Sofia University “St. Kliment Ohridski”
Faculty of Mathematics and Informatics

Master’s thesis

Mathematics – Logic and Algorithms

Efficient representation of regular contextual replacement rules

Author
Stanimir Petrov
Faculty number: 8MI3100005

Supervisor
Prof. DSc. Stoyan Mihov

2024-07-03 02:40:51+03:00

Contents

1 Introduction

Text rewriting according to contextual replacement rules is widely used in fields like information retrieval and natural language processing. Generally, it is desired that each input string has a unique rewriting based on these rules. Different strategies are known for achieving this, one of which is leftmost-longest replacement strategy. It is well-known fact that the resulting string function is regular (see, e.g., [1]).

An interesting problem is representing this function with a finite-state device so that the rewriting can be carried out by automatic means. Since the function is regular, it can be represented by a finite-state transducer. But in general, this transducer cannot be deterministic [2, remark 5.1.11], which makes the rewriting highly inefficient. Bimachines, on the other hand, traverse its input in both directions and provide a fully deterministic processing of any regular string function in time proportional to the length of the string which is being rewritten. This advantage of them is essential and makes them useful for rewriting large texts even on devices with relatively low computing power.

One option for constructing the bimachine is to first find the regular function determined by these rules and represent it by a real-time functional finite-state transducer [1]. Then this transducer can be used to construct an equivalent bimachine [3, 4]. However, in general, the intermediate steps of finding the functional transducer may take an enormous amount of time and memory, while the final result might be smaller in size.

The constructions presented in this thesis directly produce a bimachine, avoiding the overhead costs of constructing the non-deterministic finite-state transducer first.[1] The main contribution of this work is the introduction of a new type of bimachine (called two-step bimachine) and a construction of such given a sequence (batch) of contextual rules. This type of bimachines have the same expressive power as classical bimachines but they are specially designed to provide an efficient construction process for a batch of rules. On top of that, like the classical bimachines, they provide text rewriting in linear time. For the example of our empirical analysis, classical bimachines and two-step bimachines perform the rewriting process in practically indistinguishable time.

In section 3 contextual replacement rules are introduced and the notion of leftmost-longest rewriting is formally defined. In section 4 definitions for finite-state automata and transducers are given, as well as some of their properties and constructions which are used later. In section 5 the two-step bimachines are introduced. A direct construction of such machine from a finite-state representation of batch of rules is described and its correctness is proven. In section 6 a similar direct construction is shown which constructs a classical bimachine with final output instead. Then the equivalence of the bimachines created by these two constructions is proven. In section 7 upper bounds for the size of the resulting bimachines from both constructions are given. Finally, in section 8

[1]Here we refer to the non-deterministic real-time functional finite-state transducer representing the leftmost-longest replacement, not the transducers representing the function part of each individual rule. The latter are still needed.

an implementation of both constructions is provided and they are empirically compared using a practically applicable example.

2 Formal preliminaries

Definition 2.1. *Let Q be a finite set. We say that the function $g : |Q| \to Q$ is an enumeration of Q if g is injective.*[2] *We will denote by* $\mathrm{Enum}(Q)$ *the set of all enumerations of Q, i.e.*

$$\mathrm{Enum}(Q) = \{g : |Q| \to Q \mid (\forall i < |Q|)(\forall j < |Q|)(i \neq j \implies g(i) \neq g(j))\}.$$

We will denote by $\mathrm{enumerate}(Q)$ *an arbitrary element of* $\mathrm{Enum}(Q)$.

We can think of an enumeration Q as a sequence which contains every element of Q exactly once. Clearly, for any finite set Q, $\mathrm{Enum}(Q) \neq \emptyset$, so $\mathrm{enumerate}(Q)$ is well-defined.

Proposition 2.2. *Let Q be a finite set. If $g \in \mathrm{Enum}(Q)$, then $g : |Q| \to Q$ is bijective.*

Proof. g is injective by definition. This implies $|\mathrm{Rng}(g)| = |Q|$. Since $\mathrm{Rng}(g) \subseteq Q$ and Q is finite, we have $\mathrm{Rng}(g) = Q$. □

Definition 2.3. *Let Q_1, Q_2 be finite sets such that $Q_1 \cap Q_2 = \emptyset$. Let $g_1 \in$* $\mathrm{Enum}(Q_1)$ *and $g_2 \in$* $\mathrm{Enum}(Q_2)$*. We define the concatenation of g_1 and g_2 as follows:*

$$g_1 \parallel g_2 = g_1 \cup \{(|g_1| + i, g_2(i)) \mid i \in \mathrm{Dom}(g_2)\}.$$

We can easily verify that $g_1 \parallel g_2 \in \mathrm{Enum}(Q_1 \cup Q_2)$.

Definition 2.4. *Let $R \subseteq A \times A$ be an equivalence relation. For any $a \in A$, we will denote by $[a]_R$ the equivalence class of a with respect to R, i.e.*

$$[a]_R = \{b \mid (a, b) \in R\}.$$

We will denote by A/R the set of all equivalence classes of R, i.e.

$$A/R = \{[a]_R \mid a \in A\}.$$

3 Contextual replacement rules

Definition 3.1. *Let Σ_1, Σ_2 be finite alphabets. Contextual replacement rule (CRR) over (Σ_1, Σ_2) is a triple (T, λ, ρ) where:*

- *$T \subseteq \Sigma_1^* \times \Sigma_2^*$ is a function,*

[2]When a natural number n is used in a context where a set is expected, we will assume that n is used in the sense of a von Neumann ordinal, i.e. $n = \{0, \ldots, n-1\}$.

- $\lambda \subseteq \Sigma_1^*$ *and* $\rho \subseteq \Sigma_1^*$.

Instead of (T, λ, ρ)*, we will often write* T/λ_ρ.

Definition 3.2. *Let* Σ_1, Σ_2 *be finite alphabets and* T/λ_ρ *be a CRR over* (Σ_1, Σ_2). *We say that* (u, v, w) *is a replacement context in* $t = t_1 \dots t_n \in \Sigma_1^*$ *for the rule* T/λ_ρ *if*

$$u \in \Sigma_1^* \lambda \ \ \& \ \ v \in \mathrm{Dom}(T) \ \ \& \ \ w \in \rho \Sigma_1^* \ \ \& \ \ t = uvw.$$

In this case we write $\mathrm{RC}((u, v, w), t, T/\lambda\_\rho)$. *We define the set of all replacement contexts in* t *for the rule* T/λ_ρ:

$$\mathrm{ctx}(t, T/\lambda\_\rho) = \{c \mid \mathrm{RC}(c, t, T/\lambda\_\rho)\}.$$

Application of that rule on a context (u, v, w) *in* t *is the replacement in* t *of the substring* v *(starting at position* $|u| + 1$*) with* $T(v)$.

Example 3.3. *Consider the rule* $R = (\{(a, \epsilon)\}^* \{(aa, A)\} \cup \{(ab, \epsilon)\}) / \{a\}\_\{aa, b\}$ *and the string* $t = aaaaaabaaab$. *Then*

$$\begin{aligned}\mathrm{ctx}(t, R) = \{&(\underline{a}, aa, \underline{aa}abaaab), (\underline{a}, aaa, \underline{aa}baaab), (\underline{a}, aaaaa, \underline{b}aaab),\\ &(a\underline{a}, aa, \underline{aa}baaab), (a\underline{a}, aaaa, \underline{b}aaab), (aa\underline{a}, aaa, \underline{b}aaab),\\ &(aaa\underline{a}, aa, \underline{b}aaab), (aaaa\underline{a}, ab, \underline{aa}ab), (aaaaaab\underline{a}, aa, \underline{b})\\ \}&.\end{aligned}$$

We will now extend the definition of *replacement context* to multiple CRR.

Definition 3.4. *Let* Σ_1, Σ_2 *be finite alphabets,* $m \in \mathbb{N}$. *Batch of contextual replacement rules (BCRR) over* (Σ_1, Σ_2) *is a sequence* $\mathrm{BR} = (T_r/\lambda_r\_\rho_r)_{r=1}^m$ *of CRR over* (Σ_1, Σ_2). *We define the set of all replacement contexts in* t *for the batch rule* $(T_r/\lambda_r\_\rho_r)_{r=1}^m$:

$$C(t, \mathrm{BR}) = \bigcup_{r=1}^{m} \mathrm{ctx}(t, T_r/\lambda_r\_\rho_r).$$

We also define $\mathrm{earliest}(t, \mathrm{BR})(c) = \min\{r \mid \mathrm{RC}(c, t, T_r/\lambda_r\_\rho_r)\}$.

From example 3.3, it is immediately obvious that ambiguities may occur when applying the rule R on the replacement contexts in the string t. One may choose to apply R once to get $aAaaabaaab$, or $aAaabaaab$, or $aAbaaab$, or $aaAaabaaab$, or $aaAbaaab$, or $aaaAbaaab$, or $aaaaAbaaab$, or $aaaaaaaab$, or $aaaaaabaAb$. On the other hand, R may be applied multiple times to get $aAAbaaab$, or $aaAAbaaab$, or $aAaAbaaab$, or $aAaaAb$, etc. Or R may not be applied at all to get the same string t.

Note that there are pairs of contexts in $\mathrm{ctx}(t, R)$ on which R cannot be applied simultaneously. For example

$$\begin{array}{lcl}(a, aa, aaabaaab) & \text{and} & (a, aaa, aabaaab), \ \text{or}\\ (aa, aa, aabaaab) & \text{and} & (a, aaa, aabaaab), \ \text{or}\\ (a, aaaaa, baaab) & \text{and} & (aaaaa, ab, aaab), \ \text{etc.}\end{array}$$

There are multiple strategies to resolve these ambiguities. In this work, we will focus only on the *leftmost-longest* replacement strategy. We are going to formalize this strategy in the definitions below. For other strategies, e.g. *leftmost-shortest*, *rightmost-longest*, and so on, similar ideas and notions can be used.

Let $m \in \mathbb{N}$, $\mathrm{BR} = (T_r/\lambda_r\text{-}\rho_r)_{r=1}^{m}$ be a BCRR over (Σ_1, Σ_2), $t = t_1 \dots t_n \in \Sigma_1^*$.

Definition 3.5. *Let $c_i = (u_i, v_i, w_i) \in C(t, \mathrm{BR})$ for $i \in \{1, 2\}$. We say that c_2 is over c_1 and we write $\mathrm{Over}(c_2, c_1)$ if $|u_1| < |u_2| < |u_1 v_1|$. We say that c_1 and c_2 overlap and we write $\mathrm{Overlap}(c_1, c_2)$ if $|u_1| < |u_2 v_2|$, $|u_2| < |u_1 v_1|$ and $|u_1| \neq |u_2|$.*

It is trivial to see that $\mathrm{Overlap}(c_1, c_2) \iff \mathrm{Over}(c_1, c_2) \vee \mathrm{Over}(c_2, c_1)$.

Definition 3.6. *Let $A, B \subseteq C(t, \mathrm{BR})$. We define the following functions:*

$$\begin{aligned}
\mathrm{After}(A, B) &= \{(u, v, w) \in A \mid (\forall (u', v', w') \in B)(|u| \geq |u'v'| \;\; \& \;\; |u| \neq |u'|)\},\\
\mathrm{Leftmost}(A) &= \{(u, v, w) \in A \mid (\forall (u', v', w') \in A)(|u| \leq |u'|)\},\\
\mathrm{Longest}(A) &= \{(u, v, w) \in A \mid (\forall (u', v', w') \in A)(|u| = |u'| \implies |v| \geq |v'|)\}.
\end{aligned}$$

Definition 3.7. *We define the sequence $(C_i)_{i=0}^{\infty}$ by recursion:*

$$\begin{aligned}
C_0(t, \mathrm{BR}) &= \emptyset,\\
C_{i+1}(t, \mathrm{BR}) &= C_i(t, \mathrm{BR}) \cup \mathrm{Leftmost}(\mathrm{After}(C(t, \mathrm{BR}), C_i(t, \mathrm{BR}))).
\end{aligned}$$

We now define $C_{LM}(t, \mathrm{BR})$ and $C_{LML}(t, \mathrm{BR})$ as follows:

$$\begin{aligned}
C_{LM}(t, \mathrm{BR}) &= \bigcup_{i=0}^{\infty} C_i(t, \mathrm{BR}),\\
C_{LML}(t, \mathrm{BR}) &= \mathrm{Longest}(C_{LM}(t, \mathrm{BR})).
\end{aligned}$$

From now on, we will omit the arguments t and BR when they are obvious. Note that the sequence $(C_i)_{i=0}^{\infty}$ is monotonically increasing and it becomes constant after finite number of steps. Clearly $C_{LML} \subseteq C_{LM} \subseteq C$. C is finite, so are C_{LM} and C_{LML}.

Let's take a look at our example again.

Example 3.8. *We will find $C_{LML}(t, \langle R \rangle)$ for t and R from example 3.3.*

$$\begin{aligned}
\mathrm{After}(C, C_0) &= \mathrm{After}(C, \emptyset) = C,\\
\mathrm{Leftmost}(\mathrm{After}(C, C_0)) &= \{(a, aa, aaabaaab), (a, aaa, aabaaab), (a, aaaaa, baaab)\},\\
C_1 &= C_0 \cup \mathrm{Leftmost}(\mathrm{After}(C, C_0))\\
&= \{(a, aa, aaabaaab), (a, aaa, aabaaab), (a, aaaaa, baaab)\},\\
\mathrm{After}(C, C_1) &= \{(aaaaaaba, aa, b)\},\\
\mathrm{Leftmost}(\mathrm{After}(C, C_1)) &= \{(aaaaaaba, aa, b)\},\\
C_2 &= C_1 \cup \mathrm{Leftmost}(\mathrm{After}(C, C_1))
\end{aligned}$$

$$= \{(a, aa, aaabaaab), (a, aaa, aabaaab), (a, aaaaa, baaab), (aaaaaaba, aa, b)\},$$

$$\mathrm{After}(C, C_2) = \emptyset,$$

$$\mathrm{Leftmost}(\mathrm{After}(C, C_1)) = \emptyset,$$

$$C_3 = C_2 \cup \mathrm{Leftmost}(\mathrm{After}(C, C_1)) = C_2.$$

Since $C_3 = C_2$, it follows that $C_i = C_2$ for all $i \geq 3$. Therefore

$$C_{LM} = \bigcup_{i=0}^{\infty} C_i = C_2,$$

$$C_{LML}(t, \langle R \rangle) = \mathrm{Longest}(C_{LM}) = \{(a, aaaaa, baaab), (aaaaaaba, aa, b)\}.$$

Thus the rewriting of $t = aaaaaabaaab$ according to $\langle R \rangle$ using leftmost-longest match strategy is $aAbaAb$. This is precisely defined in definition 3.12.

Proposition 3.9. *There are no overlapping contexts in C_{LM}.*

Proof. Let $c_i = (u_i, v_i, w_i) \in C_{LM}$, $k_i = \min\{k \mid c_i \in C_k\}$ for $i \in \{1, 2\}$. Note that $k_i > 0$ since $C_0 = \emptyset$.

case 1: $k_1 = k_2$. Let $A = \mathrm{After}(C, C_{k_1 - 1})$. We have $c_1, c_2 \in C_{k_1} \setminus C_{k_1 - 1}$, so $c_1, c_2 \in \mathrm{Leftmost}(A)$ and hence $c_1, c_2 \in A$. Since $c_1 \in \mathrm{Leftmost}(A)$, we get that $(\forall (u', v', w') \in A)(|u_1| \leq |u'|)$. In particular, since $c_2 \in A$, we have $|u_1| \leq |u_2|$.

By similar reasoning, we get that $|u_2| \leq |u_1|$.

Therefore $|u_1| = |u_2|$, so $\neg\mathrm{Overlap}(c_1, c_2)$.

case 2: $k_1 < k_2$. Let $A = \mathrm{After}(C, C_{k_2 - 1})$. We have $c_2 \in C_{k_2} \setminus C_{k_2 - 1}$, so $c_2 \in \mathrm{Leftmost}(A)$ and hence $c_2 \in A$. Since $c_2 \in A$, we get that $(\forall (u', v', w') \in C_{k_2 - 1})(|u_2| \geq |u'v'| \ \&\ |u_2| \neq |u'|)$. In particular, $c_1 \in C_{k_1} \subseteq C_{k_2 - 1}$ (since $k_1 \leq k_2 - 1$). Therefore $|u_2| \geq |u_1 v_1|$, so $\neg\mathrm{Overlap}(c_1, c_2)$.

case 3: $k_1 > k_2$. Same reasoning as in the previous case with c_1 and c_2 swapped.

□

It follows immediately that there are no overlapping contexts in C_{LML} either. Moreover, by the definition of C_{LML},

$$(\forall (u_1, v_1, w_1), (u_2, v_2, w_2) \in C_{LML})(|u_1| = |u_2| \implies |v_1| = |v_2|).$$

Proposition 3.10. *Let $c_1 = (u, v_1, w_1) \in C$ and $c_2 = (u, v_2, w_2) \in C$. Then*

$$c_1 \in C_{LM} \iff c_2 \in C_{LM}.$$

Proof. Let $c_1 \in C_{LM}$. Let $k_1 = \min\{k \mid c_1 \in C_k\}$. Let $A = \mathrm{After}(C, C_{k_1 - 1})$. Then $c_1 \in \mathrm{Leftmost}(A)$ and hence $c_1 \in A$. Since $c_1 \in A$, we get that $(\forall (u', v', w') \in C_{k_2 - 1})(|u| \geq |u'v'| \ \&\ |u| \neq |u'|)$. But this also means that $c_2 \in A$. Now, since $c_1 \in \mathrm{Leftmost}(A)$, we get that $(\forall (u', v', w') \in A)(|u| \leq |u'|)$. But again this also means that $c_2 \in \mathrm{Leftmost}(A)$. Therefore $c_2 \in C_{k_1} \subseteq C_{LM}$.

The other direction is completely symmetrical. □

Proposition 3.11. *Let $c_1 = (u_1, v_1, w_1) \in C \setminus C_{LM}$. Then*

$$(\exists c_2 \in C_{LML})\, \mathrm{Over}(c_1, c_2).$$

Proof. Let $A_i = \mathrm{After}(C, C_i)$. We can easily see that the sequence $(A_i)_{i=0}^{\infty}$ is monotonically decreasing with respect to set-theoretical inclusion. Moreover, if $A_i \neq \emptyset$, then $A_i \supset A_{i+1}$. Therefore there is a unique $j \in \mathbb{N}$ such that $c_1 \in A_j \setminus A_{j+1}$. Because $c_1 \notin A_{j+1}$, we have that

$$(\exists (u', v', w') \in C_{j+1})(|u_1| < |u'v'| \vee |u_1| = |u'|).$$

Take a witness $c_2' = (u_2', v_2', w_2')$ for this existence. By proposition 3.10, we can immediately conclude that $|u_1| \neq |u_2'|$ ($c_1 \notin C_{LM}$, but $c_2' \in C_{j+1} \subseteq C_{LM}$). Therefore it must be the case that $|u_1| < |u_2'v_2'|$.

On the other hand, since $c_1 \in A_j$, we have that

$$(\forall (u', v', w') \in C_j)(|u_1| \geq |u'v'| \ \ \& \ \ |u_1| \neq |u'|).$$

Hence $c_2' \notin C_j$. From the definition of C_{j+1}, it follows that $c_2' \in \mathrm{Leftmost}(A_j)$ which means that $(\forall (u', v', w') \in A)(|u_2'| \leq |u'|)$. In particular, since $c_1 \in A_j$, we get $|u_2'| \leq |u_1|$. But we already know that $|u_1| \neq |u_2'|$, so $|u_2'| < |u_1|$.

We now have that $|u_2'| < |u_1| < |u_2'v_2'|$, hence $\mathrm{Over}(c_1, c_2')$. Since $c_2' \in C_{LM}$, there is some $c_2 \in C_{LML}$ such that $c_2 = (u_2', v_2, w_2)$ and $|v_2| \geq |v_2'|$. Therefore $\mathrm{Over}(c_1, c_2)$. □

Definition 3.12. *We say that $t' = x_0 y_0 x_1 y_1 \ldots x_n y_n$ is a rewriting of t according to the batch of rules* BR *using leftmost-longest match strategy where:*

- $x_0 = \epsilon$,
- $$x_i = \begin{cases} T_{\mathrm{earliest}((u,v,w))}(v) & (\exists (u,v,w) \in C_{LML})(|u| + 1 = i \ \& \ i \leq |uv|) \\ \epsilon & (\exists (u,v,w) \in C_{LML})(|u| + 1 < i \ \& \ i \leq |uv|) \\ t_i & \textit{otherwise} \end{cases}$$
  for each $0 < i \leq n$,
- $$y_i = \begin{cases} T_{\mathrm{earliest}((u,v,w))}(\epsilon) & (\exists (u,v,w) \in C_{LML})(|u| = i \ \& \ v = \epsilon) \\ \epsilon & \textit{otherwise} \end{cases}$$
  for each $0 \leq i \leq n$.

We will denote this rewriting t' by $\mathrm{Rewrite}_{LML}(\mathrm{BR}, t)$.

If $c \in C_{LML}$, then $\{r \mid RC(c, t, T_r/\lambda_r\text{-}\rho_r)\} \neq \emptyset$. From this and the fact that there are no overlapping contexts in C_{LML}, it follows that $\mathrm{Rewrite}_{LML}(\mathrm{BR}, t)$ is correctly defined.

4 Finite-state automata and transducers

Definition 4.1. *A monoid is a triple $\mathcal{M} = (M, \circ, e)$ where:*

- $M \neq \emptyset$ *is a set,*
- $\circ : M \times M \to M$ *is the monoid operation and has the property*

$$(\forall a \in M)(\forall b \in M)(\forall c \in M)((a \circ b) \circ c = a \circ (b \circ c)),$$

- $e \in M$ *is the unit element and has the property*

$$(\forall a \in M)(a \circ e = a \ \& \ e \circ a = a).$$

Definition 4.2. *A monoidal finite-state automaton (monoidal FSA) is a tuple $A = (\mathcal{M}, Q, S, F, \Delta)$ where:*

- $\mathcal{M} = (M, \circ, e)$ *is a monoid,*
- Q *is a finite set of states,*
- $S \subseteq Q$ *is the set of initial states,*
- $F \subseteq Q$ *is the set of final states,*
- $\Delta \subseteq Q \times M \times Q$ *is a finite set of transitions.*

For $m \in \mathbb{N}$, we define $\Delta^{(n)}$ by recursion:

$$\begin{aligned}\Delta^{(0)} &= \{(q, e, q) \mid q \in Q\},\\ \Delta^{(n+1)} &= \{(q_1, a \circ \alpha, q_3) \mid \exists q_2((q_1, a, q_2) \in \Delta \ \& \ (q_2, \alpha, q_3) \in \Delta^{(n)})\}.\end{aligned}$$

We define the generalized transition relation Δ^* as follows:

$$\Delta^* = \bigcup_{n=0}^{\infty} \Delta^{(n)}.$$

The language accepted by A is $\mathrm{L}(A) = \{w \mid (\exists s \in I)(\exists f \in F)(s, w, f) \in \Delta^*\}$. The left language of a state $q \in Q$ is $\overleftarrow{\mathrm{L}_A(q)} = \{w \mid (\exists s \in S)(s, w, q) \in \Delta^*\}$. The right language of a state $q \in Q$ is $\overrightarrow{\mathrm{L}_A(q)} = \{w \mid (\exists f \in F)(q, w, f) \in \Delta^*\}$. We say that $(q_i)_{i=0}^n$ is an execution of A over $a_1 \circ \cdots \circ a_n$ if $q_0 \in S$ and, for each $i < n$, $(q_i, a_{i+1}, q_{i+1}) \in \Delta$. We say that the execution $(q_i)_{i=0}^n$ is successful if $q_n \in F$. We say that A is e-free if $\Delta \subseteq Q \times (M \setminus \{e\}) \times Q$. We denote the set of a-successors of $P \subseteq Q$ by $\mathrm{Succ}_A(P, a) = \{q' \mid (\exists q \in P)(q, a, q') \in \Delta\}$.

Proposition 4.3. *For every monoidal FSA $A = ((M, \circ, e), Q, S, F, \Delta)$ there is an e-free monoidal FSA A' equivalent to A, i.e. $\mathrm{L}(A') = \mathrm{L}(A)$.*

Proof. See [2, proposition 2.5.4]. $\square$

Definition 4.4. *Let $A = ((M, \circ, e), Q, S, F, \Delta)$ be an e-free monoidal FSA. We say that A is left-simple if $S = \{s\}$ and $\Delta \cap (Q \times M \times S) = \emptyset$, i.e. A has only one initial state and there are no transitions going into it. We say that A is right-simple if $F = \{f\}$ and $\Delta \cap (F \times M \times Q) = \emptyset$, i.e. A has only one final state and there are no transitions coming out of it. We say that A is simple if it is left-simple and right-simple.*

Proposition 4.5. *Let $A = (\mathcal{M}, Q, S, F, \Delta)$ be an e-free monoidal FSA. Then there exist a left-simple monoidal FSA A_{ls} and a right-simple monoidal FSA A_{rs}, such that $\mathrm{L}(A) = \mathrm{L}(A_{ls}) = \mathrm{L}(A_{rs})$.*

Proof. Let $s, f \notin Q$.

$$A_{ls} = (\mathcal{M}, Q \cup \{s\}, \{s\}, F_{ls}, \Delta_{ls}), \text{ where}$$

- $F_{ls} = F \cup \begin{cases} \emptyset & S \cap F = \emptyset \\ \{s\} & S \cap F \neq \emptyset \end{cases}$,
- $\Delta_{ls} = \Delta \cup \{(s, a, q) \mid (\exists s' \in S)(s', a, q) \in \Delta\}$.

$$A_{rs} = (\mathcal{M}, Q \cup \{f\}, S_{rs}, \{f\}, \Delta_{rs}), \text{ where}$$

- $S_{rs} = S \cup \begin{cases} \emptyset & S \cap F = \emptyset \\ \{f\} & S \cap F \neq \emptyset \end{cases}$,
- $\Delta_{rs} = \Delta \cup \{(q, a, f) \mid (\exists f' \in F)(q, a, f') \in \Delta\}$.

Clearly, A_{ls} is left-simple, A_{rs} is right-simple and $\mathrm{L}(A) = \mathrm{L}(A_{ls}) = \mathrm{L}(A_{rs})$. □

Definition 4.6. *We say that a monoidal finite-state automaton $A = ((\Sigma^*, \cdot, \epsilon), Q, S, F, \Delta)$ is classical if:*

- *Σ is a finite set called alphabet,*
- *$(\Sigma^*, \cdot, \epsilon)$ is the free monoid over Σ,*
- *$\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$.*

For simplicity, we will write $A = (\Sigma, Q, S, F, \Delta)$. When we say that A is a nondeterministic finite-state automaton (NFA) or simply a finite-state automaton (FSA), we mean that A is classical. The reversed FSA of A is $A^{\mathrm{rev}} = (\Sigma, Q, F, S, \Delta^{\mathrm{rev}})$, where $\Delta^{\mathrm{rev}} = \{(p, a, q) \mid (q, a, p) \in \Delta\}$.

Definition 4.7. *We say that a nondeterministic finite-state automaton $A = (\Sigma, Q, \{q_0\}, F, \Delta)$ is deterministic (DFA) if:*

- *$\Delta \subseteq Q \times \Sigma \times Q$,*
- *$(\forall q, q', q'' \in Q)(\forall a \in \Sigma)((q, a, q') \in \Delta \;\&\; (q, a, q'') \in \Delta \implies q' = q'')$.*

For simplicity, we will write $A = (\Sigma, Q, q_0, F, \delta)$, where $\delta : Q \times \Sigma \nrightarrow Q$ is a partial function called transition function and $\delta(q, a) = q' \iff (q, a, q') \in \Delta$.

Definition 4.8. *We say that a monoidal finite-state automaton $A = (\mathcal{M}, Q, S, F, \Delta)$ is a classical finite-state transducer or simply a finite-state transducer (FST) if $\mathcal{M}$ can be represented as a Cartesian product of free monoids over some finite sets Σ_1 and Σ_2. For simplicity we will write $A = (\Sigma_1, \Sigma_2, Q, S, F, \Delta)$. We say that A is letter transducer if $\Delta \subseteq Q \times ((\Sigma_1 \cup \{\epsilon\}) \times \Sigma_2^*) \times Q$. We say that A is real-time transducer if $\Delta \subseteq Q \times (\Sigma_1 \times \Sigma_2^*) \times Q$. The underlying automaton of A is $A^{\text{dom}} = (\Sigma_1, Q, S, F, \Delta^{\Sigma_1})$, where $\Delta^{\Sigma_1} = \{(q, a, q') \mid (\exists m \in \Sigma_2^*)(q, (a, m), q') \in \Delta\}$. We say that A is functional if $\mathrm{L}(A)$ is a function.*

Proposition 4.9. *Let $A = (\Sigma_1, \Sigma_2, Q, S, F, \Delta)$ be a real-time finite-state transducer. Then there exists a simple FST A_s such that $\mathrm{L}(A_s) = \mathrm{L}(A) \setminus \{(\epsilon, \epsilon)\}$ and A_s is also real-time.*

Proof. Let $s, f \notin Q$ and $s \neq f$.

$$A_s = (\Sigma_1, \Sigma_2, Q \cup \{s, f\}, \{s\}, \{f\}, \Delta_s), \text{ where}$$

$$\begin{aligned} \Delta_s = \Delta \quad & \cup \quad \{(s, a, q) \mid (\exists s' \in S)(s', a, q) \in \Delta\} \\ & \cup \quad \{(q, a, f) \mid (\exists f' \in F)(q, a, f') \in \Delta\} \\ & \cup \quad \{(s, a, f) \mid (\exists s' \in S)(\exists f' \in F)(s', a, f') \in \Delta\}. \end{aligned}$$

Clearly, A_s is simple, real-time and $\mathrm{L}(A_s) = \mathrm{L}(A) \setminus \{(\epsilon, \epsilon)\}$. □

Proposition 4.10. *For every functional FST $A = (\Sigma_1, \Sigma_2, Q, S, F, \Delta)$ there is a real-time FST A' such that $\mathrm{L}(A') \setminus (\{\epsilon\} \times \Sigma_2^*) = \mathrm{L}(A) \setminus (\{\epsilon\} \times \Sigma_2^*)$.*

Proof. See [2, proposition 4.4.8]. □

Definition 4.11. *We say that T is regular function if $T = \mathrm{L}(\mathcal{T})$ for some FST $\mathcal{T}$.*[3] *We say that a CRR T/λ_ρ is regular if all T, λ and ρ are regular. We say that a BCCR $(T_r/\lambda_r\_\rho_r)_{r=1}^m$ is regular if all of its members are regular.*

From now on, we will assume that all of the contextual replacement rules and batches of contextual replacement rules we use are regular.

Definition 4.12. *Let $(T_r/\lambda_r\_\rho_r)_{r=1}^m$ be a BCRR over (Σ_1, Σ_2). We say that $((\mathcal{T}_r, \mathcal{E}_r, A_{\lambda_r}, A_{\rho_r}))_{r=1}^m$ is a finite-state representation (FSR) of $(T_r/\lambda_r\_\rho_r)_{r=1}^m$ if for each r, r' such that $1 \leq r \leq m$, $1 \leq r' \leq m$:*

- *$A_{\lambda_r} = (\Sigma_1, Q_{\lambda_r}, S_{\lambda_r}, \{f_{\lambda_r}\}, \Delta_{\lambda_r})$ is a right-simple NFA such that $\mathrm{L}(A_{\lambda_r}) = \Sigma_1^* \cdot \lambda_r$,*
- *$A_{\rho_r} = (\Sigma_1, Q_{\rho_r}, \{s_{\rho_r}\}, F_{\rho_r}, \Delta_{\rho_r})$ is a left-simple NFA such that $\mathrm{L}(A_{\rho_r}) = \rho_r \cdot \Sigma_1^*$,*
- *$\mathcal{T}_r = (\Sigma_1, \Sigma_2, Q_{T_r}, \{s_{T_r}\}, \{f_{T_r}\}, \Delta_{T_r})$ is a simple real-time transducer such that $\mathrm{L}(\mathcal{T}_r) = T_r \setminus (\{\epsilon\} \times \Sigma_2^*)$,*

[3] This is a simplified definition which tacitly uses Kleene's theorem. For a proper definition of monoidal regular language see [2, definition 2.3.1]

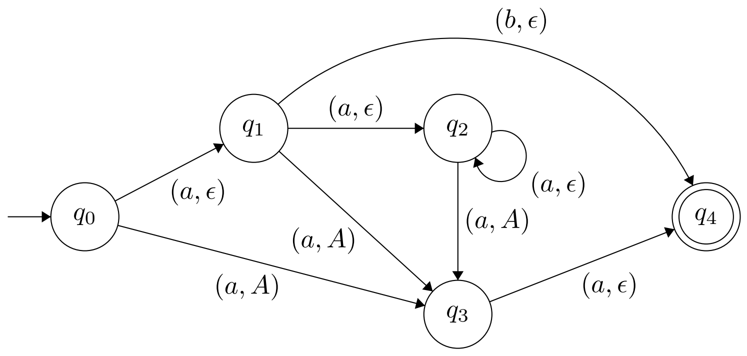


Figure 4.1: A *simple* real-time transducer representing the regular function $\{(a,\epsilon)\}^*\{(aa,A)\}\cup\{(ab,\epsilon)\}$.

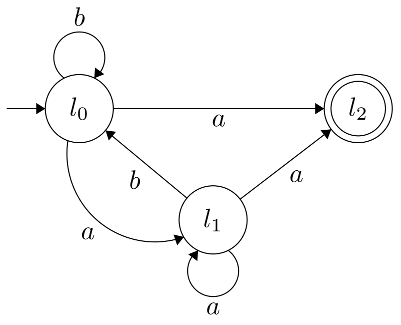


Figure 4.2: A *right-simple* NFA representing the language $\{a,b\}^*\cdot\{a\}$.

- $\mathcal{E}_r = \begin{cases} T_r(\epsilon) & \epsilon \in \text{Dom}(T_r) \\ \uparrow^{4} & \textit{otherwise} \end{cases},$
- *If* $r \neq r'$, *then* $Q_{\lambda_r} \cap Q_{\lambda_{r'}} = \emptyset$, $Q_{\rho_r} \cap Q_{\rho_{r'}} = \emptyset$ *and* $Q_{T_r} \cap Q_{T_{r'}} = \emptyset$,
- $Q_{T_r} \cap Q_{\rho_{r'}} = \emptyset$.

Example 4.13. *Consider the rule* $R = (\{(a,\epsilon)\}^*\{(aa,A)\}\cup\{(ab,\epsilon)\})/\{a\}\_\{aa,b\}$ *from example 3.3 and assume* $\Sigma_1 = \{a,b\}$. *A possible FSR of* $\langle R\rangle$ *is* $\langle(\mathcal{T}_1,\mathcal{E}_1,A_{\lambda_1},A_{\rho_1})\rangle$ *where* $\mathcal{E}_1$ *is undefined and* $\mathcal{T}_1, A_{\lambda_1}, A_{\rho_1}$ *are as shown in figure 4.1, figure 4.2 and figure 4.3 respectively.*

Construction 4.14. *Let, for each* $1 \leq r \leq m$, $A_r = (\Sigma, Q_r, S_r, F_r, \Delta_r)$ *be a NFA such that for* $i \neq j$, $Q_i \cap Q_j = \emptyset$. *Let* A *be constructed as follows:*

$$A \coloneqq \left(\Sigma, \bigcup_{r=1}^{m} Q_r, \bigcup_{r=1}^{m} S_r, \bigcup_{r=1}^{m} F_r, \bigcup_{r=1}^{m} \Delta_r\right).$$

[4]We will use $\uparrow$ to indicate that something is undefined and $\downarrow$ to indicate that it is defined.

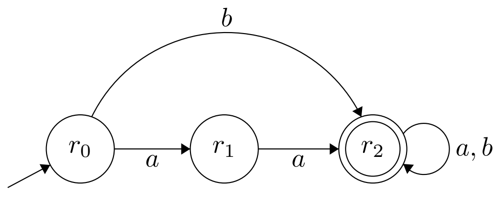


Figure 4.3: A *left-simple* NFA representing the language $\{aa, b\} \cdot \{a, b\}^*$.

Proposition 4.15. *Let, for each* $1 \leq r \leq m$, $A_r = (\Sigma, Q_r, S_r, F_r, \Delta_r)$ *be a NFA such that for* $i \neq j$, $Q_i \cap Q_j = \emptyset$. *Let* A *be obtained from construction 4.14 applied on the sequence* $(A_r)_{r=1}^m$. *Then* $\mathrm{L}(A) = \bigcup_{r=1}^m \mathrm{L}(A_r)$.

Construction 4.16. *Let* $A_1 = (\Sigma, Q_1, S_1, \{f_1\}, \Delta_1)$ *and* $A_2 = (\Sigma, Q_2, \{s_2\}, F_2, \Delta_2)$ *be NFA such that* $Q_1 \cap Q_2 = \emptyset$. *An NFA* $A \coloneqq (\Sigma, Q_1 \cup Q_2, S_1, F, \Delta)$ *is constructed where:*

- $F \coloneqq F_2 \cup \begin{cases} \emptyset & s_2 \notin F_2 \\ \{f_1\} & s_2 \in F_2 \end{cases}$,
- $\Delta \coloneqq \Delta_1 \cup \Delta_2 \cup \{(f_1, a, q) \mid (s_2, a, q) \in \Delta_2\}$.

Proposition 4.17. *Let* $A_1 = (\Sigma, Q_1, S_1, \{f_1\}, \Delta_1)$ *and* $A_2 = (\Sigma, Q_2, \{s_2\}, F_2, \Delta_2)$ *be NFA such that* $Q_1 \cap Q_2 = \emptyset$. *Let* A *be obtained from construction 4.16 applied on* A_1 *and* A_2. *Then* $\mathrm{L}(A) = \mathrm{L}(A_1) \cdot \mathrm{L}(A_2)$.

5 Two-step bimachines

Definition 5.1. *We say that* $A = (\Sigma_1, \Sigma_2, A_L, A_R, Q, q_0, \delta, \tau, \psi_\delta, \psi_\tau)$ *is a two-step bimachine where:*

- Σ_1 *is a finite set called input alphabet,*
- Σ_2 *is a finite set called output alphabet,*
- Q *is a finite set of states,*
- $A_L = (\Sigma_1, Q_L, s_L, Q_L, \delta_L)$ *is a DFA - the left automaton,*
- $A_R = (\Sigma_1, Q_R, s_R, Q_R, \delta_R)$ *is a DFA - the right automaton,*
- $q_0 \in Q$ *is the initial state,*
- $\delta : Q \times \Sigma_1 \times Q_R \nrightarrow Q$,
- $\tau : Q \times Q_L \times Q_R \nrightarrow Q$,
- $\psi_\delta : Q \times \Sigma_1 \times Q_R \nrightarrow \Sigma_1 \cup \Sigma_2^*$,

- $\psi_\tau : Q \times Q_L \times Q_R \nrightarrow \Sigma_2^*$.

Definition 5.2. *Let $A = (\Sigma_1, \Sigma_2, A_L, A_R, Q, q_0, \delta, \tau, \psi_\delta, \psi_\tau)$ be a two-step bimachine. Let $t = t_1 \dots t_n \in \Sigma_1^*$. Let $L_0, \dots, L_n$ be the execution of A_L over t and $R_n, \dots, R_0$ be the execution of A_R over* $\mathrm{rev}(t)$. *Execution of A over t is the sequence $((L_i, q_i, \overline{q_i}, R_i, o_i, \overline{o_i}))_{i=0}^n$ where:*

- $\overline{q_i} = \tau(q_i, L_i, R_i), \quad 0 \le i \le n,$
- $q_i = \delta(\overline{q_{i-1}}, t_i, R_i), \quad 0 < i \le n,$
- $o_0 = \epsilon,$
- $\overline{o_i} = \psi_\tau(q_i, L_i, R_i), \quad 0 \le i \le n,$
- $o_i = \psi_\delta(\overline{q_{i-1}}, t_i, R_i), \quad 0 < i \le n.$

Output of A for t is $\mathrm{Out}_A(t) = o_0\overline{o_0} \dots o_n\overline{o_n}$.

Construction 5.3. *Let $(T_r/\lambda_r\text{-}\rho_r)_{r=1}^m$ be a BCRR over (Σ_1, Σ_2). Let $((\mathcal{T}_r, \mathcal{E}_r, A_{\lambda_r}, A_{\rho_r}))_{r=1}^m$ be a FSR of $(T_r/\lambda_r\text{-}\rho_r)_{r=1}^m$ (the input for the construction).*

Let $A_\lambda = (\Sigma_1, Q_\lambda, S_\lambda, F_\lambda, \Delta_\lambda)$ be obtained from construction 4.14 applied on $(A_{\lambda_r})_{r=1}^m$. The left automaton $A_L = (\Sigma_1, Q_L, s_L, Q_L, \delta_L)$ is obtained by applying a determinization procedure on A_λ, i.e.

- $Q_L \subseteq \mathcal{P}(Q_\lambda)$,
- $s_L \coloneqq S_\lambda$,
- $\delta_L(L, a) \coloneqq \mathrm{Succ}_{A_\lambda}(L, a)$, *for $L \in Q_L$ and $a \in \Sigma_1$.*

Let $\mathcal{T}_r^{\mathrm{dom}} = (\Sigma_1, Q_{T_r}, \{s_{T_r}\}, \{f_{T_r}\}, \Delta_{T_r}^{\Sigma_1})$ be the underlying automaton of $\mathcal{T}_r$. Let $\mathcal{T}^{\mathrm{dom}} = (\Sigma_1, Q_T, S_T, F_T, \Delta_T^{\Sigma_1})$ be obtained from construction 4.14 applied on $(\mathcal{T}_r^{\mathrm{dom}})_{r=1}^m$. Let $\Delta_T = \bigcup_{r=1}^m \Delta_{T_r}$. For each $1 \le r \le m$, let $A_{R_r} = (\Sigma_1, Q_{R_r}, S_{R_r}, F_{R_r}, \Delta_{R_r})$ be obtained from construction 4.16 applied on $\mathcal{T}_r^{\mathrm{dom}}$ and A_{ρ_r}.

Now, let $A_{RU} = (\Sigma_1, Q_{RU}, S_{RU}, F_{RU}, \Delta_{RU})$ be obtained from construction 4.14 applied on $(A_{R_r}^{\mathrm{rev}})_{r=1}^m$. Let $\mathrm{type} = \{(q, r) \mid q \in Q_{T_r}\}$. *Since $Q_{T_r} \cap Q_{T_{r'}} = \emptyset$ for $r \ne r'$,* $\mathrm{type} : Q_T \to \{1, \dots, m\}$ *is a function. For $q \in Q_T$, we will say that* $\mathrm{type}(q)$ *is the type of q. The right automaton is $A_R = (\Sigma_1, Q_R, s_R, Q_R, \delta_R)$ where:*

- $Q_R \subseteq \mathcal{P}(Q_{RU}) \times \bigcup_{Q \subseteq Q_T} \mathrm{Enum}(Q)$,
- $s_R \coloneqq (S_{RU}, g_s)$, *where*
  - $g_s \in \mathrm{Enum}(S_{RU} \cap F_T)$,
  - $(\forall i \in \mathrm{Dom}(g_s))(\forall j \in \mathrm{Dom}(g_s))(i \le j \implies \mathrm{type}(g_s(i)) \le \mathrm{type}(g_s(j)),$
- $\delta_R((R', g'), a) \coloneqq (R, g)$, *for $(R', g') \in Q_R$ and $a \in \Sigma_1$ where (R, g) is an arbitrary element with the following properties:*

  - $R = \mathrm{Succ}_{A_{RU}}(R', a)$,
  - $g \in \mathrm{Enum}(R \cap Q_T)$,
  - $(\forall i \in \mathrm{Dom}(g))(\forall j \in \mathrm{Dom}(g))(i \leq j \implies ((g(i) \in F_T \;\&\; g(j) \in F_T \;\&\; \mathrm{type}(g(i)) \leq \mathrm{type}(g(j))) \;\vee\; (g(i) \notin F_T \;\&\; g(j) \in F_T) \;\vee\; \mu(g(i), a, g') \leq \mu(g(j), a, g')))$, *where* $\mu(q, a, g') = \min\{i' \mid (q, a, g'(i')) \in \Delta_T^{\Sigma_1}\}$.

Let $L \in Q_L$, $(R, g) \in Q_R$. We will define the following functions:

$$\begin{aligned} I(L,g) &:= \{i \mid g(i) \in S_T \;\&\; f_{\lambda_{\mathrm{type}(g(i))}} \in L\}, \\ \nu(L,g) &:= g(\min(I(L,g))), \\ J(L,R) &:= \{r \mid \mathcal{E}_r\downarrow \;\&\; f_{\lambda_r} \in L \;\&\; f_{T_r} \in R\}. \end{aligned}$$

The output of this construction is a two-step bimachine $A = (\Sigma_1, \Sigma_2, A_L, A_R, Q, q_{\mathrm{err}}, \delta, \tau, \psi_\delta, \psi_\tau)$ where:

- $Q = Q_T \cup \{q_{\mathrm{err}}\}, q_{\mathrm{err}} \notin Q_T$,
- $\delta(\overline{q}, a, (R', g')) := \begin{cases} g'(\mu(\overline{q}, a, g')) & \overline{q} \in \mathrm{Rng}(g) \setminus F_T, \text{ where } (R, g) = \delta_R((R', g'), a) \\ q_{\mathrm{err}} & \text{otherwise} \end{cases}$,
- $\tau(q, L, (R, g)) := \begin{cases} q & q \in Q_T \setminus F_T \\ \nu(L, g) & q \notin Q_T \setminus F_T \;\&\; I(L,g) \neq \emptyset \\ q_{\mathrm{err}} & \text{otherwise} \end{cases}$,
- $\psi_\delta(\overline{q}, a, (R', g')) := \begin{cases} \mathrm{elem\_of}\{m \mid (\overline{q}, (a, m), \delta(\overline{q}, a, (R', g'))) \in \Delta_T\} & \delta(\overline{q}, a, (R', g')) \in Q_T \\ a & \text{otherwise} \end{cases}$,
- $\psi_\tau(q, L, (R, g)) := \begin{cases} \mathcal{E}_{\min(J(L,R))} & \tau(q, L, (R, g)) = q_{\mathrm{err}} \;\&\; J(L,R) \neq \emptyset \\ \epsilon & \text{otherwise} \end{cases}$.

Example 5.4. *A possible result of applying construction 5.3 on the FSR $\langle(\mathcal{T}_1, \mathcal{E}_1, A_{\lambda_1}, A_{\rho_1})\rangle$ from example 4.13 is $\mathcal{A} = (\{a, b\}, \{A\}, A_L, A_R, Q, q_{\mathrm{err}}, \delta, \tau, \psi_\delta, \psi_\tau)$ where*

- $A_L = (\{a, b\}, Q_L, L_0, Q_L, \delta_L)$, *where*
  - $Q_L = \{L_0, L_1\}$,
  - $L_0 = \{l_0\}$ $\quad L_1 = \{l_1, l_2\}$,
  - *For δ_L see figure 5.1.*
- $A_R = (\{a, b\}, Q_R, R_0, Q_R, \delta_R)$, *where*
  - $Q_R = \{R_0, R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8, R_9, R_{10}\}$,

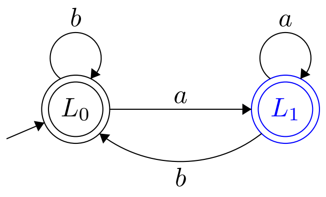


Figure 5.1: The left automaton of the two-step bimachine from example 5.4. It lands on state L_1 (which contains the final state l_2 of A_{λ_1}) iff the last symbol read was $'a'$. This is used to determine whether a valid left context is present after reading some prefix of the input string.

–
$$\begin{array}{llll}
R_0 = (\{r_2\}, & \langle\rangle) & R_6 = (\{r_0, r_1, r_2, q_3, q_4\}, & \langle q_3, q_4\rangle) \\
R_1 = (\{r_1, r_2\}, & \langle\rangle) & R_7 = (\{r_0, r_1, r_2, q_0, q_1, q_2, q_4\}, & \langle q_1, q_2, q_0, q_4\rangle) \\
R_2 = (\{r_0, r_2, q_4\}, & \langle q_4\rangle) & R_8 = (\{r_1, r_2, q_0, q_3\}, & \langle q_0, q_3\rangle) \\
R_3 = (\{r_0, r_1, r_2, q_4\}, & \langle q_4\rangle) & R_9 = (\{r_0, r_1, r_2, q_0, q_1, q_2, q_3, q_4\}, & \langle q_1, q_2, q_0, q_3, q_4\rangle) \\
R_4 = (\{r_1, r_2, q_3\}, & \langle q_3\rangle) & R_{10} = (\{r_0, r_1, r_2, q_0, q_1, q_2, q_3, q_4\}, & \langle q_0, q_1, q_2, q_3, q_4\rangle) \\
R_5 = (\{r_0, r_2, q_4\}, & \langle q_4\rangle) & &
\end{array},$$

– *For δ_R see figure 5.2.*

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{err}}\}$,
- *For δ and ψ_δ see table 5.1, for τ and ψ_τ – table 5.2.*

$\overline{q}$	x	R	$\delta(\overline{q}, x, R)$	$\psi_\delta(\overline{q}, x, R)$
q_0, q_1, q_2	a	R_4, R_6, R_8	q_3	A
q_0	a	R_5, R_7, R_9, R_{10}	q_1	ϵ
q_1, q_2	a	R_7, R_9, R_{10}	q_2	ϵ
q_1	b	R_2, R_3, R_5, R_6, R_7, R_9, R_{10}	q_4	ϵ
q_3	a	R_2, R_3, R_5, R_6, R_7, R_9, R_{10}	q_4	ϵ

Table 5.1: The functions δ and ψ_δ of $\mathcal{A}$ from example 5.4. Unless otherwise specified, $\delta(\overline{q}, x, R) = q_{\text{err}}$ and $\psi_\delta(\overline{q}, x, R) = x$.

The execution of $\mathcal{A}$ over the string $t = aaaaaabaaab$ is shown on figure 5.3.

Proposition 5.5. *Construction 5.3 correctly defines a two-step bimachine A.*

Proof. Clearly, τ and ψ_τ are correctly defined. For g_s, we just need to sort the states in $S_{RU} \cap F_T$ by their type.

Let $(R', g') \in Q_R$, $a \in \Sigma_1$ and $R = \text{Succ}_{A_{RU}}(R', a)$. We are going to show that there exists $g \in \text{Enum}(R \cap Q_T)$ such that (R, g) can be a value of

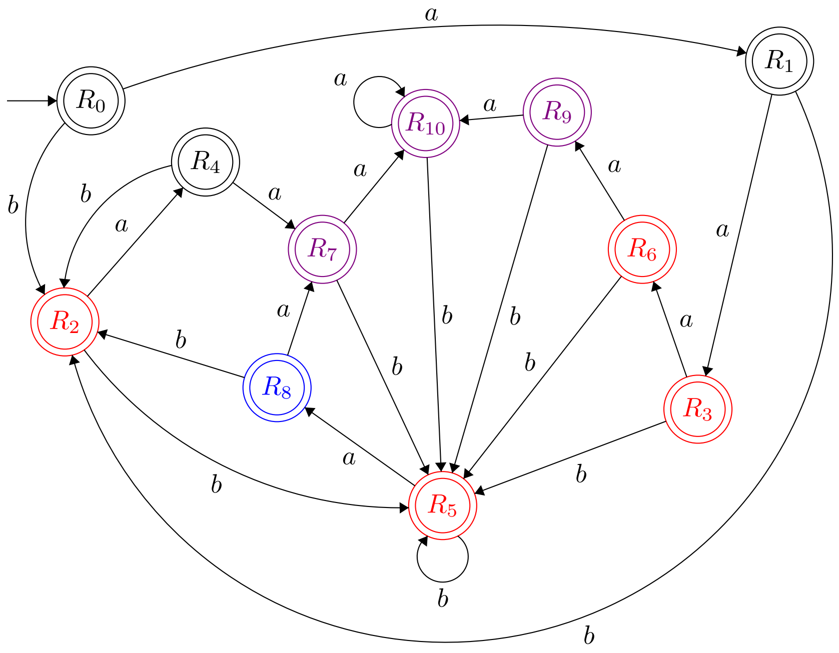


Figure 5.2: The right automaton of the two-step bimachine from example 5.4. The colors of the states are not part of the construction, their only purpose is to make the example easier to understand. If $t = \alpha\beta$, after reading β from right to left, it lands on a state colored in

- red iff only an empty (but no non-empty) replacement may occur just after α (the enumeration part of red states contain q_4 and do not contain q_0),
- blue iff only a non-empty (but no empty) replacement may occur just after α (the enumeration part of blue states contain q_0 and do not contain q_4),
- violet iff both empty and non-empty replacement may occur just after α (the enumeration part of violet states contain both q_0 and q_4).

Note that in our construction, if after reading β from right to left, the right automaton lands on a state colored in violet, an empty replacement will never occur just after α (non-empty replacements are longer than empty replacements).

q	L	R	$\tau(q,L,R)$	$\psi_\tau(q,L,R)$
q_4, q_{err}	L_1	R_7, R_8, R_9, R_{10}	q_0	ϵ
q_0	any	any	q_0	ϵ
q_1	any	any	q_1	ϵ
q_2	any	any	q_2	ϵ
q_3	any	any	q_3	ϵ

Table 5.2: The functions τ and ψ_τ of $\mathcal{A}$ from example 5.4. Unless otherwise specified, $\tau(q,L,R) = q_{\text{err}}$ and $\psi_\tau(q,L,R) = \epsilon$.

$$\begin{tikzcd}
R_{10} & R_{10} \arrow[l, "a"'] & R_{10} \arrow[l, "a"'] & R_{10} \arrow[l, "a"'] & R_7 \arrow[l, "a"'] & R_8 \arrow[l, "a"'] & R_5 \arrow[l, "a"'] & R_{10} \arrow[l, "b"'] & R_7 \arrow[l, "a"'] & R_4 \arrow[l, "a"'] & R_2 \arrow[l, "a"'] & R_0 \arrow[l, "b"'] \\
L_0 \arrow[r, "a"] & L_1 \arrow[r, "a"] & L_1 \arrow[r, "a"] & L_1 \arrow[r, "a"] & L_1 \arrow[r, "a"] & L_1 \arrow[r, "a"] & L_1 \arrow[r, "b"] & L_0 \arrow[r, "a"] & L_1 \arrow[r, "a"] & L_1 \arrow[r, "a"] & L_1 \arrow[r, "b"] & L_0 \\
q_{\text{err}} \arrow[d, "\epsilon"'] & q_{\text{err}} \arrow[d, "\epsilon"'] & q_1 \arrow[d, "\epsilon"'] & q_2 \arrow[d, "\epsilon"'] & q_2 \arrow[d, "\epsilon"'] & q_3 \arrow[d, "\epsilon"'] & q_4 \arrow[d, "\epsilon"'] & q_{\text{err}} \arrow[d, "\epsilon"'] & q_{\text{err}} \arrow[d, "\epsilon"'] & q_3 \arrow[d, "\epsilon"'] & q_4 \arrow[d, "\epsilon"'] & q_{\text{err}} \arrow[d, "\epsilon"'] \\
q_{\text{err}} \arrow[ur, "a", "a"'] & q_0 \arrow[ur, "a", "\epsilon"'] & q_1 \arrow[ur, "a", "\epsilon"'] & q_2 \arrow[ur, "a", "\epsilon"'] & q_2 \arrow[ur, "a", "A"'] & q_3 \arrow[ur, "a", "\epsilon"'] & q_{\text{err}} \arrow[ur, "b", "b"'] & q_{\text{err}} \arrow[ur, "a", "a"'] & q_0 \arrow[ur, "a", "A"'] & q_3 \arrow[ur, "a", "\epsilon"'] & q_{\text{err}} \arrow[ur, "b", "b"'] & q_{\text{err}}
\end{tikzcd}$$

Figure 5.3: The execution of $\mathcal{A}$ over $t = aaaaaabaaab$. $\text{Out}_{\mathcal{A}}(t) = aAbaAb$.

$\delta_R((R', g'), a)$. For this purpose, we will define a sequence $(g_i)_{i=0}^{|g'|-1}$ where

$$\begin{aligned}
g_0 &= \text{enumerate}(\{q \mid (q, a, g'(0)) \in \Delta_T^{\Sigma_1}\}), \\
g_{i+1} &= g_i \parallel \text{enumerate}(\{q \mid (q, a, g'(i+1)) \in \Delta_T^{\Sigma_1} \ \&\ q \notin \text{Rng}(g_i)\}).
\end{aligned}$$

Note that the requirement $q \notin \text{Rng}(g_i)$ ensures that g_i and the new enumeration generated in step $i+1$ have no common elements in their range, so the concatenation is well-defined and therefore the whole sequence is well-defined. Let $g_{\text{old}} = g_{|g'|-1}$. Let g_{new} be the enumeration of $R \cap F_T$ in which the states are sorted in ascending order by their type. $\text{Rng}(g_{\text{old}}) \cap \text{Rng}(g_{\text{new}}) = \emptyset$ because, for each $1 \le r \le m$, $\mathcal{T}_r$ is simple and in particular there are no transitions in Δ_T coming out of $f_{\mathcal{T}_r}$ (so $\text{Rng}(g_{\text{old}}) \cap F_T = \emptyset$). We claim that

$$g = g_{\text{old}} \parallel g_{\text{new}}$$

has the desired properties. Let $i \le j$. It is not possible that $g(i) \in F_T$ and $g(j) \notin F_T$. If $g(i), g(j) \in F_T$, then clearly $g(i), g(j) \in \text{Rng}(g_{\text{new}})$ and so $\text{type}(g(i)) \le \text{type}(g(j))$ because the states in g_{new} are sorted in ascending order by their type. Now assume $g(i), g(j) \notin F_T$. Then $g(i), g(j) \in \text{Rng}(g_{\text{old}})$. Let $k_1 = \min\{k \mid g(i) \in \text{Rng}(g_k)\}$ and $k_2 = \min\{k \mid g(j) \in \text{Rng}(g_k)\}$. $i \le j$ implies that $k_1 \le k_2$. By the definition of k_1, $(g(i), a, g'(k_1)) \in \Delta_T^{\Sigma_1}$ and also $(\forall k < k_1)(g(i) \notin \text{Rng}(g_k))$. Therefore $(\forall k < k_1)(g(i), a, g'(k)) \notin \Delta_T^{\Sigma_1}$. This means $k_1 = \mu(g(i), a, g')$. Similarly, we can see that $k_2 = \mu(g(j), a, g')$. It remains to see that $g \in \text{Enum}(R \cap Q_T)$, i.e. $\text{Rng}(g) = R \cap Q_T$:

$\subseteq$: Let $q \in \text{Rng}(g)$. If $q \in \text{Rng}(g_{\text{new}})$, then $q \in (R \cap F_T) \subseteq (R \cap Q_T)$. Otherwise

$q \in \mathrm{Rng}(g_{\mathrm{old}})$ and there is $q' \in \mathrm{Rng}(g') = R' \cap Q_T$ such that $(q, a, q') \in \Delta_T^{\Sigma_1}$. Therefore $q \in R \cap Q_T$.

$\supseteq$: Let $q \in R \cap Q_T$. If $q \in F_T$, then $q \in \mathrm{Rng}(g_{\mathrm{new}}) \subseteq \mathrm{Rng}(g)$. Assume $q \notin F_T$. Since $q \in R$, there is $q' \in R'$ such that $(q', a, q) \in \Delta_{RU}$. Since $q \in Q_T \setminus F_T$, it follows that $q' \in Q_T$ which means $(q, a, q') \in \Delta_T^{\Sigma_1}$ and hence $q' \in (R' \cap Q_T) = \mathrm{Rng}(g')$. Therefore $q \in \mathrm{Rng}(g_i)$ for some $0 \leq i < |g'|$. Thus $q \in \mathrm{Rng}(g_{\mathrm{old}}) \subseteq \mathrm{Rng}(g)$.

With the above reasonings we have shown that g has the desired properties and therefore the definition of δ_R is correct.

For δ, let $(R, g) = \delta_R((R', g'), a)$ and assume $\overline{q} \in \mathrm{Rng}(g) \setminus F_T$. Then $\overline{q} \in R = \mathrm{Succ}(R', a, \Delta_{RU})$ which means that there exists $q' \in R'$ such that $(q', a, \overline{q}) \in \Delta_{RU}$. Since $\overline{q} \in \mathrm{Rng}(g) \setminus F_T$, we can conclude that $(q', a, \overline{q}) \in \left(\Delta_T^{\Sigma_1}\right)^{\mathrm{rev}}$, i.e. $(\overline{q}, a, q') \in \Delta_T^{\Sigma_1}$. Therefore $q' \in (R' \cap Q_T) = \mathrm{Rng}(g')$, so $\{i' \mid (\overline{q}, a, g'(i')) \in \Delta_T^{\Sigma_1}\} \neq \emptyset$. Hence $\mu(\overline{q}, a, g')$ is defined. All elements of $\{i' \mid (\overline{q}, a, g'(i')) \in \Delta_T^{\Sigma_1}\}$ belong to the domain of g' and in particular $\mu(\overline{q}, a, g')$ does. We can conclude that $g'(\mu(\overline{q}, a, g'))$ is defined and so is $\delta(\overline{q}, a, (R', g'))$.

For ψ_δ, let $q' = \delta(\overline{q}, a, (R', g'))$ and assume $q' \in Q_T$. Note that $q' \notin S_T$. We have that $(\overline{q}, a, q') \in \Delta_T^{\Sigma_1}$, so $\{m \mid (\overline{q}, (a, m), q') \in \Delta_T\} \neq \emptyset$. Hence $\psi_\delta(\overline{q}, a, (R', g'))$ is correctly defined. $\square$

Let $\mathrm{BR} = (T_r/\lambda_r\text{-}\rho_r)_{r=1}^m$ be a BCRR over (Σ_1, Σ_2). Let $A = (\Sigma_1, \Sigma_2, A_L, A_R, Q, q_{\mathrm{err}}, \delta, \tau, \psi_\delta, \psi_\tau)$ be obtained from construction 5.3 applied on $((\mathcal{T}_r, \mathcal{E}_r, A_{\lambda_r}, A_{\rho_r}))_{r=1}^m$ – a FSR of BR.

Proposition 5.6. *Let $t = t_1 \dots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. If $\overline{q_i} \in Q_T$, then $\overline{q_i} \in \mathrm{Rng}(g_i) \setminus F_T$.*

Proof. $q_i \notin Q_T \setminus F_T$: From $\overline{q_i} \neq q_{\mathrm{err}}$ it follows that $\overline{q_i} \in \mathrm{Rng}(g_i)$ and $\overline{q_i} \in S_T$. Since $S_T \cap F_T = \emptyset$, we have $\overline{q_i} \notin F_T$ and hence $\overline{q_i} \in \mathrm{Rng}(g_i) \setminus F_T$.

$q_i \in Q_T \setminus F_T$: Recall that $q_0 = q_{\mathrm{err}} \notin Q_T$. Therefore $i > 0$. Then $q_i = \delta(\overline{q_{i-1}}, t_i, (R_i, g_i)) \in \mathrm{Rng}(g_i)$. By definition, $\overline{q_i} = \tau(q_i, L_i, (R_i, g_i)) = q_i \in \mathrm{Rng}(g_i)$. Since $q_i \notin F_T$ and $\overline{q_i} = q_i$, it follows $\overline{q_i} \in \mathrm{Rng}(g_i) \setminus F_T$.

$\square$

Proposition 5.7. *Let $t = t_1 \dots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. Then $\overline{q_n} = q_{\mathrm{err}}$.*

Proof. Suppose $\overline{q_n} \neq q_{\mathrm{err}}$. Then $\overline{q_n} \in Q_T \setminus F_T$.

$\overline{q_n} \in S_T$: Then $\overline{q_n} \in \mathrm{Rng}(g_n) = \mathrm{Rng}(g_s) \subseteq F_T$ - an immediate contradiction.

$\overline{q_n} \in Q_T \setminus (S_T \cup F_T)$: Then $\overline{q_n} = q_n \notin F_T$. But $q_n \in \mathrm{Rng}(g_n) = \mathrm{Rng}(g_s) \subseteq F_T$ - again a contradiction.

$\square$

Proposition 5.8. *Let $t = t_1 \dots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. Then, for each $0 \le i \le n$,*

$$\begin{aligned}
f_{\lambda_r} \in L_i &\iff t_1 \dots t_i \in \Sigma_1^* \cdot \lambda_r, \\
s_{T_r} \in R_i &\iff t_{i+1} \dots t_n \in (\mathrm{Dom}(T_r) \setminus \{\epsilon\}) \cdot \rho_r \cdot \Sigma_1^*, \\
f_{T_r} \in R_i &\iff t_{i+1} \dots t_n \in \rho_r \cdot \Sigma_1^*.
\end{aligned}$$

Definition 5.9. *Let $t, v \in \Sigma_1^*$, $q \in Q_T$ and $r = \mathrm{type}(q)$. We define the following helper predicates:*

$$\begin{aligned}
\mathrm{submatch}(q, v, t) &\iff v \in \overrightarrow{\mathrm{L}_{\mathcal{T}_r^{\mathrm{dom}}}}(q) \ \&\ v^{-1}t \in \rho_r \Sigma_1^*, \\
\mathrm{lsubmatch}(q, v, t) &\iff \mathrm{submatch}(q, v, t) \ \&\ \forall \overline{v}(\mathrm{submatch}(q, \overline{v}, t) \implies |\overline{v}| \le |v|).
\end{aligned}$$

Proposition 5.10. *Let $t, v \in \Sigma_1^*$, $(R, g) = \delta_R^*(s_R, \mathrm{rev}(t))$, $j \in \mathrm{Dom}(g)$ and $\mathrm{submatch}(g(j), v, t)$. If $i \le j$, then $\exists v_l(\mathrm{submatch}(g(i), v_l, t) \ \&\ |v_l| \ge |v|)$.*

Proof. Let $i \le j$, $r_i = \mathrm{type}(g(i))$ and $r_j = \mathrm{type}(g(j))$. The proof goes by induction on $|v|$:

$v = \epsilon$: By definition $g(i) \in R \cap Q_T$, so there is v_l such that $\mathrm{submatch}(g(i), v_l, t)$. Of course, $|v_l| \ge |v| = 0$.

$v = av', a \in \Sigma_1$: Let $(R', g') = \delta_R^*(s_R, \mathrm{rev}(a^{-1}t))$. Since $v \ne \epsilon$, $v \in \overrightarrow{\mathrm{L}_{\mathcal{T}_{r_j}^{\mathrm{dom}}}}(g(j))$ and $\Delta_{T_{r_j}}^{\Sigma_1} \cap (\{f_{T_{r_j}}\} \times \Sigma_1 \times Q_{T_{r_j}}) = \emptyset$, it follows that $g(j) \notin F_T$. Then, by the definition of g, we get that $i_0' := \mu(g(i), a, g') \le \mu(g(j), a, g') =: j_0'$. Let $q' \in Q_T$ be such that $(g(j), a, q') \in \Delta_{T_{r_j}}^{\Sigma_1}$ and $v' \in \overrightarrow{\mathrm{L}_{\mathcal{T}_{r_j}^{\mathrm{dom}}}}(q')$. Then $\mathrm{submatch}(q', v', a^{-1}t)$. Note that $q' \in \mathrm{Rng}(g')$ and let j' be such that $q' = g'(j')$. Then $j' \in \{j' \mid (g(j), a, g'(j')) \in \Delta_T^{\Sigma_1}\}$ which means $i_0' \le j_0' \le j'$. By the i.h. we get that

$$\exists v_l'(\mathrm{submatch}(g'(i_0'), v_l', a^{-1}t) \ \&\ |v_l'| \ge |v'|).$$

Take a witness v_l' for this existence. Since $(g(i), a, g'(i_0')) \in \Delta_{T_{r_i}}^{\Sigma_1}$ and $v_l' \in \overrightarrow{\mathrm{L}_{\mathcal{T}_{r_i}^{\mathrm{dom}}}}(g'(i_0'))$, it follows that $av_l' \in \overrightarrow{\mathrm{L}_{\mathcal{T}_{r_i}^{\mathrm{dom}}}}(g(i))$. Hence $\mathrm{submatch}(g(i), av_l', t)$ and $|av_l'| \ge |av'| = |v|$.

□

Proposition 5.11. *Let $v = av' \in \Sigma_1^*$, $t = at' \in \Sigma_1^*$, $q \in Q_T$ and $\mathrm{lsubmatch}(q, v, t)$. If $(q, a, q') \in \Delta_T^{\Sigma_1}$ and $\mathrm{submatch}(q', \overline{v}, t')$, then $|\overline{v}| \le |v'|$.*

Proof. Suppose that $|\overline{v}| > |v'|$. Then $\mathrm{submatch}(q, a\overline{v}, t)$ and $|a\overline{v}| > |av'| = |v|$. This is a contradiction with $\mathrm{lsubmatch}(q, v, t)$. □

Proposition 5.12. *Let $v = av' \in \Sigma_1^*$, $t = at' \in \Sigma_1^*$, $(R, g) = \delta_R^*(s_R, \mathrm{rev}(t))$, $(R', g') = \delta_R^*(s_R, \mathrm{rev}(t'))$ and $q \in \mathrm{Rng}(g)$. If $\mathrm{lsubmatch}(q, v, t)$, then $\mathrm{lsubmatch}(g'(\mu(q, a, g')), v', t')$.*

Proof. Let $k_0 = \mu(q, a, g')$. There exists $q' \in \mathrm{Rng}(g')$ such that $(q, a, q') \in \Delta_T^{\Sigma_1}$ and $\mathrm{submatch}(q', v', t')$. Let k' be such that $q' = g'(k')$. Then $k_0 \leq k' \in \mathrm{Dom}(g')$. By proposition 5.10 (since $\mathrm{submatch}(g'(k'), v', t')$), there is v_l such that $\mathrm{submatch}(g'(k_0), v_l, t')$ and $|v_l| \geq |v'|$. By proposition 5.11, $|v_l| \leq |v'|$. Therefore $|v_l| = |v'|$. Both v_l and v' are prefixes of the same string t', so $v_l = v'$. Thus $\mathrm{submatch}(g'(k_0), v', t')$ and applying proposition 5.11 once again gives us $\mathrm{lsubmatch}(g'(k_0), v', t')$. □

Corollary 5.13. *Let $t = t_1 \ldots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. If $\overline{q_i} \in Q_T \setminus F_T$ and $\mathrm{lsubmatch}(\overline{q_i}, t_{i+1}v', t_{i+1} \ldots t_n)$, then $(\overline{q_i}, t_{i+1}, q_{i+1}) \in \Delta_T^{\Sigma_1}$ and $\mathrm{lsubmatch}(q_{i+1}, v', t_{i+2} \ldots t_n)$.*

Proof. By proposition 5.6, $\overline{q_i} \in \mathrm{Rng}(g_i)$. Note that $i < n$ (proposition 5.7). Then $q_{i+1} = \delta(\overline{q_i}, t_{i+1}, (R_{i+1}, g_{i+1})) = g_{i+1}(\mu(\overline{q_i}, t_{i+1}, g_{i+1}))$. Clearly, $(\overline{q_i}, t_{i+1}, q_{i+1}) \in \Delta_T^{\Sigma_1}$. For $\mathrm{lsubmatch}(q_{i+1}, v', t_{i+2} \ldots t_n)$, apply proposition 5.12. □

Proposition 5.14. *Let $t, v \in \Sigma_1^*$, $(R, g) = \delta_R^*(s_R, \mathrm{rev}(t))$, $i, j \in \mathrm{Dom}(g)$, $\mathrm{lsubmatch}(g(i), v, t)$ and $\mathrm{lsubmatch}(g(j), v, t)$. If $i \leq j$, then $\mathrm{type}(g(i)) \leq \mathrm{type}(g(j))$.*

Proof. Assume $i \leq j$. By induction on $|v|$:

$v = \epsilon$: In this case $g(i) \in F_T$ and $g(j) \in F_T$. Thus, by the definition of g, $\mathrm{type}(g(i)) \leq \mathrm{type}(g(j))$.

$v = av', a \in \Sigma_1$: In this case $g(i) \notin F_T$ and $g(j) \notin F_T$. Thus, by the definition of g, $i_0' := \mu(g(i), a, g') \leq \mu(g(j), a, g') =: j_0'$. Proposition 5.12 gives us $\mathrm{lsubmatch}(g'(i_0'), v', a^{-1}t)$ and $\mathrm{lsubmatch}(g'(j_0'), v', a^{-1}t)$. By the i.h. $\mathrm{type}(g'(i_0')) \leq \mathrm{type}(g'(j_0'))$. But $\mathrm{type}(g'(i_0')) = \mathrm{type}(g(i))$ and $\mathrm{type}(g'(j_0')) = \mathrm{type}(g(j))$.

□

Proposition 5.15. *Let $t = t_1 \ldots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. Let $\overline{q_i} = s_{T_r}$ and $\mathrm{lsubmatch}(\overline{q_i}, v, t_{i+1} \ldots t_n)$. Then for each $i < k < i + |v|$, $q_k \in Q_T \setminus F_T$ and $q_{i+|v|} \in F_T$. Furthermore, $\overline{q_i}, q_{i+1}, \ldots, q_{i+|v|}$ is a successful execution of $\mathcal{T}_r^{\mathrm{dom}}$ over v.*

Proof. Obviously $v \neq \epsilon$ since $\epsilon \notin \mathrm{L}(\mathcal{T}_r^{\mathrm{dom}})$ (in particular $s_{T_r} \neq f_{T_r}$). The proof can be completed trivially by induction on $|v|$ applying corollary 5.13. □

Proposition 5.16. *Let $t = t_1 \ldots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. Then*

$$\overline{q_i} \in S_T \iff (\exists (u, v, w) \in C_{LML})(|u| = i \ \&\ v \neq \epsilon).$$

Proof. By complete induction on i: Let $0 \leq i \leq n$ and assume that the claim is true for all $j < i$.

$\Longrightarrow$: Let $\overline{q_i} \in S_T$. Then $\overline{q_i} = \tau(q_i, L_i, (R_i, g_i)) = g_i(\min(I(L_i, g_i)))$. Therefore, for $r = \text{type}(\overline{q_i})$, $f_{\lambda_r} \in L_i$, $s_{T_r} \in R_i$ and $q_i \notin Q_T \setminus F_T$. The first two imply that $t_1 \dots t_i \in \Sigma_1^* \cdot \lambda_r$ and $t_{i+1} \dots t_n \in (\text{Dom}(T_r) \setminus \{\epsilon\}) \cdot \rho_r \cdot \Sigma_1^*$, thus

$$\exists v\, (t_1 \dots t_i, v, v^{-1}(t_{i+1} \dots t_n)) \in C.$$

Take the longest such v and let $u = t_1 \dots t_i$ and $w = v^{-1}(t_{i+1} \dots t_n)$. $v \in \text{Dom}(T_r) \setminus \{\epsilon\}$, so $v \neq \epsilon$. Suppose that $(u, v, w) \notin C_{LM}$. By proposition 3.11 there is $(u', v', w') \in C_{LML}$ such that $\text{Over}((u, v, w), (u', v', w'))$, i.e. $|u'| < |u| < |u'v'|$. $v' \neq \epsilon$ because otherwise $|u'| < |u| < |u'|$ which is impossible. By the i.h. $\overline{q_{|u'|}} \in S_T$. By proposition 5.15 we have that $q_k \in Q_T \setminus F_T$ for each $|u'| < k < |u'v'|$. But $|u'| < i < |u'|$, so $q_i \in Q_T \setminus F_T$. This contradiction shows that $(u, v, w) \in C_{LM}$. The choice of v gives us $(u, v, w) \in C_{LML}$.

$\Longleftarrow$: Let $(u, v, w) \in C_{LML}$, $|u| = i$ and $v \neq \epsilon$. Then $I(L_i, g_i) \neq \emptyset$. Suppose $q_i \in Q_T \setminus F_T$. Then there is $j < i$ such that $\overline{q_j} \in S_T$. Take the largest such j. By the i.h. there exists $(u', v', w') \in C_{LML}$, $|u'| = j$ and $v' \neq \epsilon$. We now have $\text{lsubmatch}(\overline{q_j}, v', t_{j+1} \dots t_n)$ and we can apply proposition 5.15 to get that, for each $j < k < j + |v'|$, $q_k \in Q_T \setminus F_T$ and $q_{j+|v'|} \in F_T$. Suppose that $|u| \geq |u'v'| = j + |v'|$.

Case 1: $i = j + |v'|$. Then $q_i = q_{j+|v'|} \in F_T$, which is a contradiction.

Case 2: $i > j + |v'|$. Then $q_{j+|v'|} \in F_T$ and hence $\overline{q_{j+|v'|}} \in S_T \cup \{q_{\text{err}}\}$. Therefore j is not the largest such that $\overline{q_j} \in S_T$, which is also a contradiction.

Therefore $|u'| < |u| < |u'v'|$, which means $\text{Overlap}((u, v, w), (u', v', w'))$. But this is impossible (proposition 3.9), so our assumption must be wrong and $q_i \notin Q_T \setminus F_T$. Then $\overline{q_i} = \tau(q_i, L_i, (R_i, g_i)) = g_i(\min(I(L_i, g_i))) \in S_T$.

□

Corollary 5.17. *Let $t = t_1 \dots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. Then*

$$q_i \in Q_T \setminus F_T \implies \neg(\exists (u, v, w) \in C_{LML})(|u| = i), \tag{1}$$

$$q_i \notin Q_T \setminus F_T \implies (\forall (u, v, w) \in C)(|u| = i \implies (u, v, w) \in C_{LM}). \tag{2}$$

Proof. The proof of (1) and (2) are similar to right-to-left and left-to-right directions of the proof of proposition 5.16 respectively. □

Proposition 5.18. *Let $t = t_1 \dots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. If $\overline{q_i} = s_{T_r}$ and $\text{lsubmatch}(\overline{q_i}, v, t_{i+1} \dots t_n)$, then $o_{i+1}\overline{o_{i+1}}o_{i+2}\overline{o_{i+2}} \dots o_{i+|v|-1}\overline{o_{i+|v|-1}}o_{i+|v|} = T_r(v)$.*

Proof. By proposition 5.15 and the definition of ψ_δ, $\overline{q_i}, q_{i+1}, \dots, q_{i+|v|}$ is a successful execution of $\mathcal{T}_r$ over $(v, o_{i+1}o_{i+2} \dots o_{i+|v|})$. $\text{L}(\mathcal{T}_r) \subseteq T_r$, T_r is a function and therefore $o_{i+1}o_{i+2} \dots o_{i+|v|} = T_r(v)$. Now let $i < k < i + |v|$. $q_k \in Q_T \setminus F_T$, so $\tau(q_k, L_k, R_k) = q_k \neq q_{\text{err}}$. Therefore $\overline{o_k} = \epsilon$. □

Proposition 5.19. *Let $t = t_1 \dots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. If $q_{i+1} \in Q_T$, then $|\{m \mid (\overline{q_i}, (a, m), q_{i+1}) \in \Delta_T\}| = 1$.*

Proof. Let $M = \{m \mid (\overline{q_i}, (a, m), q_{i+1}) \in \Delta_T\}$. From the correctness of the construction we know that $M \neq \emptyset$. Let $m_1, m_2 \in M$. Take the largest $j \leq i$ such that $\overline{q_j} = s_{T_r}$ for some $1 \leq r \leq m$. Let v be such that $\text{lsubmatch}(\overline{q_j}, v, t_{j+1} \dots t_n)$. Then $j + |v| \geq i + 1$. By proposition 5.18,

$$o_{j+1} \dots o_i m_1 o_{i+2} \dots o_{j+|v|} = T_r(v) = o_{j+1} \dots o_i m_2 o_{i+2} \dots o_{j+|v|}.$$

Therefore $m_1 = m_2$ and since m_1 and m_2 are arbitrary elements of M, it means $|M| = 1$. □

Proposition 5.20. *Let $t = t_1 \dots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. Let $x_0 y_0 \dots x_n y_n$ be a rewriting of t according to* BR, *where x_i and y_i are as in definition 3.12, $0 \leq i \leq n$. Then $\overline{o_i} = y_i$.*

Proof. Let $c = (t_1 \dots t_i, \epsilon, t_{i+1} \dots t_n)$.

Case 1: $\tau(q_i, L_i, (R_i, g_i)) \neq q_{\text{err}} \ \vee \ J(L, R) = \emptyset$. Then $\overline{o_i} = \epsilon$.

If $\tau(q_i, L_i, (R_i, g_i)) \neq q_{\text{err}}$, then $\overline{q_i} \in S_T$ or $\overline{q_i} = q_i \in Q_T \setminus F_T$. If $\overline{q_i} \in S_T$, by proposition 5.16

$$(\exists (u, v, w) \in C_{LML})(|u| = i \ \& \ v \neq \epsilon).$$

If $q_i \in Q_T \setminus F_T$, by corollary 5.17

$$\neg(\exists (u, v, w) \in C_{LML})(|u| = i).$$

If $J(L, R) = \emptyset$, then $c \notin C$.

In all cases $c \notin C_{LML}$ which means $y_i = \epsilon$.

Case 2: $\tau(q_i, L_i, (R_i, g_i)) = q_{\text{err}} \ \& \ J(L, R) \neq \emptyset$. Let $r = \min(J(L, R))$. Then $\overline{o_i} = \mathcal{E}_r = T_r(\epsilon)$. Since $J(L, R) \neq \emptyset$, $c \in C$. By corollary 5.17 (since $q_i \notin Q_T \setminus F_T$), $c \in C_{LM}$. Since $\overline{q_i} \notin S_T$, $c \in C_{LML}$ (proposition 5.16). Thus $y_i = T_{\text{earliest}(c)}(\epsilon)$. It is clear that $J(L, R) = \{r \mid \text{RC}(c, t, T_r/\lambda_r\text{-}\rho_r)\}$, so $r = \text{earliest}(c)$.

□

Proposition 5.21. *Let $t = t_1 \dots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$ be an execution of A over t. Let $x_0 y_0 \dots x_n y_n$ be a rewriting of t according to* BR, *where x_i and y_i are as in definition 3.12, $0 \leq i \leq n$. If $\overline{q_i} \in S_T \cup \{q_{\text{err}}\}$, then $o_0 \overline{o_0} \dots o_i \overline{o_i} = x_0 y_0 \dots x_i y_i$.*

Proof. By complete induction on i:

$i = 0$: $o_0 = x_0 = \epsilon$ (by definition), $\overline{o_0} = y_0$ (proposition 5.20).

$0 < i \leq n$: $\overline{o_i} = y_i$ (proposition 5.20). Note that $q_i \in F_T \cup \{q_{\text{err}}\}$.

Case 1: $q_i = q_{\text{err}}$. Then $\overline{q_{i-1}} = q_{\text{err}}$. By the i.h. $o_0\overline{o_0}\dots o_{i-1}\overline{o_{i-1}} = x_0y_0\dots x_{i-1}y_{i-1}$. Clearly $o_i = \psi_\delta(\overline{q_{i-1}}, t_i, (R_i, g_i)) = t_i$. We want to show that $x_i = t_i$. Suppose that there exists $(u, v, w) \in C_{LML}$ such that $|u| < i \leq |uv|$. Then we immediately get that $q_i \in Q_T$ (proposition 5.15) which is a contradiction. Therefore x_i could not have been defined by the first two cases of its definition, thus $x_i = t_i$.

Case 2: $q_i \in F_T$. Take the largest $j < i$ such that $\overline{q_j} \in S_T$. Let $r = \text{type}(\overline{q_j})$. By the i.h. $o_0\overline{o_0}\dots o_j\overline{o_j} = x_0y_0\dots x_jy_j$. By proposition 5.16

$$(\exists(u, v, w) \in C_{LML})(|u| = j \;\&\; v \neq \epsilon).$$

Take a witness $c = (u, v, w)$ for the existence above and note that $|uv| = i$. Let $r_0 = \text{earliest}(c)$. Then we have all of the following:

- $x_{j+1} = T_{r_0}(v)$,
- $x_{j+2} = \dots = x_i = \epsilon$,
- $y_{j+1} = \dots = y_{i-1} = \epsilon$,
- $o_{j+1}\overline{o_{j+1}}o_{j+2}\overline{o_{j+2}}\dots o_{i-1}\overline{o_{i-1}}o_i = T_r(v)$ (proposition 5.18).

Let $k_0 = \min(I(L_j, g_j))$. Then $\overline{q_j} = g_j(k_0) = s_{T_r}$. Clearly, $r \in \{r \mid \text{RC}(c, t, T_r/\lambda_r\text{-}\rho_r)\}$ and also $k_0 \leq g^{-1}(s_{T_{r_0}}) \in I(L_j, g_j)$. By proposition 5.14,

$$r = \text{type}(g(k_0)) \leq \text{type}(g(g^{-1}(s_{T_{r_0}}))) = \text{type}(s_{T_{r_0}}) = r_0.$$

$r_0 = \min\{r \mid \text{RC}(c, t, T_r/\lambda_r\text{-}\rho_r)\}$, therefore $r = r_0$ and hence

$$o_{j+1}\overline{o_{j+1}}o_{j+2}\overline{o_{j+2}}\dots o_{i-1}\overline{o_{i-1}}o_i = x_{j+1}y_{j+1}x_{j+2}y_{j+2}\dots x_{i-1}y_{i-1}x_i.$$

□

Theorem 5.22. *Let $t \in \Sigma_1^*$. Then* $\text{Out}_A(t) = \text{Rewrite}_{LML}(\text{BR}, t)$.

Proof. By proposition 5.7 we have $\overline{q_n} = q_{\text{err}}$. Therefore we can apply proposition 5.21 to get the desired result. □

For completeness, we are going to show that the expressive power of the two-step bimachines coincides with the class of the regular functions. It is obvious that for every two-step bimachine $\mathcal{A}$, $\text{Out}_\mathcal{A}$ is a function. In the following proposition we will see that $\text{Out}_\mathcal{A}$ is regular relation.

Proposition 5.23. *For every two-step bimachine $\mathcal{A} = (\Sigma_1, \Sigma_2, A_L, A_R, Q, q_{\text{err}}, \delta, \tau, \psi_\delta, \psi_\tau)$, there exists a finite-state transducer $\mathcal{T}$, such that* $\text{Out}_\mathcal{A} = \text{L}(\mathcal{T})$.

Proof. Consider the FST

$$\mathcal{T} = (\Sigma_1, \Sigma_1\cup\Sigma_2, 2\times Q\times Q_L\times Q_R, \{0\}\times\{q_{\text{err}}\}\times\{s_L\}\times Q_R, \{1\}\times Q\times Q_L\times\{s_R\}, \Delta),$$

where

$$\begin{aligned}\Delta =&\{((0, q, L, R), (\epsilon, \psi_\tau(q, L, R)), (1, \tau(q, L, R), L, R)) \mid q \in Q \;\&\; L \in Q_L \;\&\; R \in Q_R\}\cup\\ &\{((1, q, L, R), (a, \psi_\delta(q, a, R')), (0, \delta(q, a, R'), \delta_L(L, a), R')) \mid q \in Q \;\&\; L \in Q_L \;\&\; R = \delta_R(R', a)\}.\end{aligned}$$

It is easy to see that each successful path in $\mathcal{T}$ corresponds to a unique execution of $\mathcal{A}$ and vice versa.

$\subseteq$: Let $t = t_1 \dots t_n \in \Sigma_1^*$ and $((L_i, q_i, \overline{q_i}, R_i, o_i, \overline{o_i}))_{i=0}^n$ be an execution of $\mathcal{A}$ over t. Then

$$((0, q_0, L_0, R_0), (\epsilon, \overline{o_0}), (1, \overline{q_0}, L_0, R_0)) \in \Delta,$$

and, for each $0 < i \leq n$,

$$((1, \overline{q_{i-1}}, L_{i-1}, R_{i-1}), (t_i, o_i), (0, q_i, L_i, R_i)) \in \Delta,$$
$$((0, q_i, L_i, R_i), (\epsilon, \overline{o_i}), (1, \overline{q_i}, L_i, R_i)) \in \Delta.$$

Therefore

$$(0, q_0, L_0, R_0) \xrightarrow{(\epsilon,\overline{o_0})} (1, \overline{q_0}, L_0, R_0) \xrightarrow{(t_1,o_1)} (0, q_1, L_1, R_1) \xrightarrow{(\epsilon,\overline{o_1})} (1, \overline{q_1}, L_1, R_1)$$
$$\xrightarrow{(t_2,o_2)} \cdots \xrightarrow{(\epsilon,\overline{o_{n-1}})} (1, \overline{q_{n-1}}, L_{n-1}, R_{n-1}) \xrightarrow{(t_n,o_n)} (0, q_n, L_n, R_n) \xrightarrow{(\epsilon,\overline{o_n})} (1, \overline{q_n}, L_n, R_n)$$

is a path in $\mathcal{T}$. Since $q_0 = q_{\text{err}}$, $L_0 = s_L$ and $R_n = s_R$, this path corresponds to a successful execution of $\mathcal{T}$.

$\supseteq$: By the definition of Δ, every successful execution of $\mathcal{T}$ with label $(t, \_)$ has the form as in the previous case and corresponds to an execution of $\mathcal{A}$ over t.

$\square$

It remains to see that for every regular function regular function f, there exists a two-step bimachine $\mathcal{A}$ such that $f = \text{Out}_{\mathcal{A}}$. We will show this in the next section (proposition 6.5) after we introduce the classical bimachines.

6 Classical bimachines

Definition 6.1. *A classical bimachine is a tuple $\mathcal{B} = (\Sigma_2, A_L, A_R, \psi)$, where:*

- *Σ_2 is a finite set - the output alphabet,*
- *$A_L = (\Sigma_1, Q_L, s_L, Q_L, \delta_L)$ and $A_R = (\Sigma_1, Q_R, s_R, Q_R, \delta_R)$ are DFA - the left and right automata of the bimachine,*
- *$\psi : Q_L \times \Sigma_1 \times Q_R \nrightarrow \Sigma_2^*$ is a partial function - the output function.*

Definition 6.2. *We say that $\mathcal{B} = (\Sigma_2, A_L, A_R, \psi, \iota)$ is a classical bimachine with final output if $(\Sigma_2, A_L, A_R, \psi)$ is a classical bimachine and $\iota : Q_L \nrightarrow \Sigma_2^*$ is the function producing the final output.*

Definition 6.3. *Let $\mathcal{B} = (\Sigma_2, A_L, A_R, \psi)$ be a classical bimachine. Let $t = t_1 \dots t_n \in \Sigma_1^*$. Let $L_0, \dots, L_n$ be the execution of A_L over t and $R_n, \dots, R_0$ be the execution of A_R over $\text{rev}(t)$. Execution of $\mathcal{B}$ over t is the sequence $((L_i, o_i, R_i))_{i=0}^n$ where:*

- $o_i = \psi(L_i, t_{i+1}, R_{i+1}), \quad 0 \le i < n,$
- $o_n = \epsilon.$

Output of $\mathcal{B}$ *for* $t \in \Sigma_1^*$ *is* $\mathrm{Out}_{\mathcal{B}}(t) = o_0 o_1 \dots o_n$.

If $\mathcal{B} = (\Sigma_2, A_L, A_R, \psi, \iota)$ is a classical bimachine with final output, the definitions of execution and output for a string $t = t_1 \dots t_n \in \Sigma_1^*$ are the same except that we define $o_n = \iota(L_n)$.

In this section we are going to present a construction of a classical bimachine with final output given a FSR of some batch of rules. The next proposition shows that extending classical bimachines with a "final output" function does not change their expressive power (except when the input string is empty).

Proposition 6.4. *For every classical bimachine with final output* $\mathcal{B} = (\Sigma_2, A_L, A_R, \psi, \iota)$*, there is a classical bimachine* $\mathcal{B}' = (\Sigma_2, A_L, A'_R, \psi')$ *such that* $\mathcal{B}$ *and* $\mathcal{B}'$ *are equivalent up to* ϵ*, i.e.* $(\forall t \in \Sigma_1^+)(\mathrm{Out}_{\mathcal{B}}(t) = \mathrm{Out}_{\mathcal{B}'}(t))$.

Proof. Let $\mathcal{B} = (\Sigma_2, A_L, A_R, \psi, \iota)$ be a classical bimachine with final output. Let $s'_R \notin Q_R$. We construct a classical bimachine $\mathcal{B}' = (\Sigma_2, A_L, A'_R, \psi')$ where

- $A'_R = (\Sigma_1, Q_R \cup \{s'_R\}, s'_R, Q_R \cup \{s'_R\}, \delta'_R)$,
- $\delta'_R(R, a) = \begin{cases} \delta_R(R, a) & R \neq s'_R \\ \delta_R(s_R, a) & R = s'_R \end{cases},$
- $\psi'(L, a, R) = \begin{cases} \psi(L, a, R) & R \neq s'_R \\ \psi(L, a, s_R) \cdot \iota(\delta_L(L, a)) & R = s'_R \end{cases}.$

Let $t = t_1 \dots t_n \in \Sigma_1^+$. Let $((L_i, o_i, R_i))_{i=0}^n$ be an execution of $\mathcal{B}$ over t and $((L_i, o'_i, R'_i))_{i=0}^n$ be an execution of $\mathcal{B}'$ over t. Since

$$R'_{n-1} = \delta'_R(R'_n, t_n) = \delta'_R(s'_R, t_n) = \delta_R(s_R, t_n) = R_{n-1}$$

and there are no transitions in A'_R going into s'_R (that is $s'_R \notin \mathrm{Rng}(\delta'_R)$), it follows that $R'_i = R_i$ for all $i < n$. Therefore, for all $i < n-1$,

$$o'_i = \psi'(L_i, t_{i+1}, R'_{i+1}) = \psi'(L_i, t_{i+1}, R_{i+1}) = \psi(L_i, t_{i+1}, R_{i+1}) = o_i.$$

Furthermore,

$$\begin{aligned} o'_{n-1} &= \psi'(L_{n-1}, t_n, R'_n) = \psi'(L_{n-1}, t_n, s'_R) = \psi(L_{n-1}, t_n, s_R) \cdot \iota(L_n), \\ o'_n &= \epsilon. \end{aligned}$$

On the other hand,

$$\begin{aligned} o_{n-1} &= \psi(L_{n-1}, t_n, R_n) = \psi(L_{n-1}, t_n, s_R), \\ o_n &= \iota(L_n). \end{aligned}$$

Thus we get $\mathrm{Out}_{\mathcal{B}}(t) = \mathrm{Out}_{\mathcal{B}'}(t)$. □

As mentioned earlier, we will now show that any regular function can be represented by a two-step bimachine. For that purpose, we will show that for any bimachine with final output, there exists an equivalent two-step bimachine.

Proposition 6.5. *Let $\mathcal{B} = (\Sigma_2, A_L, A_R, \psi, \iota)$ be a classical bimachine with final output. There exists a two-step bimachine $\mathcal{A}$ such that $\mathrm{Out}_{\mathcal{B}} = \mathrm{Out}_{\mathcal{A}}$.*

Proof. Without loss of generality assume that A_R is left-simple. We define $\mathcal{A} = (\Sigma_1, \Sigma_2, A_L, A_R, Q_L, s_L, \delta, \tau, \psi, \psi_\tau)$ where:

- $\delta(q, a, R) = q$,
- $\tau(q, L, R) = L$,
- $\psi_\tau(q, L, R) = \begin{cases} \epsilon & R \neq s_R \\ \iota(L) & R = s_R \end{cases}$.

Let $t = t_1 \dots t_n \in \Sigma_1^*$,

$$((L_i, q_i, \overline{q_i}, R_i, o_i, \overline{o_i}))_{i=0}^n$$

be an execution of A over t and

$$((L_i, o'_i, R_i))_{i=0}^n$$

be an execution of $\mathcal{B}$ over t. Due to the definition of τ, we have $\overline{q_i} = L_i$ for each $0 \leq i \leq n$. Note that, for $0 \leq i < n$,

$$o_{i+1} = \psi(\overline{q_i}, t_{i+1}, R_{i+1}) = \psi(L_i, t_{i+1}, R_{i+1}) = o'_i.$$

We also have that

$$\overline{o_n} = \psi_\tau(q_n, L_n, R_n) = \psi_\tau(q_n, L_n, s_R) = \iota(L_n) = o'_n.$$

Since A_R is left-simple, for each $i < n$, we have $R_i \neq s_R$ and thus $\overline{o_i} = \epsilon$. We can conclude that $\mathrm{Out}_{\mathcal{B}}(t) = \mathrm{Out}_{\mathcal{A}}(t)$ (in particular, both sides of the equality may be undefined). $\square$

Construction 6.6. *Let $(T_r/\lambda_r\text{-}\rho_r)_{r=1}^m$ be a BCRR over (Σ_1, Σ_2). Let $((\mathcal{T}_r, \mathcal{E}_r, A_{\lambda_r}, A_{\rho_r}))_{r=1}^m$ be a FSR of $(T_r/\lambda_r\text{-}\rho_r)_{r=1}^m$ (the input for the construction). The output of this construction is a classical bimachine with final output $\mathcal{B} = (\Sigma_1 \cup \Sigma_2, A_L, A_R, \psi, \iota)$, where the right automaton $A_R = (\Sigma_1, Q_R, s_R, Q_R, \delta_R)$ is exactly the same as in construction 5.3.*

Let A_λ, $\mathcal{T}_r^{\mathrm{dom}}$, $\mathcal{T}^{\mathrm{dom}}$, the set Δ_T and the functions type, *I, ν, J be as in construction 5.3. Let $L \subseteq Q_\lambda$, $(R, g) \in Q_R$ and $\phi : Q_R \nrightarrow Q_T$. We define the following predicates:*

$$\begin{aligned}
\mathrm{NonemptyMatchNotFinished}(\phi, (R, g)) &\overset{\mathrm{def}}{\iff} \phi((R, g)) \downarrow \ \& \ \phi((R, g)) \in \mathrm{Rng}(g) \setminus F_T, \\
\mathrm{NonemptyMatchFinished}(\phi, (R, g)) &\overset{\mathrm{def}}{\iff} \phi((R, g)) \downarrow \ \& \ \phi((R, g)) \in F_T, \\
\mathrm{NonemptyMatchBegin}(L, g) &\overset{\mathrm{def}}{\iff} I(L, g) \neq \emptyset, \\
\mathrm{OutsideOfMatch}(\phi, (R, g)) &\overset{\mathrm{def}}{\iff} \phi((R, g)) \uparrow, \\
\mathrm{EmptyMatchBegin}(L, R) &\overset{\mathrm{def}}{\iff} J(L, R) \neq \emptyset.
\end{aligned}$$

The left automaton is $A_L = (\Sigma_1, Q_L, s_L, Q_L, \delta_L)$ *where:*

- $Q_L \subseteq \mathcal{P}(Q_\lambda) \times \{\phi \mid \phi : Q_R \nrightarrow Q_T\}$,
- $s_L \coloneqq (S_\lambda, \phi_0)$, *where* $\phi_0((R,g)) \coloneqq \begin{cases} \nu(S_\lambda, g) & \text{NonemptyMatchBegin}(S_\lambda, g) \\ \uparrow & \textit{otherwise} \end{cases}$,
- $\delta_L((L,\phi), a) \coloneqq (L', \phi')$, *for* $(L,\phi) \in Q_L$ *and* $a \in \Sigma_1$ *where*
  - $L' \coloneqq \text{Succ}_{A_\lambda}(L, a)$,
  - $$\phi'((R', g')) \coloneqq \begin{cases} g'(\mu(\phi((R,g)), a, g')) & \text{NonemptyMatchNotFinished}(\phi, (R,g)) \\ g'(\mu(\nu(L,g), a, g')) & \begin{array}{l}\text{NonemptyMatchFinished}(\phi, (R,g))\ \&\\ \text{NonemptyMatchBegin}(L,g)\end{array} \\ \nu(L', g') & \begin{array}{l}(\ \text{OutsideOfMatch}(\phi, (R,g)) \vee \\ \text{NonemptyMatchFinished}(\phi, (R,g))\ )\ \& \\ \neg\text{NonemptyMatchBegin}(L,g)\ \& \\ \text{NonemptyMatchBegin}(L', g')\end{array} \\ \uparrow & \textit{otherwise} \end{cases},$$

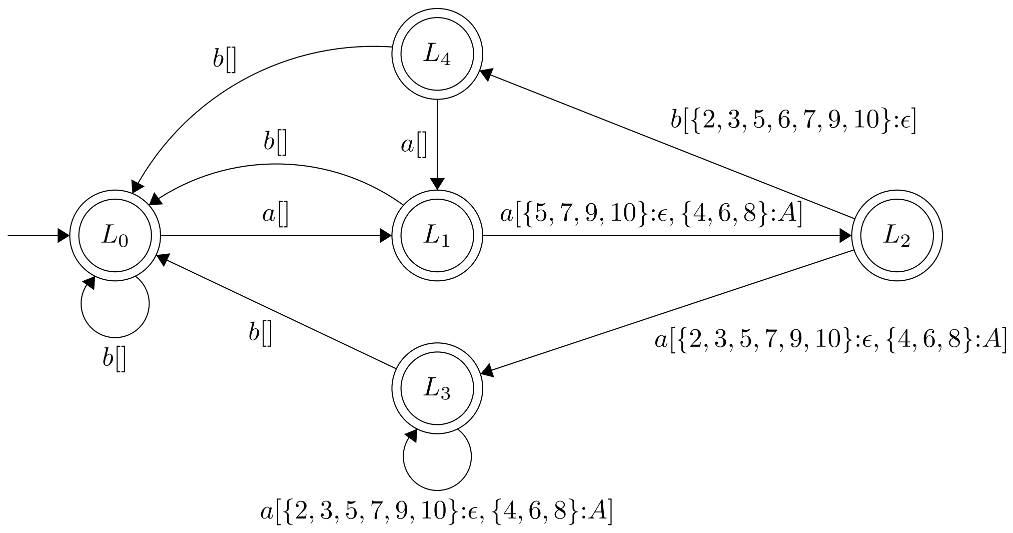
    where
    - $(R, g) = \delta_R((R', g'), a)$,
    - $\mu(q, a, g') = \min\{i' \mid (q, a, g'(i')) \in \Delta_T^{\Sigma_1}\}$.

The output function ψ *is defined as follows:*

$$\psi((L,\phi), a, (R', g')) \coloneqq \begin{cases} \text{Output}(\phi((R,g)), a, \phi'((R',g'))) & \text{NonemptyMatchNotFinished}(\phi, (R,g)) \\ \text{Output}(\nu(L,g), a, \phi'((R',g'))) & \begin{array}{l}\text{NonemptyMatchFinished}(\phi, (R,g))\ \& \\ \text{NonemptyMatchBegin}(L,g)\end{array} \\ \mathcal{E}_{\min(J(L,R))} \cdot a & \begin{array}{l}(\ \text{OutsideOfMatch}(\phi, (R,g)) \vee \\ \text{NonemptyMatchFinished}(\phi, (R,g))\ )\ \& \\ \neg\text{NonemptyMatchBegin}(L,g)\ \& \\ \text{EmptyMatchBegin}(L,R)\end{array} \\ a & \textit{otherwise} \end{cases},$$

where

- $(L', \phi') = \delta_L((L,\phi), a)$,
- $(R, g) = \delta_R((R', g'), a)$,



L_4
$b[]$
$b[\{2,3,5,6,7,9,10\}{:}\epsilon]$
$b[]$
$a[]$
L_0
$a[]$
L_1
$a[\{5,7,9,10\}{:}\epsilon, \{4,6,8\}{:}A]$
L_2
$b[]$
$b[]$
$a[\{2,3,5,7,9,10\}{:}\epsilon, \{4,6,8\}{:}A]$
L_3
$a[\{2,3,5,7,9,10\}{:}\epsilon, \{4,6,8\}{:}A]$

Figure 6.1: The left automaton of the bimachine with final output from example 6.7. The output function ψ is also shown. A transition from L_i into L_j of the form $x[\mathrm{SET}_0{:}y_0, \ldots, \mathrm{SET}_p{:}y_p]$ denotes that

$$\delta_L(L_i, x) = L_j \text{ and } (\forall p' \leq p)(\forall k \in \mathrm{SET}_{p'})\psi(L_i, x, R_k) = y_{p'}.$$

Unless otherwise specified, $\psi(L, x, R) = x$.

- $\mathrm{Output}(q, a, q') = \mathrm{elem\_of}\{m \mid (q, (a, m), q') \in \Delta_T\}$.

  The function ι, producing the final output, is defined as follows:

$$\iota((L, \phi)) \coloneqq \begin{cases} \mathcal{E}_{\min(J(L, S_{RU}))} & \mathrm{EmptyMatchBegin}(L, S_{RU}) \\ \epsilon & \textit{otherwise} \end{cases},$$

where $(S_{RU}, g_s) = s_R$.

Example 6.7. *A possible result of applying construction 6.6 on the FSR $\langle(\mathcal{T}_1, \mathcal{E}_1, A_{\lambda_1}, A_{\rho_1})\rangle$ from example 4.13 is $\mathcal{B} = (\{a, b, A\}, A_L, A_R, \psi, \iota)$ where A_L and ψ are shown in figure 6.1, A_R is the same as in example 5.4 and $\iota(L) = \epsilon$ for all $L \in Q_L$.*

The execution of $\mathcal{B}$ over $t = aaaaaabaaab$ is shown on figure 6.2.

Proposition 6.8. *Construction 6.6 correctly defines a classical bimachine with final output.*

Proof. For the correctness of A_R, see proposition 5.5. Clearly, ϕ_0 and ι are correctly defined.

Let $(L, \phi) \in Q_L$, $a \in \Sigma_1$, $(R', g') \in Q_R$ and $(R, g) = \delta_R((R', g'), a)$. We will show that

$$(L', \phi') = \delta_L((L, \phi), a)$$

$$\begin{tikzcd}
R_{10} & R_{10} \arrow[l, "a"'] & R_{10} \arrow[l, "a"'] & R_{10} \arrow[l, "a"'] & R_7 \arrow[l, "a"'] & R_8 \arrow[l, "a"'] & R_5 \arrow[l, "a"'] & R_{10} \arrow[l, "b"'] & R_7 \arrow[l, "a"'] & R_4 \arrow[l, "a"'] & R_2 \arrow[l, "a"'] & R_0 \arrow[l, "b"'] \\
L_0 \arrow[r, "a", "a"'] & L_1 \arrow[r, "a", "\epsilon"'] & L_2 \arrow[r, "a", "\epsilon"'] & L_3 \arrow[r, "a", "\epsilon"'] & L_3 \arrow[r, "a", "A"'] & L_3 \arrow[r, "a", "\epsilon"'] & L_3 \arrow[r, "b", "b"'] & L_0 \arrow[r, "a", "a"'] & L_1 \arrow[r, "a", "A"'] & L_2 \arrow[r, "a", "\epsilon"'] & L_3 \arrow[r, "b", "b"'] & L_0
\end{tikzcd}$$

Figure 6.2: The execution of $\mathcal{B}$ over $t = aaaaaabaaab$.
$\mathrm{Out}_{\mathcal{B}}(t) = a\epsilon\epsilon\epsilon A\epsilon baA\epsilon b \cdot \iota(L_0) = aAbaAb$.

is correctly defined.

If NonemptyMatchNotFinished$(\phi, (R, g))$, then $\phi((R, g)) \in \mathrm{Rng}(g) \setminus F_T$. We can conclude that $\mu(\phi((R, g)), a, g')$ is defined in exactly the same way as in the proof of the correctness for δ in proposition 5.5. Again, all elements of $\{i' \mid (\phi((R, g)), a, g'(i')) \in \Delta_T^{\Sigma_1}\}$ belong to the domain of g' and in particular $\mu(\phi((R, g)), a, g')$ does. Thus $g'(\mu(\phi((R, g)), a, g'))$ is defined.

If NonemptyMatchBegin(L, g), then $I(L, g) \neq \emptyset$. Let $q = \nu(L, g)$. By the definitions of I and ν, we have $q \in S_T$ and $q \in \mathrm{Rng}(g)$ which implies $q \in \mathrm{Rng}(g) \setminus F_T$. To see that $g'(\mu(\nu(L, g), a, g'))$ is defined, the same reasoning as in the previous case can be used.

Finally, if NonemptyMatchBegin(L', g'), immediately follows that $\nu(L', g')$ is defined. Thus ϕ' is correctly defined.

For ψ, if NonemptyMatchNotFinished$(\phi, (R, g))$, then $(\phi((R, g)), a, \phi((R', g'))) \in \Delta_T^{\Sigma_1}$, so

$$\{m \mid (\phi((R, g)), (a, m), \phi((R', g'))) \in \Delta_T\} \neq \emptyset$$

and hence Output$(\phi((R, g)), a, \phi'((R', g')))$ is defined.

Similarly, if NonemptyMatchFinished$(\phi, (R, g))$&NonemptyMatchBegin(L, g), then

$$\{m \mid (\nu(L, g), (a, m), \phi((R', g'))) \in \Delta_T\} \neq \emptyset$$

and Output$(\nu(L, g), a, \phi'((R', g')))$ is defined. If EmptyMatchBegin(L, R), it follows directly from the definitions that $\mathcal{E}_{\min(J(L,R))}$ is defined. Thus ψ is correctly defined. $\square$

Let $A = (\Sigma_1, \Sigma_2, A_L, A_R, Q, q_{\mathrm{err}}, \delta, \tau, \psi_\delta, \psi_\tau)$ be obtained from construction 5.3 and $\mathcal{B} = (\Sigma_1 \cup \Sigma_2, A'_L, A_R, \psi, \iota)$ be obtained from construction 6.6 applied on the same FSR $((\mathcal{T}_r, \mathcal{E}_r, A_{\lambda_r}, A_{\rho_r}))_{r=1}^m$. We are going to show that A and $\mathcal{B}$ are equivalent, i.e. for each input string $t \in \Sigma_1^*$, both A and $\mathcal{B}$ produce the same output for t. For this purpose, we are going to prove a couple of propositions which show the relationships between the two machines.

Let $t = t_1 \dots t_n \in \Sigma_1^*$,

$$((L_i, q_i, \overline{q_i}, (R_i, g_i), o_i, \overline{o_i}))_{i=0}^n$$

be an execution of A over t and

$$(((L_i, \phi_i), o'_i, (R_i, g_i)))_{i=0}^n$$

be an execution of $\mathcal{B}$ over t. Note that the executions of the right automata of the two bimachines coincide since they both have the same automaton A_R

as right automaton. Similarly, the L_i components in both executions coincide since they are constructed in one and the same way.

The first proposition will show the relationship between the states of A and the "selected" states by the functions ϕ_i.

Proposition 6.9. *Let $0 \leq i \leq n$. All of the following implications hold:*

$$\begin{aligned} q_i \in Q_T \setminus S_T \quad &\Longrightarrow \quad \phi_i((R_i, g_i)) = q_i, && (3) \\ q_i = q_{\text{err}} \ \&\ \overline{q_i} = q_{\text{err}} \quad &\Longrightarrow \quad \phi_i((R_i, g_i)) \uparrow, && (4) \\ q_i \notin F_T \ \&\ \overline{q_i} \in S_T \quad &\Longrightarrow \quad \phi_i((R_i, g_i)) = \overline{q_i}. && (5) \end{aligned}$$

Proof. Assume that the implications hold for all $j < i$.

(3) Assume $q_i \in Q_T \setminus S_T$. Then $i > 0$ ($q_0 = q_{\text{err}} \notin Q_T$) and $\overline{q_{i-1}} \in \text{Rng}(g_{i-1}) \setminus F_T$.

Case 1: $\overline{q_{i-1}} \notin S_T$. Then $\overline{q_{i-1}} = q_{i-1}$ and hence $q_{i-1} \in Q_T \setminus S_T$. By the i.h. for (3), $\phi_{i-1}((R_{i-1}, g_{i-1})) = q_{i-1}$. Therefore $\phi_{i-1}((R_{i-1}, g_{i-1})) \downarrow$ and $\phi_{i-1}((R_{i-1}, g_{i-1})) \in \text{Rng}(g_{i-1}) \setminus F_T$, i.e.

$$\text{NonemptyMatchNotFinished}(\phi_{i-1}, (R_{i-1}, g_{i-1})).$$

We conclude that

$$\begin{aligned} \phi_i((R_i, g_i)) &= g_i(\mu(\phi_{i-1}((R_{i-1}, g_{i-1})), t_i, g_i)) \\ &= g_i(\mu(\overline{q_{i-1}}, t_i, g_i)) = q_i. \end{aligned}$$

Case 2: $\overline{q_{i-1}} \in S_T$. Then $q_{i-1} \notin Q_T \setminus F_T$, $I(L_{i-1}, g_{i-1}) \neq \emptyset$ and $\overline{q_{i-1}} = \nu(L_{i-1}, g_{i-1})$. Therefore in this case we have $\text{NonemptyMatchBegin}(L_{i-1}, g_{i-1})$.

Case 2.1: $q_{i-1} \in F_T$. But since $F_T \subseteq Q_T \setminus S_T$, by i.h. for (3) we have $\phi_{i-1}((R_{i-1}, g_{i-1})) = q_{i-1}$. Thus $\text{NonemptyMatchFinished}(\phi_{i-1}, (R_{i-1}, g_{i-1}))$ holds. We conclude that

$$\begin{aligned} \phi_i((R_i, g_i)) &= g_i(\mu(\nu(L_{i-1}, g_{i-1}), t_i, g_i)) \\ &= g_i(\mu(\overline{q_{i-1}}, t_i, g_i)) = q_i. \end{aligned}$$

Case 2.2: $q_{i-1} \notin F_T$. Then $q_{i-1} = q_{\text{err}}$. By i.h. for (5), $\phi_{i-1}((R_{i-1}, g_{i-1})) = \overline{q_{i-1}}$. We have $\overline{q_{i-1}} \in S_T \subseteq Q_T$, so by proposition 5.6 we get $\overline{q_{i-1}} \in \text{Rng}(g_{i-1}) \setminus F_T$. Therefore, as in case 1, we have $\text{NonemptyMatchNotFinished}(\phi_{i-1}, (R_{i-1}, g_{i-1}))$ and $\phi_i((R_i, g_i)) = q_i$.

(4) Assume $q_i = \overline{q_i} = q_{\text{err}}$. Then $I(L_i, g_i) = \emptyset$, i.e. $\neg\text{NonemptyMatchBegin}(L_i, g_i)$.

Case 1: $i = 0$. Then $L_i = L_0 = S_\lambda$, so $\neg\text{NonemptyMatchBegin}(S_\lambda, g_i)$ and hence $\phi_i((R_i, g_i)) \uparrow$.

Case 2: $i > 0$. Then $q_i = q_{\text{err}}$ implies $\overline{q_{i-1}} = \tau(q_{i-1}, L_{i-1}, (R_{i-1}, g_{i-1})) = q_{\text{err}}$. Therefore $q_{i-1} \notin Q_T \setminus F_T$ and $I(L_{i-1}, g_{i-1}) = \emptyset$, i.e.

$$\neg\text{NonemptyMatchBegin}(L_{i-1}, g_{i-1}).$$

Case 2.1: $q_{i-1} \in F_T$. Then, since $F_T \subseteq Q_T \setminus S_T$, by the i.h. for (3), $\phi_{i-1}((R_{i-1}, g_{i-1})) = q_{i-1} \notin \text{Rng}(g_{i-1}) \setminus F_T$. Therefore

$$\neg\text{NonemptyMatchNotFinished}(\phi_{i-1}, (R_{i-1}, g_{i-1})).$$

Case 2.2: $q_{i-1} \notin F_T$. Then $q_{i-1} = q_{\text{err}}$. By the i.h. for (4), $\phi_{i-1}((R_{i-1}, g_{i-1}))\uparrow$. Therefore again

$$\neg\text{NonemptyMatchNotFinished}(\phi_{i-1}, (R_{i-1}, g_{i-1})).$$

We have shown that none of the three conditions for $\phi_i((R_i, g_i))$ to be defined is satisfied, so $\phi_i((R_i, g_i))\uparrow$.

(5) Assume $q_i \notin F_T$ and $\overline{q_i} \in S_T$. Then $q_i \notin Q_T \setminus F_T$, $I(L_i, g_i) \neq \emptyset$ and $\overline{q_i} = \nu(L_i, g_i)$. Therefore in this case we have $\text{NonemptyMatchBegin}(L_i, g_i)$. Since $q_i \notin F_T$, we have $q_i = q_{\text{err}}$.

Case 1: $i = 0$. Then $L_i = L_0 = S_\lambda$, so $\text{NonemptyMatchBegin}(S_\lambda, g_i)$ and hence

$$\phi_i((R_i, g_i)) = \nu(S_\lambda, g_i) = \nu(L_i, g_i) = \overline{q_i}.$$

Case 2: $i > 0$. In this case $\overline{q_{i-1}} = q_{\text{err}}$. Therefore $q_{i-1} \notin Q_T \setminus F_T$ and $I(L_{i-1}, g_{i-1}) = \emptyset$, i.e. $\neg\text{NonemptyMatchBegin}(L_{i-1}, g_{i-1})$.

Case 2.1: $q_{i-1} \in F_T$. Then, since $F_T \subseteq Q_T \setminus S_T$, by the i.h. for (3) we get $\phi_{i-1}((R_{i-1}, g_{i-1})) = q_{i-1}$. Thus we have

$$\text{NonemptyMatchFinished}(\phi_{i-1}, (R_{i-1}, g_{i-1})).$$

Case 2.2: $q_{i-1} \notin F_T$. Then $q_{i-1} = q_{\text{err}}$. By the i.h. for (4), $\phi_{i-1}((R_{i-1}, g_{i-1}))\uparrow$. Thus we have

$$\text{OutsideOfMatch}(\phi_{i-1}, (R_{i-1}, g_{i-1})).$$

In either case $\phi_i((R_i, g_i)) = \nu(L_i, g_i) = \overline{q_i}$.

□

The next two proposition show the relationship between the outputs of the two bimachines at any particular step.

Proposition 6.10. *Let $0 \leq i < n$. Then $\overline{o_i} \cdot o_{i+1} = o'_i$.*

Proof. In the following proof we will tacitly use the result of proposition 5.19.

Case 1: $q_{i+1} \in Q_T$. Then $\overline{q_i} \in \mathrm{Rng}(g_i) \setminus F_T$ and $o_{i+1} = \mathrm{Output}(\overline{q_i}, t_{i+1}, q_{i+1})$. Note that $q_{i+1} \notin S_T$, so we can apply (3) to get $\phi_{i+1}((R_{i+1}, g_{i+1})) = q_{i+1}$. We have $\tau(q_i, L_i, (R_i, g_i)) = \overline{q_i} \neq q_{\mathrm{err}}$, so $\overline{o_i} = \epsilon$.

Case 1.1: $\overline{q_i} \notin S_T$. Then $\overline{q_i} = q_i \in Q_T \setminus S_T$. By (3), $\phi_i((R_i, g_i)) = q_i$. Note that $\overline{q_i} \notin F_T$ and $\overline{q_i} = q_i$, so $\phi_i((R_i, g_i)) \in Q_T \setminus F_T$, that is

$$\mathrm{NonemptyMatchNotFinished}(\phi_i, (R_i, g_i)).$$

Thus

$$\begin{aligned} o'_i &= \psi(L_i, t_{i+1}, (R_{i+1}, g_{i+1})) \\ &= \mathrm{Output}(\phi_i((R_i, g_i)), t_{i+1}, \phi_{i+1}((R_{i+1}, g_{i+1}))) \\ &= \mathrm{Output}(q_i, t_{i+1}, q_{i+1}) \\ &= \mathrm{Output}(\overline{q_i}, t_{i+1}, q_{i+1}) = o_{i+1} = \overline{o_i} \cdot o_{i+1}. \end{aligned}$$

Case 1.2: $\overline{q_i} \in S_T$. Then $q_i \notin Q_T \setminus F_T$, $I(L_i, g_i) \neq \emptyset$ and $\overline{q_i} = \nu(L_i, g_i)$. Therefore in this case we have $\mathrm{NonemptyMatchBegin}(L_i, g_i)$.

Case 1.2.1: $q_i \in F_T$. Then $q_i \in Q_T \setminus S_T$ and by (3) we have $\phi_i((R_i, g_i)) = q_i \in F_T$. This means

$$\mathrm{NonemptyMatchFinished}(\phi_i, (R_i, g_i))$$

and thus

$$\begin{aligned} o'_i &= \mathrm{Output}(\nu(L_i, g_i), t_{i+1}, \phi_{i+1}((R_{i+1}, g_{i+1}))) \\ &= \mathrm{Output}(\overline{q_i}, t_{i+1}, q_{i+1}) = o_{i+1} = \overline{o_i} \cdot o_{i+1}. \end{aligned}$$

Case 1.2.2: $q_i \notin F_T$. Then, by (5), $\phi_i((R_i, g_i)) = \overline{q_i} \in S_T \subseteq Q_T$. By proposition 5.6, $\overline{q_i} \in \mathrm{Rng}(g_i) \setminus F_T$, so $\phi_i((R_i, g_i)) \in \mathrm{Rng}(g_i) \setminus F_T$. Therefore, we have

$$\mathrm{NonemptyMatchNotFinished}(\phi_i, (R_i, g_i))$$

and thus

$$\begin{aligned} o'_i &= \psi(L_i, t_{i+1}, (R_{i+1}, g_{i+1})) \\ &= \mathrm{Output}(\phi_i((R_i, g_i)), t_{i+1}, \phi_{i+1}((R_{i+1}, g_{i+1}))) \\ &= \mathrm{Output}(\overline{q_i}, t_{i+1}, q_{i+1}) = o_{i+1} = \overline{o_i} \cdot o_{i+1}. \end{aligned}$$

Case 2: $q_{i+1} \notin Q_T$. Then $q_{i+1} = q_{\mathrm{err}}$ and $o_{i+1} = t_{i+1}$. We also have that $\overline{q_i} = q_{\mathrm{err}}$, so $q_i \notin Q_T \setminus F_T$ and $I(L_i, g_i) = \emptyset$, i.e.

$$\neg\mathrm{NonemptyMatchBegin}(L_i, g_i).$$

If $q_i \in F_T$, then (by (3)) $q_i = \phi_i((R_i, g_i)) \in F_T$, that is

$$\mathrm{NonemptyMatchFinished}(\phi_i, (R_i, g_i))$$

and

$$\neg\text{NonemptyMatchNotFinished}(\phi_i, (R_i, g_i)).$$

Otherwise $q_i = q_{\text{err}}$ and (by (4)) $\phi_i((R_i, g_i)) \uparrow$. In either case we have

$$\neg\text{NonemptyMatchNotFinished}(\phi_i, (R_i, g_i))$$

and

$$\text{OutsideOfMatch}(\phi_i, (R_i, g_i)) \vee \text{NonemptyMatchFinished}(\phi_i, (R_i, g_i)).$$

Case 2.1: $J(L_i, R_i) \neq \emptyset$ (EmptyMatchBegin(L_i, R_i)). Then, since $\overline{q_i} = q_{\text{err}}$, we have $\overline{o_i} = \mathcal{E}_{\min(J(L_i,R_i))}$. In this case

$$o'_i = \mathcal{E}_{\min(J(L_i,R_i))} \cdot t_{i+1} = \overline{o_i} \cdot o_{i+1}.$$

Case 2.2: $J(L_i, R_i) = \emptyset$ ($\neg$EmptyMatchBegin(L_i, R_i)). Then $\overline{o_i} = \epsilon$. In this case

$$o'_i = t_{i+1} = o_{i+1} = \overline{o_i} \cdot o_{i+1}.$$

□

Proposition 6.11. *$\overline{o_n} = o'_n$.*

Proof. By proposition 5.7, $\tau(q_n, L_n, (R_n, g_n)) = \overline{q_n} = q_{\text{err}}$. By definition, $(R_n, g_n) = s_R = (S_{RU}, g_s)$. Therefore, if $J(L_n, R_n) \neq \emptyset$, then

$$\overline{o_n} = \mathcal{E}_{\min(J(L_n,R_n))} = \mathcal{E}_{\min(J(L_n,S_{RU}))} = o'_n.$$

Otherwise $\overline{o_n} = o'_n = \epsilon$. □

We are now ready to show that A and $\mathcal{B}$ are equivalent.

Theorem 6.12. *$\text{Out}_A(t) = \text{Out}_{\mathcal{B}}(t)$.*

Proof. Applying proposition 6.10 n times gives

$$\overline{o_0}o_1\overline{o_1}o_2 \ldots \overline{o_{n-1}}o_n = o'_0 o'_1 \ldots o'_{n-1}.$$

Finally, we use proposition 6.11 and the fact that $o_0 = \epsilon$ by definition to get

$$\text{Out}_A(t) = o_0\overline{o_0}o_1\overline{o_1}o_2 \ldots \overline{o_{n-1}}o_n\overline{o_n} = o'_0 o'_1 \ldots o'_{n-1}o'_n = \text{Out}_{\mathcal{B}}(t)$$

which completes the proof. □

7 Complexity analysis

Proposition 7.1. *Let Q be a finite set with $|Q| \geq 2$. The number of the enumerations of subsets of Q is less than $3\,|Q|!$.*

Proof. Let $n = |Q|$. The number of the subsets of Q can be expressed as

$$\sum_{k=0}^{n} \binom{n}{k}.$$

A subset of k element has $k!$ distinct enumerations (the permutations of the elements), so the total number of enumerations is

$$\sum_{k=0}^{n} k! \binom{n}{k}.$$

Using our assumption that $|Q| \geq 2$ and the fact that $(n-k)! \geq 2^{n-k}$ for $k \leq n-2$ we obtain an upper bound for this total number of enumerations:

$$\begin{aligned}\sum_{k=0}^{n} k! \binom{n}{k} &= \sum_{k=0}^{n} \frac{n!}{(n-k)!} = 2n! + \sum_{k=0}^{n-2} \frac{n!}{(n-k)!} \\ &\leq 2n! + \sum_{k=0}^{n-2} \frac{n!}{2^{n-k}} = 2n! + n! \sum_{p=2}^{n} \frac{1}{2^p} < 3n! \in O(n!).\end{aligned}$$

□

Proposition 7.2. *Let $A = (\Sigma_1, \Sigma_2, A_L, A_R, Q, q_{\text{err}}, \delta, \tau, \psi_\delta, \psi_\tau)$ be obtained from construction 5.3 applied on some FSR $((\mathcal{T}_r, \mathcal{E}_r, A_{\lambda_r}, A_{\rho_r}))_{r=1}^{m}$. Let $Q_\rho = \bigcup_{r=1}^{m} Q_{\rho_r}$. Then*

$$\begin{aligned}|Q_L| &\leq 2^{|Q_\lambda|}, \\ |\delta_L| &= |Q_L| \cdot |\Sigma_1| \leq 2^{|Q_\lambda|} \cdot |\Sigma_1|, \\ |Q_R| &\leq 2^{|Q_\rho|} \cdot 3\,|Q_T|! \in O(2^{|Q_\rho|} \cdot |Q_T|!), \\ |\delta_R| &= |Q_R| \cdot |\Sigma_1| \leq 2^{|Q_\rho|} \cdot 3\,|Q_T|! \cdot |\Sigma_1|, \\ |\delta| = |\psi_\delta| &\leq |\Sigma_1| \cdot 2^{|Q_\rho|} \cdot 3(|Q_T|+1)! \in O(|\Sigma_1| \cdot 2^{|Q_\rho|} \cdot (|Q_T|+1)!), \\ |\tau| = |\psi_\tau| &\leq 2^{|Q_\lambda|} \cdot 2^{|Q_\rho|} \cdot 3(|Q_T|+1)! \in O(2^{|Q_\lambda|+|Q_\rho|} \cdot (|Q_T|+1)!), \\ \text{Rng}(\delta) &\leq |Q_T| + 1, \\ \text{Rng}(\tau) &\leq |Q_T| + 1.\end{aligned}$$

The total size of A is

$$\begin{aligned}& |\delta_L| + |\delta_R| + |\delta| + |\psi_\delta| + |\tau| + |\psi_\tau| \\ \leq\ & (2^{|Q_\lambda|} + 2^{|Q_\rho|} \cdot 3\,|Q_T|!) \cdot |\Sigma_1| + 6 \cdot 2^{|Q_\rho|} \cdot (|Q_T|+1)! \cdot (2^{|Q_\lambda|} + |\Sigma_1|) \\ \in\ & O(2^{|Q_\rho|} \cdot (|Q_T|+1)! \cdot (2^{|Q_\lambda|} + |\Sigma_1|) + 2^{|Q_\lambda|} \cdot |\Sigma_1|).\end{aligned}$$

Proof. The number of states of A_λ is

$$|Q_\lambda| = \sum_{r=1}^{m} |Q_{\lambda_r}| \,.$$

The number of states of A_L is at most $2^{|Q_\lambda|}$.

Assuming $|Q_T| \geq 2$ (which is always the case unless the batch of rules is empty), an obvious upper bound for the number of states of A_R is

$$|Q_R| \leq 2^{|Q_{RU}|} \sum_{k=0}^{|Q_T|} k! \binom{|Q_T|}{k} \leq 2^{|Q_{RU}|} \cdot 3\, |Q_T|! \in O(2^{|Q_{RU}|} \cdot |Q_T|!).$$

If we look more carefully, we can see that for each $g \in \bigcup_{Q \subseteq Q_T} \mathrm{Enum}(Q)$, the number of pairs (R, g) which may belong to Q_R is no greater than $2^{|Q_\rho|}$. This is because, for any $(R, g) \in Q_R$, the states of Q_T which belong to R are exactly the states in $\mathrm{Rng}(g)$. This observation gives us a more precise upper bound for the states of Q_R:

$$|Q_R| \leq 2^{|Q_\rho|} \sum_{k=0}^{|Q_T|} k! \binom{|Q_T|}{k} \leq 2^{|Q_\rho|} \cdot 3\, |Q_T|! \in O(2^{|Q_\rho|} \cdot |Q_T|!).$$

For the transition functions and the output functions of A we have the following upper bounds:

$$\begin{aligned}
|\delta| = |\psi_\delta| &= (|Q_T|+1) \cdot |\Sigma_1| \cdot |Q_R| \\
&\leq (|Q_T|+1) \cdot |\Sigma_1| \cdot 2^{|Q_\rho|} \cdot 3\, |Q_T|! \in O(|\Sigma_1| \cdot 2^{|Q_\rho|} \cdot (|Q_T|+1)!), \\
|\tau| = |\psi_\tau| &= (|Q_T|+1) \cdot |Q_L| \cdot |Q_R| \\
&\leq (|Q_T|+1) \cdot 2^{|Q_\lambda|} \cdot 2^{|Q_\rho|} \cdot 3\, |Q_T|! \in O(2^{|Q_\lambda|+|Q_\rho|} \cdot (|Q_T|+1)!).
\end{aligned}$$

□

Proposition 7.3. *Let $\mathcal{B} = (\Sigma_1 \cup \Sigma_2, A_L, A_R, \psi, \iota)$ be obtained from construction 6.6 applied on some FSR $((\mathcal{T}_r, \mathcal{E}_r, A_{\lambda_r}, A_{\rho_r}))_{r=1}^m$. Let $Q_\rho = \bigcup_{r=1}^m Q_{\rho_r}$. Then*

$$\begin{aligned}
|Q_R| &\leq 2^{|Q_\rho|} \cdot 3\, |Q_T|! \in O(2^{|Q_\rho|} \cdot |Q_T|!), \\
|\delta_R| &= |Q_R| \cdot |\Sigma_1| \leq 2^{|Q_\rho|} \cdot 3\, |Q_T|! \cdot |\Sigma_1| \,, \\
|\iota| = |Q_L| &\leq 2^{|Q_\lambda|} \cdot (|Q_T|+1)^{3|Q_T|!} \in O(2^{|Q_\lambda|} \cdot |Q_T|^{3|Q_T|!}), \\
|\delta_L| &= |Q_L| \cdot |\Sigma_1| \leq 2^{|Q_\lambda|} \cdot (|Q_T|+1)^{3|Q_T|!} \cdot |\Sigma_1| \,, \\
|\psi| &\leq 2^{|Q_\lambda|+|Q_\rho|} \cdot (|Q_T|+1)^{3|Q_T|!} \cdot |\Sigma_1| \cdot 3\, |Q_T|! \\
&\in O(2^{|Q_\lambda|+|Q_\rho|} \cdot |Q_T|^{3|Q_T|!} \cdot |\Sigma_1| \cdot |Q_T|!).
\end{aligned}$$

The total size of $\mathcal{B}$ is

$$\begin{aligned} & |\delta_R| + |\delta_L| + |\iota| + |\psi| \\ \leq\ & 2^{|Q_\rho|} \cdot 3\,|Q_T|! \cdot |\Sigma_1| + 2^{|Q_\lambda|} \cdot (|Q_T|+1)^{3|Q_T|!} \cdot (|\Sigma_1|+1) + 2^{|Q_\lambda|+|Q_\rho|} \cdot (|Q_T|+1)^{3|Q_T|!} \cdot |\Sigma_1| \cdot 3\,|Q_T|! \\ =\ & 2^{|Q_\rho|} \cdot 3\,|Q_T|! \cdot |\Sigma_1| + 2^{|Q_\lambda|} \cdot (|Q_T|+1)^{3|Q_T|!} \cdot (|\Sigma_1| + 1 + 2^{|Q_\rho|} \cdot |\Sigma_1| \cdot 3\,|Q_T|!) \\ \in\ & O(2^{|Q_\lambda|+|Q_\rho|} \cdot |Q_T|^{3|Q_T|!} \cdot |\Sigma_1| \cdot |Q_T|!). \end{aligned}$$

Proof. The number of states of A_R are as in proposition 7.2. The number of subsets of Q_λ is $2^{|Q_\lambda|}$ and the number of the partial functions $\phi : Q_R \nrightarrow Q_T$ is $(|Q_T|+1)^{|Q_R|}$. Note that by the construction, if $(L, \phi) \in Q_L$ and $(R, g) \in Q_R$, then $\phi((R, g))$ only depends on the enumeration g and does not depend on the states of Q_ρ in R, i.e. $\phi((R', g)) = \phi((R, g))$ for any $(R', g) \in Q_R$. This means for any fixed L, the number of functions ϕ such that $(L, \phi) \in Q_L$ is no greater than $(|Q_T|+1)^{3|Q_T|!}$. This is because the number of the enumerations of subsets of Q_T are less than $3\,|Q_T|!$. Therefore, we have the following upper bound for the number of states of A_L:

$$|Q_L| \quad \leq \quad 2^{|Q_\lambda|} \cdot (|Q_T|+1)^{3|Q_T|!} \in O(2^{|Q_\lambda|} \cdot |Q_T|^{3|Q_T|!}).$$

For the output function we have

$$\begin{aligned} |\psi| \quad &= \quad |Q_L| \cdot |\Sigma_1| \cdot |Q_R| \\ &\leq \quad 2^{|Q_\lambda|} \cdot (|Q_T|+1)^{3|Q_T|!} \cdot |\Sigma_1| \cdot 2^{|Q_\rho|} \cdot 3\,|Q_T|! \\ &\in \quad O(2^{|Q_\lambda|+|Q_\rho|} \cdot |Q_T|^{3|Q_T|!} \cdot |\Sigma_1| \cdot |Q_T|!). \end{aligned}$$

□

Remark 7.4. *Since for any $q \in Q$, $a \in \Sigma_1$, $(R', g'), (R'', g'') \in Q_R$,*

$$g' = g'' \implies \delta(q, a, (R', g')) = \delta(q, a, (R'', g'')),$$

one may store a function $\tilde{\delta} : Q \times \Sigma_1 \times (Q_R/{\sim_R})$ instead of δ, where ${\sim_R} \subseteq Q_R \times Q_R$ is defined as

$$(R', g') \sim_R (R'', g'') \iff g' = g''$$

and

$$\tilde{\delta}(q, a, [(R, g)]_{\sim_R}) = \delta(q, a, (R, g)).$$

Clearly, this correctly defines a function $\tilde{\delta}$ due to the definition of $\sim_R$ and the observation above. The size of this function is

$$\left|\tilde{\delta}\right| = (|Q_T|+1) \cdot |\Sigma_1| \cdot |Q_R/{\sim_R}| \in O(|\Sigma_1| \cdot (|Q_T|+1)!),$$

thus eliminating the factor $2^{|Q_\rho|}$. Note that by using this, one may need to store the projection $(R, g) \mapsto g$ for using it during the rewriting process. Same idea can be applied for the functions ψ_δ, τ, ψ_τ and ψ.

Similarly, since for any $q \in Q$, $L', L'' \in Q_L$, $R \in Q_R$,

$$L' \cap F_\lambda = L'' \cap F_\lambda \implies \tau(q, L', R) = \tau(q, L'', R),$$

where $F_\lambda = \{f_{\lambda_1}, \dots, f_{\lambda_r}\}$, one may store a function $\tilde{\tau} : Q \times (Q_L/ \sim_L) \times Q_R$ instead of τ, where $\sim_L \subseteq Q_L \times Q_L$ is defined as

$$L' \sim_L L'' \iff L' \cap F_\lambda = L'' \cap F_\lambda$$

and

$$\tilde{\tau}(q, [L]_{\sim_L}, R) = \tau(q, L, R).$$

Again, it is clear that this correctly defines a function $\tilde{\tau}$. The size of this function is

$$|\tilde{\tau}| = (|Q_T| + 1) \cdot |Q_L/{\sim_L}| \cdot |Q_R| \in O(2^{m+|Q_\rho|} \cdot (|Q_T| + 1)!),$$

thus replacing the factor $2^{|Q_\lambda|}$ with 2^m. Note that by using this, one may need to store the projection $L \mapsto L \cap F_\lambda$ for using it during the rewriting process. Same idea can be applied for the functions ψ_τ and ψ.

Of course, both optimization can be applied simultaneously in the obvious way for the functions τ, ψ_τ and ψ.

Remark 7.5. *It is not an easy task to see whether the presented upper bounds are least. It is known that for every functional real-time transducer with n states there is an equivalent bimachine which has at most $O(2^n)$ states [4]. However, we do not work with the functional transducer for the leftmost-longest replacement explicitly but only with the transducers for the functions of each individual rule. As can be seen in [1], the regular function for the leftmost-longest replacement and hence the corresponding transducer depend quite non-trivially on the rules. If we can find the exact upper bound for the size of this transducer, we may also be able to analyze more precisely the sizes of the bimachines based on the known results. Unfortunately, we have not made any progress in solving this so it remains an open problem.*

8 Implementation

Despite the upper bounds given in section 7, it is still very hard to predict how these constructions will perform in practice. The sizes of the obtained machines vary greatly depending on the input batch of rules. In this section we are going to compare both constructions empirically to get an idea of the strengths and the weaknesses of each of them.

In our experiment we are going to use the Porter stemming algorithm [5], which finds application in information retrieval. It was chosen not only because it is complex enough for the task, but also as a demonstration of the usefulness of these constructions for practical purposes.

As part of this work, the described constructions were implemented using standard C++23. The implementation is briefly described in subsection 8.1.

Then, in subsection 8.2, the Porter stemmer is introduced and it is shown how the algorithm is adapted for our experiment. Finally, the results and conclusions are provided in subsection 8.3.

8.1 Implementation overview

The two main constructions presented in this thesis are implemented in C++ using object-oriented approach (though, some of the good OOP practices may have been sacrificed in favor of performance). The program consists of several classes, which are used to implement the two main constructions presented in this thesis. The input batch of rules is represented by regular expressions, which are converted into finite-state machines using the function template `regexToMFSA`. Then, a FSR is constructed for this batch of rules, which in turn is used as an input for the TSBM/BMFO construction. The source code can be found at `https://github.com/StanimirSP/Masters-project`.

MonoidalFSA. The class template `MonoidalFSA` implements a monoidal finite-state automaton in which the operation is concatenation. It contains several public member functions, which provide the basic functionalities needed for these devices such as `Union`, `Concatenation`, `KleeneStar`, `removeEpsilon`[5], `trim`[6], `pseudoDeterm`[7], etc. The public member function `pseudoMinimize` transforms the monoidal FSA into a pseudo-minimal[8] one using the Hopcroft's minimization algorithm. The algorithm was modified to support minimization of colored deterministic automata[9]. This is provided by the method `coloredPseudoMinimize` and will be used for pseudo-minimizing bimachines.

ClassicalFSA. The class `ClassicalFSA` derives from `MonoidalFSA` and adds functionalities specific to classical automata (e.g. intersection and complement of automata).

Transducer. Similarly, the class template `Transducer` derives from `MonoidalFSA` and adds functionalities specific to classical transducers (e.g. composition, transforming to letter or real-time transducer, projections).

RegularExpression. The class template `RegularExpression` represents a generalized regular expression such that it can be used for creating either a classical automaton or a transducer, depending on its type. This is done by converting the regular expression into reverse Polish notation (also known as postfix notation) and then using the Thompson's construction to obtain an automaton or a transducer.

[5]transforms to e-free automaton, see definition 4.2
[6]see [2, definition 2.5.1]
[7]pseudo-determinization, see [2, definition 3.7.1]
[8]see [2, definition 3.7.3]
[9]see [2, section 3.6]

ContextualReplacementRuleRepresentation. The class `ContextualRe⌋ placementRuleRepresentation` constructs the FSR of a contextual replacement rule. An array of `ContextualReplacementRuleRepresentation` is used to represent a FSR of a batch of rules.

TSBM_LeftAutomaton, TSBM_RightAutomaton. The classes `TSBM_LeftAutomaton` and `TSBM_RightAutomaton` represent the left and the right deterministic automaton respectively of a two-step bimachine. `TSBM_RightAutomaton` is reused during the construction of bimachine with final output since the right automaton of both types of bimachines are constructed exactly the same way. `TSBM_LeftAutomaton` is also reused as a helper for the construction of the left automaton of a BMFO.

TwostepBimachine. The class `TwostepBimachine` provides the main functionality of constructing a two-step bimachine given a FSR of batch of rules. The functions δ, ψ_δ, τ and ψ_τ are represented by hash maps in order to have $O(1)$ lookup in the average case and hence achieve rewriting in linear time. The functions are slightly compressed by having a default value for each of them. If a tuple of arguments is mapped to the default value, this entry is not stored. The functions τ and ψ_τ are implemented as a functions of two arguments only (the states of the left automaton and the right automaton). The dependence on the bimachine state q is trivial and can easily be separated in order to achieve better space efficiency. This, however, leads to the need of storing the set F_T even after the construction process has completed and that is why it is included in table 8.2. The construction is optimized by factorizing the states of A_L based on the final states they contain and the states of A_R based on their enumeration part (see remark 7.4). This makes the construction faster by considering indistinguishable (with respect to the four functions) states only once. Additionally, the two-step bimachines are pseudo-minimized using a modified version of the algorithm described in [2, section 6.3].

BimachineWithFinalOutput. The class `BimachineWithFinalOutput` provides the main functionality of constructing a bimachine with final output given a FSR of batch of rules. The optimizations used in this class during the construction are very similar to these in `TwostepBimachine`.

8.2 Porter stemmer and its representation with contextual replacement rules

The purpose of this subsection is to introduce the reader to the Porter stemming algorithm [5] and the way we transformed its rules to fit into our model. The next couple of paragraphs summarize the original paper.

Automatic suffix stripping is especially useful in the field of information retrieval (IR). In an IR system, documents are typically represented by the words in their titles and possibly in their abstracts. When multiple words share a common stem, such as

CONNECT,
CONNECTED,
CONNECTING,
CONNECTION,
CONNECTIONS,

they often have similar meanings. By stripping suffixes like ”-ED”, ”-ING”, ”-ION”, and ”-IONS”, these words can be merged into a single word ”CONNECT”, enhancing the performance of the IR system.

Various techniques for suffix stripping have been reported in literature. The effectiveness of these techniques depends on different factors like whether a stem dictionary or a suffix lists is used, and of course on the purpose for which the suffix stripping is being done.

In cases where a stem dictionary is not used and the aim is to enhance IR performance, a suffix stripping program is typically given an explicit list of suffixes, and, with each suffix, the criterion under which it may be removed from a word to leave a valid stem. This is the approach adopted in the Porter stemming algorithm.

It is important to note that the algorithm presented by Porter is designed simply to improve IR performance, and not as a linguistic exercise. In his paper he states that it would not be at all obvious under what circumstances a suffix should be removed, even if the suffixes of a word could be determined by automatic means. The algorithm therefore is not linguistically correct and it conflates together terms which may have quite distinct meanings (e.g. RELATE and RELATIVITY in a context of theoretical physics). It may also fail to conflate together terms which have similar meanings.

Originally, the algorithm was implemented by its author in a short BCPL program. It consists of a series of rules grouped in 5 steps. Some of the steps are further split into smaller substeps. The rules of the algorithm are copied into appendix A in their original form. Each word from an input dictionary goes through each step sequentially and the final result is the stripped word.

In our program, the steps of the algorithm were implemented using the notion of contextual replacement rule. Since in the original paper the rules are described in non-regular manner, some transformations were made to them in order to be expressed as contextual replacement rules. The result is a sequence of 9 batches of rules (see appendix B or `PorterStemmer.hpp`). A dictionary of words should be sequentially rewritten according to each of them to get the final result.

Each of these batches can be represented either by a classical bimachine or by a two-step bimachine. Then the machines can be used for rewriting the dictionary automatically. These 9 machines can be further composed into one, but this goes out of the scope of this thesis.

step	$\|Q_L\|$		$\|\delta_L\|$		$\|Q_R\|$		$\|\delta_R\|$		$\|\psi\|$		total size / bytes	
		min.		min.		min.		min.		min.		**min.**
#1	17	12	561	396	8	3	264	99	332	69	18 381	**6732**
#2	16	10	528	330	8	5	264	165	29	18	8085	**5049**
#3	104	57	3432	1881	26	3	858	99	115	106	42 405	**21 318**
#4	4	3	132	99	3	2	99	66	2	1	2145	**1518**
#5	179	55	5907	1815	68	20	2244	660	88	49	76 263	**23 892**
#6	29	21	957	693	30	15	990	495	101	75	20 856	**13 167**
#7	140	23	4620	759	43	19	1419	627	1679	238	109 758	**20 328**
#8	15	10	495	330	3	2	99	66	4	1	5478	**3597**
#9	8	6	264	198	5	3	165	99	8	7	4125	**2904**
											287 496	**98 505**

Table 8.1: Size of the constructed bimachines with final output for Porter stemming algorithm *before* *after* pseudo-minimization. The function ι is omitted because it is empty at every step. Data structures overhead is not included in the total size.

step	$\|Q_L\|$	$\|\delta_L\|$	$\|Q_R\|$	$\|\delta_R\|$	$\|\delta\|$	$\|\psi_\delta\|$	$\|\tau\|$	$\|\psi_\tau\|$	$\|F_T\|$	**total size / bytes**
#1	4	132	8	264	14	6	13	1	5	**4152**
#2	4	132	8	264	15	8	8	0	4	**4135**
#3	5	165	26	858	68	25	25	0	6	**11 240**
#4	3	99	3	99	2	2	1	0	1	**1890**
#5	4	132	68	2244	256	56	84	0	42	**27 736**
#6	4	132	30	990	38	26	7	0	7	**11 562**
#7	7	231	43	1419	158	79	133	0	38	**21 259**
#8	13	429	3	99	4	4	3	0	2	**4980**
#9	6	198	5	165	6	6	7	0	2	**3635**
										90 589

Table 8.2: Size of the constructed two-step bimachines for Porter stemming algorithm. Data structures overhead is not included in the total size.

step	construction time / ms			rewriting time / ms	
	FSR	TSBM	BMFO	TSBM	BMFO
#1	5	< 1	2	7	7
#2	16	< 1	1	7	8
#3	3	1	26	8	7
#4	5	< 1	< 1	6	6
#5	143	13	805	6	6
#6	47	1	10	6	6
#7	205	9	316	6	6
#8	17	< 1	< 1	5	6
#9	8	< 1	< 1	6	6
	449	**27**	**1159**	**57**	**58**

Table 8.3: Construction times and times for rewriting `voc.txt` ($\approx$ 186 KiB) from [5] of the two-step bimachines (TSBM) and the bimachines with final output (BMFO) for Porter stemming algorithm. All times are the average of 5 executions of the program and are rounded up to a millisecond.

8.3 Empirical results

Table 8.1 shows the sizes of the bimachines with final output for each step of the algorithm. It also compares the sizes before and after pseudo-minimization. It can be seen that the pseudo-minimization significantly reduces the size of these machines.

Table 8.2 shows the sizes of the two-step bimachines for each step of the same algorithm. The difference in sizes before and after pseudo-minimization is negligible and it is not shown.

Table 8.3 compares the time taken to construct both types of bimachines and the time taken to rewrite a sample dictionary of words provided in [5].[10] The experiment was conducted on Intel Core i7-8086K CPU. The code was compiled using GCC 13.2.0 with flags `-O3 -std=c++23`.

From these results we can observe that while pseudo-minimization has almost no effect on two-step bimachines, the presented construction already produces them with sizes similar to the sizes of the pseudo-minimized bimachines with final output. This not only results in faster construction, but also the memory needed by the construction algorithm is significantly less in these cases. This can be crucial for larger examples where the main memory is insufficient to store the non-pseudo-minimized bimachine with final output, but it is sufficient to store the two-step bimachine. The huge difference in the construction times of both machines is also due to the fact that the left and the right automaton of TSBM are independent, while for the BMFO, the left automaton depends on the right so that during the construction of *each* left state, all of the right states

[10]Since our implementation of regular expressions does not support indication of the empty string at the beginning/end of the line (^ and $ in POSIX regex), at least one whitespace must be inserted before the first word and after the last word of the file.

have to be traversed.

In some cases it is possible that the rewriting with a two-step bimachine is a little slower compared to the rewriting with a classical bimachine. This is due to the fact that two-step bimachines may have to query multiple function at each step in order to compute their next state and the corresponding output. However, there is no noticeable difference in rewriting times between the two types of bimachines in this example (due to small optimization which eliminates the lookups in δ and ψ_δ when the two-step bimachine is in state q_{err} (i.e. it is outside of match), and the lookups in τ and ψ_τ when the two-step bimachine is in the middle of a match).

9 Conclusion

In this thesis we introduced the two-step bimachines and a construction which creates a two-step bimachine given a finite-state representation of batch of rules. The construction directly creates the two-step bimachine without a need to construct the corresponding real-time functional transducer first. We proved the correctness of the presented construction and saw that two-step bimachines represent exactly the class of regular functions. We showed a similar construction which creates an equivalent bimachine with final output which in turn can be expressed as an equivalent up to ϵ classical bimachine. We found upper bounds for the sizes of the bimachines obtained from these constructions. It remains an open question whether these upper bounds can be improved.

We implemented both constructions and compared how they perform in practice. We used the rules of the Porter stemming algorithm as an example and saw that in all steps that are complex enough to notice the difference, the two-step bimachine was constructed much faster than the corresponding bimachine with final output. Namely, the total time for constructing the two-step bimachines was about 40 times lower than the total time for constructing the bimachines with final output. We also observed that the sizes of the constructed two-step bimachines are already similar to the sizes of the pseudo-minimized bimachines with final output which suggests that the first construction can be executed on machines having lower amount of internal memory. Lastly, we saw that, in this example, the time for text rewriting with the two types of bimachines is almost the same.

References

[1] Ronald M. Kaplan and Martin Kay. 1994. Regular Models of Phonological Rule Systems. Computational Linguistics, 20(3):331–378.

[2] Stoyan Mihov and Klaus U. Schulz: Finite-State Techniques, Automata, Transducers and Bimachines, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2019.

[3] Gerdjikov, S., Mihov, S., Schulz, K.U. (2017). A Simple Method for Building Bimachines from Functional Finite-State Transducers. In: Carayol, A., Nicaud, C. (eds) Implementation and Application of Automata. CIAA 2017. Lecture Notes in Computer Science(), vol 10329. Springer, Cham. https://doi.org/10.1007/978-3-319-60134-2_10

[4] Gerdjikov, S., Mihov, S., Schulz, K.U. (2019). Space-efficient bimachine construction based on the equalizer accumulation principle. https://doi.org/10.1016/j.tcs.2019.04.027

[5] M. F. Porter (1980): The Porter Stemming Algorithm (An algorithm for suffix stripping). https://tartarus.org/martin/PorterStemmer/

A Compact version of the original description of the Porter stemming algorithm

In this appendix the rules of the Porter stemmer and the description of their format are copied from [5]. This is only for comparison with our transformation of them (appendix B).

```
A \consonant\ in a word is a letter other than A, E, I, O or U,
and other than Y preceded by a consonant. If a letter is not a
consonant it is a \vowel\.

A consonant will be denoted by c, a vowel by v. A list ccc... of
length greater than 0 will be denoted by C, and a list vvv... of
length greater than 0 will be denoted by V. Any word, or part of
a word, may be represented by the form

        [C]VCVC ... [V]

where the square brackets denote arbitrary presence of their
contents. Using (VC){m} to denote VC repeated m times, this may
be written as

        [C](VC){m}[V].

m will be called the \measure\ of any word or word part when
represented in this form.

The \rules\ for removing a suffix will be given in the for

        (condition) S1 -> S2

The `condition' part may also contain the following:

        *S  - the stem ends with S (and similarly for the other
        letters).
        *v* - the stem contains a vowel.
        *d  - the stem ends with a double consonant (e.g. -TT,
        -SS).
        *o  - the stem ends cvc, where the second c is not W, X
        or Y (e.g. -WIL, -HOP).

In a set of rules written beneath each other, only one is obeyed,
and this will be the one with the longest matching S1 for the
given word.

Step 1a
```

```
        SSES -> SS
        IES  -> I
        SS   -> SS
        S    ->

Step 1b
        (m>0) EED -> EE
        (*v*) ED  ->
        (*v*) ING ->

If the second or third of the rules in Step 1b is successful, the
following is done:
        AT -> ATE
        BL -> BLE
        IZ -> IZE
        (*d and not (*L or *S or *Z))  -> single letter
        (m=1 and *o) -> E

Step 1c
        (*v*) Y -> I

Step 2
        (m>0) ATIONAL ->  ATE
        (m>0) TIONAL  ->  TION
        (m>0) ENCI    ->  ENCE
        (m>0) ANCI    ->  ANCE
        (m>0) IZER    ->  IZE
        (m>0) BLI     ->  BLE
        (m>0) ALLI    ->  AL
        (m>0) ENTLI   ->  ENT
        (m>0) ELI     ->  E
        (m>0) OUSLI   ->  OUS
        (m>0) IZATION ->  IZE
        (m>0) ATION   ->  ATE
        (m>0) ATOR    ->  ATE
        (m>0) ALISM   ->  AL
        (m>0) IVENESS ->  IVE
        (m>0) FULNESS ->  FUL
        (m>0) OUSNESS ->  OUS
        (m>0) ALITI   ->  AL
        (m>0) IVITI   ->  IVE
        (m>0) BILITI  ->  BLE
        (m>0) LOGI    ->  LOG

Step 3
        (m>0) ICATE ->  IC
        (m>0) ATIVE ->
        (m>0) ALIZE ->  AL
        (m>0) ICITI ->  IC
        (m>0) ICAL  ->  IC
        (m>0) FUL   ->
        (m>0) NESS  ->

Step 4
        (m>1) AL    ->
        (m>1) ANCE  ->
        (m>1) ENCE  ->
        (m>1) ER    ->
        (m>1) IC    ->
        (m>1) ABLE  ->
        (m>1) IBLE  ->
        (m>1) ANT   ->
        (m>1) EMENT ->
        (m>1) MENT  ->
        (m>1) ENT   ->
        (m>1 and (*S or *T)) ION ->
        (m>1) OU    ->
        (m>1) ISM   ->
        (m>1) ATE   ->
        (m>1) ITI   ->
        (m>1) OUS   ->
        (m>1) IVE   ->
        (m>1) IZE   ->

Step 5a
        (m>1) E     ->
        (m=1 and not *o) E ->

Step 5b
        (m > 1 and *d and *L) -> single letter
```

B Porter stemming algorithm in form of contextual replacement rules (`PorterStemmer.hpp`)

```
#ifndef PORTERSTEMMER_HPP
#define PORTERSTEMMER_HPP

#include <vector>
#include <string>
#include <cstddef>
#include "constants.hpp"
#include "contextualReplacementRule.hpp"

namespace PorterStemmer
{
  inline const std::string
    alphabet = "abcdefghijklmnopqrstuvwxyz \r\n\t\v\x01\x02",
    letter = "(a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z)",
    whitespace = "( |\r|\n|\t|\v)",
    always_vowel = "(a|e|i|o|u)",
    vowel_or_y = "(a|e|i|o|u|y)",
    always_consonant = "(b|c|d|f|g|h|j|k|l|m|n|p|q|r|s|t|v|w|x|z)",
    consonant_or_y = "(b|c|d|f|g|h|j|k|l|m|n|p|q|r|s|t|v|w|x|y|z)",
    consonant_not_wxy = "(b|c|d|f|g|h|j|k|l|m|n|p|q|r|s|t|v|z)",

    contains_vowel = '(' + letter + '*' + always_vowel + letter + "*|" +
      letter + '*' + consonant_or_y + vowel_or_y + letter + "*)",
    C = '(' + consonant_or_y + always_consonant + "*)",
    V = '(' + vowel_or_y + always_vowel + "*)",
    opt_C = '(' + C + "|_)",
    opt_V = '(' + V + "|_)",
    VC = '(' + V + C + ')',
    V_starting_non_y = '(' + always_vowel + always_vowel + "*)",

    m_gt_0 = "((" + C + VC + '|' + V_starting_non_y + C + ')' +
      VC + '*' + opt_V + ')',
    m_gt_1 = "((" + C + VC + '|' + V_starting_non_y + C + ')' +
      VC + VC + '*' + opt_V + ')',
    eps{Constants::Epsilon},
    rctx{'\x02'};

  using namespace std::string_literals;
  inline const std::vector<ContextualReplacementRule> steps[] = {
    { // 1a (#1)
      {"[sses,ss]"s, eps, whitespace},            // SSES -> SS
      {"[ies,i]"s, eps, whitespace},              // IES  -> I
      {"[ss,ss]"s, letter, whitespace},            // (length > 2) SS   -> SS
      {"[s,_]"s, letter + letter, whitespace},        // (length > 2) S    ->
      {"[_,\x02]"s, letter + letter + letter, whitespace},  // length > 2
    },
    { // 1b (#2) // '\x01' is used as a marker for whether step 1b' should take place
      {"[eed,ee]"s, m_gt_0, rctx},      // (m>0) EED -> EE
      {"[eed,eed]"s, eps, rctx}, // suppresses the rules below if the word ends in 'eed'
      {"[ed,\x01]"s, contains_vowel, rctx},  // (*v*) ED  ->
      {"[ing,\x01]"s, contains_vowel, rctx},  // (*v*) ING ->
    },
    { // 1b' (#3)
      {"[at\x01,ate]"s, eps, rctx},    // AT -> ATE
      {"[bl\x01,ble]"s, eps, rctx},    // BL -> BLE
      {"[iz\x01,ize]"s, eps, rctx},    // IZ -> IZE
      {"([bb,b]|[cc,c]|[dd,d]|[ff,f]"
       "|[gg,g]|[hh,h]|[jj,j]|[kk,k]"
       "|[mm,m]|[nn,n]|[pp,p]|[qq,q]"
       "|[rr,r]|[tt,t]|[vv,v]|[ww,w]"
       "|[xx,x])[\x01,_]"s, eps, rctx
      },               // (*d and not (*L or *S or *Z)) -> single letter
      {"[\x01,e]"s, whitespace + C + vowel_or_y +
        consonant_not_wxy, rctx
      },          // (m=1 and *o) -> E
      {"[\x01,_]"s, eps, rctx},      // deletes marker '\x01'
    },
    { // 1c (#4)
      {"[y,i]"s, contains_vowel, rctx},  // (*v*) Y -> I
    },
    { // 2 (#5)
      {"[ational,ate]"s, m_gt_0, rctx},  // (m>0) ATIONAL ->  ATE
      {"[ational,ational]"s, eps, rctx},
      {"[tional,tion]"s, m_gt_0, rctx},  // (m>0) TIONAL  ->  TION
      {"[tional,tional]"s, eps, rctx},
      {"[enci,ence]"s, m_gt_0, rctx},    // (m>0) ENCI    ->  ENCE
      {"[enci,enci]"s, eps, rctx},
      {"[anci,ance]"s, m_gt_0, rctx},    // (m>0) ANCI    ->  ANCE
      {"[anci,anci]"s, eps, rctx},
      {"[izer,ize]"s, m_gt_0, rctx},    // (m>0) IZER    ->  IZE
      {"[izer,izer]"s, eps, rctx},
      {"[bli,ble]"s, m_gt_0, rctx},    // (m>0) BLI    ->  BLE
      {"[bli,bli]"s, eps, rctx},
      {"[alli,al]"s, m_gt_0, rctx},    // (m>0) ALLI    ->  AL
      {"[alli,alli]"s, eps, rctx},
      {"[entli,ent]"s, m_gt_0, rctx},    // (m>0) ENTLI   ->  ENT
      {"[entli,entli]"s, eps, rctx},
      {"[eli,e]"s, m_gt_0, rctx},      // (m>0) ELI     ->  E
      {"[eli,eli]"s, eps, rctx},
      {"[ousli,ous]"s, m_gt_0, rctx},    // (m>0) OUSLI   ->  OUS
      {"[ousli,ousli]"s, eps, rctx},
      {"[ization,ize]"s, m_gt_0, rctx},  // (m>0) IZATION ->  IZE
      {"[ization,ization]"s, eps, rctx},
      {"[ation,ate]"s, m_gt_0, rctx},    // (m>0) ATION   ->  ATE
      {"[ation,ation]"s, eps, rctx},
      {"[ator,ate]"s, m_gt_0, rctx},    // (m>0) ATOR    ->  ATE
      {"[ator,ator]"s, eps, rctx},
      {"[alism,al]"s, m_gt_0, rctx},    // (m>0) ALISM   ->  AL
      {"[alism,alism]"s, eps, rctx},
      {"[iveness,ive]"s, m_gt_0, rctx},  // (m>0) IVENESS ->  IVE
      {"[iveness,iveness]"s, eps, rctx},
      {"[fulness,ful]"s, m_gt_0, rctx},  // (m>0) FULNESS ->  FUL
      {"[fulness,fulness]"s, eps, rctx},
      {"[ousness,ous]"s, m_gt_0, rctx},  // (m>0) OUSNESS ->  OUS
      {"[ousness,ousness]"s, eps, rctx},
      {"[aliti,al]"s, m_gt_0, rctx},    // (m>0) ALITI   ->  AL
      {"[aliti,aliti]"s, eps, rctx},
      {"[iviti,ive]"s, m_gt_0, rctx},    // (m>0) IVITI   ->  IVE
      {"[iviti,iviti]"s, eps, rctx},
      {"[biliti,ble]"s, m_gt_0, rctx},  // (m>0) BILITI  ->  BLE
      {"[biliti,biliti]"s, eps, rctx},
      {"[logi,log]"s, m_gt_0, rctx},    // (m>0) LOGI    ->  LOG
      {"[logi,logi]"s, eps, rctx},
    },
    { // 3 (#6)
      {"[icate,ic]"s, m_gt_0, rctx},    // (m>0) ICATE ->  IC
      {"[ative,_]"s, m_gt_0, rctx},    // (m>0) ATIVE ->
      {"[alize,al]"s, m_gt_0, rctx},    // (m>0) ALIZE ->  AL
      {"[iciti,ic]"s, m_gt_0, rctx},    // (m>0) ICITI ->  IC
      {"[ical,ic]"s, m_gt_0, rctx},    // (m>0) ICAL  ->  IC
      {"[ful,_]"s, m_gt_0, rctx},      // (m>0) FUL   ->
      {"[ness,_]"s, m_gt_0, rctx},    // (m>0) NESS  ->
    },
    { // 4 (#7)
      {"[al,_]"s, m_gt_1, rctx},      // (m>1) AL    ->
      {"[al,al]"s, eps, rctx},
      {"[ance,_]"s, m_gt_1, rctx},    // (m>1) ANCE  ->
      {"[ance,ance]"s, eps, rctx},
      {"[ence,_]"s, m_gt_1, rctx},    // (m>1) ENCE  ->
      {"[ence,ence]"s, eps, rctx},
      {"[er,_]"s, m_gt_1, rctx},      // (m>1) ER    ->
      {"[er,er]"s, eps, rctx},
      {"[ic,_]"s, m_gt_1, rctx},      // (m>1) IC    ->
      {"[ic,ic]"s, eps, rctx},
      {"[able,_]"s, m_gt_1, rctx},    // (m>1) ABLE  ->
      {"[able,able]"s, eps, rctx},
      {"[ible,_]"s, m_gt_1, rctx},    // (m>1) IBLE  ->
      {"[ible,ible]"s, eps, rctx},
      {"[ant,_]"s, m_gt_1, rctx},      // (m>1) ANT   ->
      {"[ant,ant]"s, eps, rctx},
      {"[ement,_]"s, m_gt_1, rctx},    // (m>1) EMENT ->
      {"[ement,ement]"s, eps, rctx},
      {"[ment,_]"s, m_gt_1, rctx},    // (m>1) MENT  ->
      {"[ment,ment]"s, eps, rctx},
      {"[ent,_]"s, m_gt_1, rctx},      // (m>1) ENT   ->
      {"[ent,ent]"s, eps, rctx},
      {"[ion,_]"s, "((" + C + VC + '|' + V_starting_non_y + C + ')' +
        VC + '*' + V + opt_C + "(s|t))", rctx
      },                        // (m>1 and (*S or *T)) ION ->
      {"[ion,ion]"s, eps, rctx},
      {"[ou,_]"s, m_gt_1, rctx},      // (m>1) OU    ->
      {"[ou,ou]"s, eps, rctx},
      {"[ism,_]"s, m_gt_1, rctx},      // (m>1) ISM   ->
      {"[ism,ism]"s, eps, rctx},
      {"[ate,_]"s, m_gt_1, rctx},      // (m>1) ATE   ->
      {"[ate,ate]"s, eps, rctx},
      {"[iti,_]"s, m_gt_1, rctx},      // (m>1) ITI   ->
      {"[iti,iti]"s, eps, rctx},
      {"[ous,_]"s, m_gt_1, rctx},      // (m>1) OUS   ->
      {"[ous,ous]"s, eps, rctx},
      {"[ive,_]"s, m_gt_1, rctx},      // (m>1) IVE   ->
      {"[ive,ive]"s, eps, rctx},
      {"[ize,_]"s, m_gt_1, rctx},      // (m>1) IZE   ->
      {"[ize,ize]"s, eps, rctx},
    },
    { // 5a (#8)
      {"[e,_]"s, m_gt_1, rctx},        // (m>1) E     ->
      {"[e,_]"s, whitespace + "((" +
        '(' + always_vowel + always_vowel + "*)" + C + opt_V + ")|(" +
        C + VC + V + ")|(" +
        C + V + always_vowel + C + opt_V + ")|(" +
        C + V + "(w|x|y)" + ")|(" +
        C + V + C + always_consonant + opt_V +
        "))", rctx
      },                        // (m=1 and not *o) E ->
    },
    { // 5b (#9)
      {"[l,_]"s, "((" + C + VC + '|' + V_starting_non_y + C + ')'
        + VC + '*' + V + opt_C + "l)", rctx
      },        // (m > 1 and *d and *L) -> single letter
```

```
      {"[\x02,_]"s, eps, whitespace},   // deletes the marker '\x02'
    },
  };
  inline constexpr std::size_t steps_cnt = sizeof(steps) / sizeof(*steps);
}

#endif
```