical Theory of Context-Free Languages.* McGraw-Hill Book Co., New York, 1966.

[8] S. Ginsburg, G. F. Rose, Preservation of Languages by Transducers. *Information and Control* **9** (1966), 153–176.

[9] T. Head, A. Weber, Deciding Code Related Properties by Means of Finite Transducers. In: R. Capocelli(ed.), *Proc. Sequences II: Methods in Communications, Security and Computer Science* **II** (1993), 260–272.

[10] J. Hopcroft, J. Ullman, *Introduction to Automata Theory, Languages, and Computation.* 1st Edition, Addison-Wesley, Massachusetts and London, 1979.

[11] S. C. Kleene, Representation of Events in Nerve Nets and Finite Automata. *Annals of Mathematics Studies* **34** (1956), 2–42.

[12] S. Konstantinidis, Transducers and the Properties of Error-Detection, Error-Correction, and Finite-Delay Decodability. *Journal of Universal Computer Science* **8**, no.2 (2002), 278–291.

[13] V. Manca, C. Martin-Vide, Gh. Paun, Iterated GSM Mappings: a Collapsing Hierarchy. In: *Jewels are Forever, Contribution on Theoretical Computer Science in Honor of Arto Salomaa.* Springer, 1999.

[14] W. S. McCulloch, W. H. Pitts, A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics* **5** (1943), 115–133.

[15] J. D. McKnight, Kleene Quotient Theorems. *Pacific Journal of Mathematics* **14** (1964), 1343–1352.

[16] G. H. Mealy, A Method for Synthesizing Sequential Circuits. *Bell System Technical Journal* **34** (1955), 1045–1079.

[17] M. Mohri, Finite-State Transducers in Language and Speech Processing. *Computational Linguistics* **23** (1997), 269–311.

[18] E. F. Moore, Gedanken-Experiments on Sequential Machines. *Annals of Mathematics Studies* **34** (1956), 129–153.

[19] M. Nivat, Transductions des Langages de Chomsky. *Annales de l'Institut Fourier* **18** (1968), 339–456.

[20] Gh. Paun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms.* Springer-Verlag, Berlin, 1998.

[21] J. E. Pin, Syntactic Semigroups. In: *Handbook of Formal Languages.* Vol. 1, Springer Verlag, Berlin Heidelberg New York, 1997.

[22] G. N. Raney, Sequential Functions. *Journal of the Association for Computing Machinery* **5** (1958), 177–180.

[23] M. P. Schutzenberger, A Remark on Finite Transducers. *Information and Control* **4** (1961), 185–196.

[24] M. P. Schutzenberger, Sur une Variante des Fonctions Sequentielles. *Theoretical Computer Science* **4** (1977), 47–57.

[25] A. M. Turing, On Computable Numbers, with an Application to the Entscheidungs-Problem. *Proceedings of the London Mathematical Society*, Series 2, **42** (1936), 230–265.

[26] S. J. Walljasper, *Non-Deterministic Automata and Effective Languages.* Ph.D. Thesis, University of Iowa, 1970.

(Received: June 17, 2003; revised: August 26, 2003)