Direct Construction of a Bimachine for
Context-Sensitive Rewrite Rule

A thesis submitted for the degree of Master of Computer Science
by
Ivan Petrov Peikov

— Sofia, 2006 —
Contents

1 Introduction 3

2 Contexts and Rules 4

3 Finite State Automata 8

4 Deterministic Finite Automata 12

5 Bimachines 15

6 Direct Construction 18
   6.1 The Left Automaton ........................................... 19
   6.2 The Right Automaton ........................................... 19
   6.3 The Output Function ........................................... 19
   6.4 Bimachine For Context-Sensitive Rewrite Rule .............. 24

7 Algorithms 27
   7.1 Concatenation to the Left .................................... 27
   7.2 Translation of an Output-Driven Bimachine .................. 28
   7.3 Direct Construction of a Bimachine for Context-Sensitive Rewrite Rule ........................................... 29

8 Complexity 31
1 Introduction

Context-sensitive rewrite rules are a well-known formalism practically useful in many fields of computational linguistics. They were first introduced in Chomsky’s papers ([1]) and proved to be expressive enough to successfully model multiple linguistic phenomena.

In 1972 Johnson ([2]) notices that with the limitation to work only over their input the context-sensitive rewrite rules become as expressive as the regular relations. This purely theoretical result is later confirmed by Kaplan and Kay ([3]) who show the practical importance of context-sensitive rewrite rules when implemented with finite state transducers. Their paper gives rise to many consecutive works and remains one of the classics in the contemporary computational linguistics.

Bimachines as introduced by Schützenberger ([4]) are deterministic abstract machines as expressive as the regular functions. They consume their input simultaneously from left to right and from right to left. On every position they produced output based on the left-hand prefix and the right-hand suffix. Bimachines are computationally very efficient which makes them applicable in practice.

Our purpose is to present a construction which by given context-sensitive rewrite rule builds directly a bimachine realizing its corresponding regular function.

There exist other methods which first construct a finite-state transducer realizing the context rule ([3]) and then translate the transducer into a bimachine ([5]). In 2004, S. et al. ([6]) present a direct construction of a bimachine for a restricted form of rewrite rule. However, to the best of our knowledge, the construction presented in the current thesis is the first direct construction of a bimachine realizing a non-restricted context-sensitive rewrite rule.

The rest of the thesis is structured as follows. In Section 2 rewrite rules are introduced and the problem is formally defined. In Sections 3 and 4 finite state automata are introduced. In Section 5 bimachines are introduced together with two of their possible operational semantics. In Section 6 the direct construction is described and proved correct. The remaining Sections 7 and 8 present some of the algorithms used and study their complexity.

We denote concatenation of two words $\alpha$ and $\beta$ with $\alpha \cdot \beta$ or $\alpha\beta$ (when no ambiguity could possibly occur). The fact that $\alpha$ is a subword of $\beta$ is denoted as $\alpha \subseteq \beta$ or $\alpha \subseteq \beta$ in case when equality is possible. All other notations will either be introduced in the appropriate context or will be considered as widely adopted.
2 Contexts and Rules

Let's fix a finite alphabet $\Sigma$. Context-sensitive rewrite rule is any rule of the type

$$E \rightarrow \beta / L \_ R$$

where $L, E, R \in \mathcal{RE}(\Sigma)$ are regular expressions over $\Sigma$, and $\beta \in \Sigma^*$. Application of such a rule over a fixed word $\alpha \in \Sigma$ is the simultaneous rewriting with $\beta$ of all $\alpha$’s subwords which belong to the language $\mathcal{L}(E)$ of $E$ and are found between a subword of $\mathcal{L}(L)$ to the left and subword of $\mathcal{L}(R)$ to the right:

$$\mathcal{L}(t; L, E, R) = \{ (u, v, w) | u \in \mathcal{L}(\Sigma^* L) \& v \in \mathcal{L}(E) \& w \in \mathcal{L}(R \Sigma^*) \& t = uvw \}$$

to be the set of all such triples or the context set for any particular $t$ and $E \rightarrow \beta / L \_ R$.

It becomes clear that there are certain ambiguities that might occur while applying a context rule over some word. For example, if two contexts share a common prefix but differ in their focuses (and respectively suffixes) the result of the rewriting will be ambiguous:

Such an ambiguity will be resolved by choosing for valid the context with the longest focus.

Example 2.1. Let’s apply the rule $a^+ \rightarrow A / b _\_ _a$ over the word $\alpha = baaaab$. The result might ambiguously be defined as $bAaab$, $bAab$ or $bAb$ (rewriting respectively in context $(b, a, aab)$, $(b, aa, aab)$ or $(b, aaa, ab)$). By agreeing to always take the context with the longest focus for valid we resolve the ambiguity and obtain the single result $bAb$.

Definition 2.2. Two contexts $(u_i, v_i, w_i) \in C$ for $i = 1, 2$ are said to overlap (written $(u_1, v_1, w_1) \prec (u_2, v_2, w_2)$), if $u_1 \subset u_2 \subset u_1 v_1$. 

4
Whenever such a situation occurs while applying a context rule not more than one of the two contexts should be taken for valid. We’ll resolve such an ambiguity by always taking the leftmost of all the possibilities and ignoring all the overlapped contexts.

We call such a strategy for resolving ambiguities leftmost longest match strategy.

Example 2.2. Let’s take the rule $xy|yz \rightarrow \epsilon / x \_ z$. During its application over word like $xyzzxyzz$ ambiguity arises. Possible rewriting contexts are $(x, yz, xxyz), (xyzzz, x, zy)$ and $(xyzxx, yz, z)$, the last two if which do overlap. The leftmost longest match strategy sorts out the first two contexts as valid for rewriting which produces the non-ambiguous result of $xzzzz$.

There exist other strategies for resolving ambiguities. We’ll concentrate on the described one because we believe it is the most natural and most useful in practice.

Let’s now define formally the set of all rewriting contexts within a fixed word which we’ll consider valid for rewriting.

Definition 2.3. We define the following operators over sets of contexts:

$$\text{OVER}(C, C') = \{(u, v, w) \in C | (\exists (u', v', w') \in C')((u', v', w') \prec (u, v, w))\}$$
$$\text{LEFT}(C) = \{(u, v, w) \in C | (\exists (u', v', w') \in C)(u' \subset u)\}$$
$$\text{LONG}(C) = \{(u, v, w) \in C | (\exists (u', v', w') \in C)(v \subset v')\}$$

The first operator \text{OVER} defines all contexts in a given set which are overlapped by a context in another set. The second operator \text{LEFT} defines the leftmost contexts in a given set.

Considering the above definitions we may proceed with defining formally the valid contexts. We construct inductively the sequence $\{C_i\}_{i=0}^{\infty}$, where $C_i \subseteq C$ for all $i \geq 0$:

- $C_0 = \emptyset$
- $C_{i+1} = C_i \cup \text{LEFT}(C - C_i - \text{OVER}(C, C_i))$

This is a monotonously increasing sequence and hence it has a least upper bound. We define $C_v = \cup_{i=0}^{\infty} C_i$ we call $C_v$ the valid contexts set. We denote $C_l = \text{LONG}(C_v)$ and we call $C_l$ the set of the longest valid contexts.

Proposition 2.1. $C_v \subseteq C$

Proposition 2.2. $C_v$ doesn’t contain any overlapping contexts.

Proof. Let’s assume $(u_i, v_i, w_i) \in C_v$ for $i = 1, 2$ and $(u_1, v_1, w_1) \prec (u_2, v_2, w_2)$. Then $u_1 \subset u_2 \subset u_1 v_1$. Let $k_i$ be the least indices for which $(u_i, v_i, w_i) \in C_{k_i}$ (we’re sure they exist because $(u_i, v_i, w_i) \in C_v = \cup_{i=0}^{\infty} C_i$).

Let’s assume that $k_1 \geq k_2$. Then

$$(u_2, v_2, w_2) \in C_{k_2} = C_{k_2 - 1} \cup \text{LEFT}(C - C_{k_2 - 1} - \text{OVER}(C, C_{k_2 - 1}))$$
We denote $R = \mathcal{C} - \mathcal{C}_{k_2-1} - \text{OVER}(\mathcal{C}, \mathcal{C}_{k_2-1})$. As $\langle u_2, v_2, w_2 \rangle \not\in \mathcal{C}_{k_2-1}$, we conclude that $\langle u_2, v_2, w_2 \rangle \in \text{LEFT}(R)$. On the other hand $k_1 \geq k_2$, hence $\langle u_1, v_1, w_1 \rangle \in R$. This is a contradiction with the definition of LEFT so finally $k_1 < k_2$.

After we saw that $k_1 < k_2$, we may further conclude that $\langle u_2, v_2, w_2 \rangle \not\in \mathcal{C}_k$ for $k \leq k_1$. For $k \geq k_1$, $\langle u_1, v_1, w_1 \rangle \in \mathcal{C}_k$ therefore $\langle u_2, v_2, w_2 \rangle \not\in \text{OVER}(\mathcal{C}, \mathcal{C}_k)$, e.g. $\langle u_2, v_2, w_2 \rangle \not\in \mathcal{C}_{k+1}$. This contradicts with the fact that $\langle u_2, v_2, w_2 \rangle \in \mathcal{C}_{k_2}$ ($k_2 > k_1$) and hence $\mathcal{C}_v$ doesn't contain any overlapping contexts.

\section*{Proposition 2.3}

Let $\langle u, v, w \rangle \in \mathcal{C}$ is a non-valid context ($\langle u, v, w \rangle \not\in \mathcal{C}_v$). There exists $(u_0, v_0, w_0) \in \mathcal{C}_v$, such that $\langle u_0, v_0, w_0 \rangle \prec \langle u, v, w \rangle$.

\begin{proof}
Let's take the sequence $\{O_i\}_{i=0}^\infty$, where $O_i = \text{OVER}(\mathcal{C}, \mathcal{C}_i)$. One could easily prove that this is a monotonously increasing sequence whose least upper bound exists and is exactly $\text{OVER}(\mathcal{C}, \mathcal{C}_v)$.

We'll verify that $\mathcal{C}_v \cup \text{OVER}(\mathcal{C}, \mathcal{C}_v) = \mathcal{C}$. Let's assume that there exists $(u, v, w) \in \mathcal{C}$ such that $(u, v, w) \not\in \mathcal{C}_v$ and $(u, v, w) \not\in \text{OVER}(\mathcal{C}, \mathcal{C}_v)$ and let's choose the one with shortest prefix. $\mathcal{C}$ is a finite set so there exists and index $k$ such that $\mathcal{C}_k = \mathcal{C}_v$. $(u, v, w) \not\in \text{OVER}(\mathcal{C}, \mathcal{C}_k)$ which implies that $\mathcal{C} - \mathcal{C}_k - \text{OVER}(\mathcal{C}, \mathcal{C}_k) \neq \emptyset$, and therefore $\mathcal{C}_v \subset \mathcal{C}_{k+1}$ is not a least upper bound of $\{O_i\}$. This is a contradiction and hence $\mathcal{C}_v \cup \text{OVER}(\mathcal{C}, \mathcal{C}_v) = \mathcal{C}$.

Now back to the proposition. Let $(u, v, w) \not\in \mathcal{C}_v$. Then (according to everything already said) $(u, v, w) \in \text{OVER}(\mathcal{C}, \mathcal{C}_v)$ which means that there is an index $p$ ($\text{OVER}(\mathcal{C}, \mathcal{C}_v)$ is the least upper bound of $\{O_i\}$ for which $(u, v, w) \in O_p$. Then by the definition of $O_i$ it follows that there exists $(u_0, v_0, w_0) \in O_p$, for which $(u_0, v_0, w_0) \prec (u, v, w)$.

\section*{Proposition 2.4}

Let $\langle u, v, w \rangle \in \mathcal{C}$ are contexts ($i = 1, 2$). Then $\langle u, v_1, w_1 \rangle$ is a valid context iff $\langle u, v_2, w_2 \rangle$ is also valid.

\begin{proof}
Let's (u, v_1, w_1) is a valid context and let's assume (u, v_2, w_2) is non-valid. Then there exists $k \geq 0$, such that $\langle u_2, v_2, w_2 \rangle \in \text{OVER}(\mathcal{C}, \mathcal{C}_k)$. This would mean that there exists $(u_0, v_0, w_0) \subset \mathcal{C}_k$ such that $u_0 \subset u \subset u_0 w_0$. Therefore $\langle u_0, v_0, w_0 \rangle \prec (u, v_1, w_1)$. But according to Proposition 2.2 there are no overlapping contexts in $\mathcal{C}_v$ which contradicts with our assumption.

Follows a definition of result from the application of a context-sensitive rule over some given input text.

\section*{Definition 2.4}

Let $\alpha \in \Sigma^*$ and $E \rightarrow \beta / L_R$ is a context-sensitive rewrite rule. Result from the application of $E \rightarrow \beta / L_R$ over $\alpha$ is the word $u_1 \cdot \beta \cdot (u_1 v_1)^{-1} u_2 \cdot \beta \cdot (u_2 v_2)^{-1} u_3 \cdot \beta \cdot \ldots \cdot (u_{k-1} v_{k-1})^{-1} u_k \cdot \beta \cdot w_k$

where $\mathcal{C}_i(\alpha; L, E, R) = \{\langle u_i, v_i, w_i \rangle | i = 1, \ldots, k \}$ and $i < j \Rightarrow u_i \subset u_j$.

The direct usage of Definition 2.4 would greatly complicate our treatment of the problem, we introduce the following equivalent definition.
Definition 2.5. Let $\alpha = a_1a_2 \ldots a_n \in \Sigma^*$ and $E \rightarrow \beta / L \ R$ is a context-sensitive rewrite rule. Result from the application of $E \rightarrow \beta / L \ R$ over $\alpha$ is the word $\omega_1 \cdot \omega_2 \ldots \omega_n \cdot \omega_{n+1}$, where for $i = 1, 2, \ldots, n$

$$\omega_i = \begin{cases} 
\epsilon & \text{if there exists } \langle u, v, w \rangle \in C_v(\alpha; L, E, R), \ |u| < i - 1 < |uv| \\
\beta & \text{if there exists } \langle u, v, w \rangle \in C_v(\alpha; L, E, R), \ |u| = i - 1 < |uv| \\
a_i & \text{if there exists } \langle u, v, w \rangle \in C_v(\alpha; L, E, R), \ |u| = i - 1 = |uv| \\
\beta \alpha_i & \text{and there is no } \langle u, v', w' \rangle \in C_v(\alpha; L, E, R), \text{ such that } v' \neq \epsilon \\
\epsilon & \text{otherwise}
\end{cases}$$

and $\omega_{n+1} \in \Sigma^*$ is defined as follows:

$$\omega_{n+1} = \begin{cases} 
\beta & \text{if } \langle \alpha, \epsilon, \epsilon \rangle \in C_v(\alpha; L, E, R) \\
\epsilon & \text{otherwise}
\end{cases}$$

In order to simplify some of the proofs we further modify Definition 2.5 and finally reach the equivalent

Definition 2.6. Let $\alpha = a_1a_2 \ldots a_n \in \Sigma^*$ and $E \rightarrow \beta / L \ R$ is a context-sensitive rewrite rule. Result from the application of $E \rightarrow \beta / L \ R$ over $\alpha$ is the word $(\omega_1\pi_1) \cdot (\omega_2\pi_2) \ldots (\omega_n\pi_n)$, where $\omega_i$ are defined as in Definition 2.5, and

$$\pi_i = \begin{cases} 
\beta & \text{if } i = n \text{ and } \langle \alpha, \epsilon, \epsilon \rangle \in C_v(\alpha; L, E, R) \\
\epsilon & \text{otherwise}
\end{cases}$$

The proof that Definitions 2.4, 2.5 and 2.6 are equivalent is trivial and too technical to be interesting for our treatment. Therefore we’ll use only Definition 2.6.
3 Finite State Automata

Definition 3.1. Finite state automaton (FSA) is a 5-tuple

$$A = \langle \Sigma, Q, S, F, \Delta \rangle$$

where $\Sigma$ is a finite alphabet, $Q$ is a finite set of states, $S \subseteq Q$ is a set of initial (starting) states, $F \subseteq Q$ is a set of accepting states, and $\Delta \subseteq Q \times \Sigma \times Q$ is transition relation. We extend inductively $\Delta$ to $\Delta^*$

- $(q, \varepsilon, q) \in \Delta^*$ for each $q \in Q$
- $(q_1, \alpha a, q_2) \in \Delta^*$ if there exists $q \in Q$, such that $(q_1, \alpha, q) \in \Delta^*$ and $(q, a, q_2) \in \Delta$

Definition 3.2. Let $\Delta \subseteq Q \times \Sigma \times Q$, $T \subseteq Q$ and $\alpha \in \Sigma^*$. We introduce the following notation

$$T \xrightarrow{\Delta^*} \{q|(\exists q \in T)((q', \alpha, q) \in \Delta)\}$$

Definition 3.3. Let $A$ is a FSA and $\alpha \in \Sigma^*$ is a word. We say that the sequence $q_0, q_1, \ldots, q_{|\alpha|}$ is an execution $A$ over $\alpha$, if $q_i \in Q$ for each $i = 0 \ldots |\alpha|$, $q_0 \in S$ and $(q_{i-1}, \alpha, q_i) \in \Delta$ for each $i = 1 \ldots |\alpha|$. One such execution will be assumed successful iff $q_{|\alpha|} \in F$ is accepting. Whenever the notation allows we'll also use an alternative definition of automaton execution, namely $\sigma : [0, |\alpha|] \to Q$. It is said that $A$ accepts (or recognizes) $\alpha$, if there exists successful execution of $A$ over $\alpha$.

Definition 3.4 (Language of FSA). Let $A$ is a FSA. We define the language of $A$ to be the set $L(A) = \{\alpha | A \text{ recognizes } \alpha\}$.

Definition 3.5 (Equivalent Automata). Let $A_1, A_2$ are two finite state automata. We say that $A_1$ is equivalent to $A_2$ (and write $A_1 \equiv A_2$), iff $L(A_1) = L(A_2)$.

Definition 3.6 (Normal form). Let $A = \langle \Sigma, Q, S, F, \Delta \rangle$ is a FSA. We say that $A$ is in normal form if for every transition $(q_1, a, q_2) \in \Delta$ it is the case that $q_1 \notin F$ and $q_2 \notin S$. Informally speaking, the normality of $A$ consists of the fact that no transition leaves accepting or enters initial state.

Proposition 3.1. For every FSA $A$, there exists an equivalent $A^N$, which is in normal form.

Proof. Let $A = \langle \Sigma, Q, S, F, \Delta \rangle$ is a FSA. We'll construct $A^N$ from $A$ by removing possible transitions which spoil its normality: $A^N = \langle \Sigma, Q^N, S^N, F^N, \Delta^N \rangle$, where

$$Q^N = Q \times \{1\} \cup S \times \{2\} \cup F \times \{3\}$$

The other components of $A^N$ are $S^N = S \times \{2\}$, $F^N = F \times \{3\}$

$$\Delta^N = \{\langle q_1, 1, a, (q_2, 1) \rangle | (q_1, a, q_2) \in \Delta \} \cup \{\langle q_1, 2, a, (q_2, 1) \rangle | (q_1, a, q_2) \in \Delta \& q_1 \in S \} \cup \{\langle q_1, 1, a, (q_2, 3) \rangle | (q_1, a, q_2) \in \Delta \& q_2 \in F \}$$
$A^N$ is obviously in normal form. Let's show that it is equivalent to $A$.

Let $\alpha \in \mathcal{L}(A)$ and $q_0, q_1, \ldots, q_{|\alpha|-1}, q_{|\alpha|}$ is a successful execution of $A$ over $\alpha$. We'll verify that $\langle q_0, 2 \rangle, \langle q_1, 1 \rangle, \langle q_2, 1 \rangle, \ldots, \langle q_{|\alpha|-1}, 1 \rangle, \langle q_{|\alpha|}, 3 \rangle$ is a successful execution of $A^N$ over $\alpha$. First, let's notice that $\langle q_0, 2 \rangle \in S^N$, because $q_0 \in S$ and $\langle q_{|\alpha|}, 3 \rangle \in F^N$, because $q_{|\alpha|} \in F$. It is clear by the construction of $\Delta^N$ that $\langle q_{i-1}, 1, \alpha_i, q_i, 1 \rangle \in \Delta^N$ for $i = 2 \ldots |\alpha| - 1$, because $\langle q_{i-1}, \alpha_i, q_i \rangle \in \Delta$ for $i = 2 \ldots |\alpha| - 1$. It is clear that $\langle q_0, \alpha_0, q_1 \rangle \in \Delta$ and $q_0 \in S$, and therefore by the definition of $\Delta^N$ it is true that $\langle \langle q_0, 2 \rangle, \alpha_0, \langle q_1, 1 \rangle \rangle \in \Delta^N$. By analogy, $\langle \langle q_{|\alpha|-1}, 1 \rangle, \alpha_{|\alpha|-1}, \langle q_{|\alpha|}, 3 \rangle \rangle \in \Delta^N$. Thus we showed that there exists a successful execution of $A^N$ over $\alpha$, i.e. $\alpha \in \mathcal{L}(A^N)$.

The proof of in the other direction is analogous. Therefore, we showed that $A \equiv A^N$.

**Theorem 3.1 (Kleene).** For every regular expression $E$, there exists a FSA $A$, such that $\mathcal{L}(A) = \mathcal{L}(E)$.

The automaton from Kleene’s Theorem is not unique and there are many constructions that build it directly from $E$ (for example [7]). The particular construction is not significant to our purposes since whenever we need to construct a FSA by a regular expression $E$, we'll write $A(E)$ and will mean a FSA in normal form such that $\mathcal{L}(A(E)) = \mathcal{L}(E)$.

**Definition 3.7 (Mirror FSA).** Let $A = (\Sigma, Q, S, F, \Delta)$ is a FSA. The FSA $\bar{A} = (\Sigma, Q, F, S, \bar{\Delta})$, where $\bar{\Delta} = \{(q_2, a, q_1) | \langle q_1, a, q_2 \rangle \in \Delta \}$ is said to be the mirror FSA of $A$.

**Proposition 3.2.** Let $A = (\Sigma, Q, S, F, \Delta)$ is a FSA and $\bar{A}$ is its mirror FSA. Then for every $\alpha \in \Sigma^*$, $q_0, q_1, \ldots, q_{|\alpha|}$ is a successful execution of $A$ iff $q_{|\alpha|-1}, q_{|\alpha|-2}, \ldots, q_0$ is a successful execution of $\bar{A}$.

**Proof.** Obvious by the definition of mirror automaton. 

It is a well-known fact that the class of regular languages is closed under concatenation. In other words if the languages $L_1, L_2$ are recognizable by (respectively) $A_{1,2}$, there exists an automaton $A$ recognizing $L_1 \cdot L_2$.

We'll define two constructions which define the concatenation of automata and have the special property that every their execution contains as subexecution an execution of (respectively) the first or the second automaton.

**Definition 3.8 (Concatenation to the Left).** Let $A_i = (\Sigma, Q_i, S_i, F_i, \Delta_i)$, for $i = 1, 2$ are FSA and $Q_1 \cap Q_2 = \emptyset$. We say that $A = (\Sigma, Q, S, F, \Delta)$ is the result from concatenating of $A_1$ to the left of $A_2$ and we write $A = A_1 \cdot A_2$, if:

- $Q = Q_1 \cup Q_2$
- $S = \begin{cases} S_1 \cup S_2 & \text{if } S_1 \cap F_1 \neq \emptyset \\ S_1 & \text{if } S_1 \cap F_1 = \emptyset \end{cases}$
- $F = F_2$
\[ \Delta = \Delta_1 \cup \Delta_2 \cup \{ \langle q_1, a, q_2 \rangle | (\exists q \in F) (\langle q_1, a, q \rangle \in \Delta_1 \land q_2 \in S_2) \} \]

In order to illustrate the concatenation to the left, let's again examine the example from Fig. 1. It is noticeable that every successful execution of the so constructed automaton \( A_1 \cdot A_2 \) contains a successful execution of \( A_2 \).

![Figure 1: Concatenation to the left](image)

**Proposition 3.3.** Let \( A_{1,2} \) are FSA and \( A = A_1 \cdot A_2 \). Then every successful execution of \( A \) over \( a_1 a_2 \ldots a_n \in \Sigma^* \) looks like

\[ q_0, q_{p-1}, q_p, \ldots, q_n \]

where \( p \in [0, n] \), \( q_p, \ldots, q_n \) is a successful execution of \( A_2 \) over \( a_{p+1} \ldots a_n \) and there exists a state \( q \in F_1 \) such that \( q_0, \ldots, q_{p-1}, q \) is a successful execution of \( A_1 \) over \( a_1 \ldots a_p \).

**Proof.** Let \( q_0, \ldots, q_n \) is some successful execution of \( A \) over \( a_1 a_2 \ldots a_n \). It is successful and therefore, \( q_n \) is an accepting state, e.g. \( q_n \in F = F_2 \) (by definition of the concatenation to the left). Let \( p \in [0, n] \) be the least index such that \( q_p \in Q_2 \) (it is certain that such a \( p \) exists, because \( q_0 \in Q_2 \)). Now we notice that for any \( i \in [p, n] \), \( q_i \in Q_2 \). This is true because if it weren't we would be able to choose \( i > p \), such that \( q_i \in Q_1 \) and it would follow that \( \langle q_{i-1}, a_i, q_i \rangle \in \Delta \), where \( q_{i-1} \in Q_2 \), and \( q_i \in Q_1 \). By the construction of \( A \), this is impossible.

Therefore \( q_0, \ldots, q_{p-1} \in Q_1 \), and \( q_p, \ldots, q_n \in Q_2 \). Now if \( p > 0 \), \( \langle q_{p-1}, a_p, q_p \rangle \in \Delta \) and by the definition of \( \Delta \) it is clear that \( q_p \in S_2 \). On the other hand if \( p = 0 \), it would follow \( q_p \in S_2 \), because \( q_0, \ldots, q_n \) is an execution of \( A \). Following the same reasoning we would deduce that \( \langle q_{i-1}, a_i, q_i \rangle \in \Delta_2 \) (for each \( i \in [p+1, n] \)), and consequently \( q_p, \ldots, q_n \) is a successful execution of \( A_2 \) over \( a_{p+1} \ldots a_n \).

If \( p = 0 \), then \( a_1 \ldots a_p = \varepsilon \). By the definition of \( A \) and the fact that \( q_0 \in S \land S_2 \neq \emptyset \), it follows that \( S \cap F_1 \neq \emptyset \). We choose a state \( q \in S \cap F_1 \) and thus we show a successful execution of \( A_1 \) over \( \varepsilon \). Now let \( p > 0 \). Then \( q_0 \in Q_1 \), and from here it follows that \( q_0 \in S_1 \). Because \( \langle q_{i-1}, a_i, q_i \rangle \in \Delta \) for \( i \in (0, p) \) and \( q_i \in Q_1 \) for \( i \in [0, p] \), it follows that \( \langle q_{i-1}, a_i, q_i \rangle \in \Delta_1 \) for \( i \in (0, p) \), and consequently \( q_0, \ldots, q_{p-1} \) is an execution of \( A_1 \) over \( a_1 \ldots a_{p-1} \). Now by the definition of \( A \) it follows that there exists \( q \in F_1 \), such that \( \langle q_{p-1}, a_p, q \rangle \in \Delta_1 \), and therefore \( q_0, \ldots, q_{p-1}, q \) is a successful execution of \( A_1 \) over \( a_1 \ldots a_p \). \[\square\]
Proposition 3.4. Let $A_{1,2}$ are FSA. Then $\mathcal{L}(A_1 \cdot A_2) = \mathcal{L}(A_1) \cdot \mathcal{L}(A_2)$.

Proof. Let $A_{1,2} = (\Sigma, Q_{1,2}, S_{1,2}, F_{1,2}, \Delta_{1,2})$ and the concatenation of $A_1$ to the left of $A_2$ is $A = A_1 \cdot A_2 = (\Sigma, Q, S, F, \Delta)$.

Let $\alpha \in \mathcal{L}(A)$. This means that there exists a successful execution $q_0, \ldots, q_n$ of $A$ over $\alpha$. According to Proposition 3.3 there exist successful executions $\overline{a}_1 \ldots a_p$ of $A_1$ over $\alpha$ and $a_{p+1} \ldots a_n$ of $A_2$ over $\overline{a}_1 \ldots a_p$. This means that (respectively) $\overline{a}_1 \ldots a_p \in \mathcal{L}(A_1)$ and $a_{p+1} \ldots a_n \in \mathcal{L}(A_2)$, and therefore $\alpha = \overline{a}_1 \ldots a_p a_{p+1} \ldots a_n \in \mathcal{L}(A_1) \cdot \mathcal{L}(A_2)$.

Conversely, $\alpha \in \mathcal{L}(A_1)$, and $\beta \in \mathcal{L}(A_2)$. Then there exist successful executions $q_{i_0}, \ldots, q_{i_m}$ of $A_1$ over $\alpha$ and $q_{j_0}, \ldots, q_{j_m}$ of $A_2$ over $\beta$. Obviously $q_i, \ldots, q_{i_{m-1}}, q_{j_1}, \ldots, q_{j_m}$ will be a successful execution of $A$ over $\alpha\beta$, and therefore $\alpha\beta \in \mathcal{L}(A)$. Thus we showed that $\mathcal{L}(A) = \mathcal{L}(A_1) \cdot \mathcal{L}(A_2)$.

By analogy with the operation concatenation to the left we define its dual — concatenation to the right. We omit the proofs as they are no different from the ones already shown.

Definition 3.9 (Concatenation to the Right). Let $A_i = (\Sigma, Q_i, S_i, F_i, \Delta_i)$, for $i = 1, 2$ are FSA and $Q_1 \cap Q_2 = \emptyset$. We say that $A = (\Sigma, Q, S, F, \Delta)$ is the result from concatenating of $A_2$ to the right of $A_1$ and we write $A = A_1 \cdot_r A_2$, if:

- $Q = Q_1 \cup Q_2$
- $S = S_1$
- $F = \left\{ \begin{array}{ll} F_1 \cup F_2 & \text{if } S_2 \cap F_2 \neq \emptyset \\ F_2 & \text{if } S_2 \cap F_2 = \emptyset \end{array} \right.$
- $\Delta = \Delta_1 \cup \Delta_2 \cup \{ (q_1, a, q_2) | (\exists q \in S_2) ((q, a, q_2) \in \Delta_2) & q_1 \in F_1 \} \}$

Proposition 3.5. Let $A_{1,2}$ are FSA and $A = A_1 \cdot_r A_2$. Then every successful execution of $A$ over $a_1a_2 \ldots a_n \in \Sigma^*$ looks like

$q_0, \ldots, q_p, q_{p+1}, \ldots, q_n$

where for some $p \in [0, n]$, $q_0, \ldots, q_p$ is a successful execution of $A_1$ over $a_1 \ldots a_p$ and there exists a state $q \in S_2$, such that $q, q_{p+1}, \ldots, q_n$ is a successful execution of $A_2$ over $a_{p+1} \ldots a_n$.

Proposition 3.6. Let $A_{1,2}$ are FSA. Then $\mathcal{L}(A_1 \cdot_r A_2) = \mathcal{L}(A_1) \cdot \mathcal{L}(A_2)$.
4 Deterministic Finite Automata

Definition 4.1. We call a finite state automaton \(A = (\Sigma, Q, S, F, \Delta)\) deterministic (DFA), if for every \(q_1 \in Q\) and \(a \in \Sigma\) there exists at most one \(q_2 \in Q\), such that \(\langle q_1, a, q_2 \rangle \in \Delta\), and \(|S| = 1\). In other words an automaton is said to be deterministic if it has a single initial state and its transition relation is functional. Because of these properties we’ll sometimes denote the deterministic automata as \(A = (\Sigma, Q, q_0, F, \delta)\), where \(q_0 \in Q\) is an initial state, and \(\delta : Q \times \Sigma \rightarrow Q\) is transition function.

Construction 4.1 (Determination). Let \(A = (\Sigma, Q, S, F, \Delta)\) be a FSA. From \(A\) we construct \(A^D = (\Sigma, Q^D, S^D, F^D, \Delta^D)\) by first constructing in parallel the sequences \(\{Q_i\}_{i=0}^\infty\) and \(\{\Delta_i\}_{i=0}^\infty\) (where \(Q_i \subseteq 2^Q\), and \(\Delta_i \subseteq 2^Q \times \Sigma \times 2^Q\)) following this inductive scheme:

- \(Q_0 = \{S\}\), \(\Delta_0 = \emptyset\)
- \(Q_{i+1} = Q_i \cup \{T \mid (\exists T' \in Q_i)(\exists a \in \Sigma) (T' \xrightarrow{a} T)\}\)
- \(\Delta_{i+1} = \Delta_i \cup \{T' \mid a, T T' \in Q_i \& T' \xrightarrow{a} T\}\)

It is immediately seen that both sequences are monotonously increasing and therefore converge to their least upper bounds. We define

- \(Q^D = \bigcup Q_i\)
- \(S^D = \{S\}\)
- \(F^D = \{T \mid T \in Q^D \& T \cap F \neq \emptyset\}\)
- \(\Delta^D = \bigcup \Delta_i\)

We say that \(A^D\) is received from \(A\) by determination.

Lemma 4.1. Let \(A\) is FSA and \(A^D\) is received from it by determination (Construction 4.1). Then \(A^D\) is deterministic.

Proof. We’ll verify that the relation \(\Delta^D\) is functional. Let’s assume that there exists \(T', T_1, T_2 \subseteq Q^D\) and \(a \in \Sigma\), such that \(\langle T', a, T_1 \rangle \in \Delta^D\), \(\langle T', a, T_2 \rangle \in \Delta^D\) and \(T_1 \neq T_2\). \(\Delta^D\) is the least upper bound of \(\{\Delta_i\}\) and therefore there exists \(n \geq 0\), such that \(\langle T', a, T_1 \rangle \in \Delta_n\) and \(\langle T', a, T_2 \rangle \in \Delta_n\), and it follows (by the definition of the sequence) that \(T' \xrightarrow{a} T_1 \& T' \xrightarrow{a} T_2\). In other words \(T_1 = \{q \mid (\exists T' \in Q^D)(\exists a, q)(q \in \Delta_i)\} = T_2\). Which is a contradiction and that’s how we showed that \(\Delta^D\) is functional relation.

Clearly \(|S^D| = |\{S\}| = 1\) and finally \(A^D\) is deterministic.

We denote the initial state of \(A^D\) with \(q_0^D = Q^D\), and the transition function with \(\delta^D : Q^D \times \Sigma \rightarrow Q^D\) (defined by its graph \(\Delta^D\)).

Lemma 4.2. Let \(A\) is a FSA, \(a \in \Sigma^*\) and \(q_0, q_1, \ldots, q_{|a|}\) is an execution of \(A\) over \(a\). Let \(A^D\) is received from \(A\) by determination. Then there exists an execution of \(A^D\): \(T_0, T_1, \ldots, T_{|a|}\), such that \(q_i \in T_i\) for each \(i = 0, 1, \ldots, |a|\).
Proof. The proof goes by induction on the length of $\alpha$.

- $|\alpha| = 0$. From $\alpha = \epsilon$ it follows that the execution of $A$ contains a single state $q_0 \in S$. The searched execution of $A^D$ also has a single state $T_0 = q_0^D = S$. Obviously $q_0 \in T_0$.

- $|\alpha| > 0$. Let $\alpha = \alpha'a$. By induction hypothesis the lemma should hold for $\alpha'$, and therefore there exists an execution $T_0, T_1, \ldots, T_{|\alpha'|}$ of $A^D$ over $\alpha'$, for which $q_i \in T_i$ for every $i = 0, \ldots, |\alpha'|$. As $|\alpha'| = |\alpha| - 1$ and $q_0, q_1, \ldots, q_{|\alpha'|} - 1, q_{|\alpha|}$ is an execution of $A$ over $\alpha'a$, it should be true that $(q_{|\alpha'|}, a, q_{|\alpha|}) \in \Delta$. Consequently, if $T_{|\alpha'|} \xrightarrow{\Delta, a} T$, then $q_{|\alpha|} \in T$. Let's now take from the construction of $A^D$ the least $i \geq 0$, such that $T_i \in Q_i$ (we are certain such execution exists because $T_{|\alpha'|} \in \cup Q_i$). Then on the next step of the construction $Q_{i+1} = Q_i \cup \{T_i\}$ for some $T_i \in \Delta_i$. Therefore $\delta^D(T_{|\alpha'|}, a) = T_i$ and $T_0, T_1, \ldots, T_{|\alpha'|}, T$ is an execution of $A^D$ over $\alpha$ complying with the requirements.

\[\square\]

Lemma 4.3. Let $A$ be a FSA and $A^D$ is received from it by determinization. Let $\alpha \in \Sigma^*$, $T_0, T_1, \ldots, T_{|\alpha|}$ is an execution of $A^D$ and $q \in T_{|\alpha|}$. There exists an execution of $A \rightarrow q_0, q_1, \ldots, q_{|\alpha|} = q$, such that $q_i \in T_i$ for every $i = 0, 1, \ldots, |\alpha|$.

Proof. The proof again follows induction on the length of $\alpha$.

- $|\alpha| = 0$. As $\alpha = \epsilon$, the execution of $A^D$ is a sequence of a single state $T_0 = q_0^D = S$. The sequence of the single state $q$ complies with the requirements because $q \in T_0 = S$.

- $|\alpha| > 0$. Let $\alpha = \alpha'a$. As $T_0, T_1, \ldots, T_{|\alpha'|}, T_{|\alpha|}$ is an execution of $A^D$, it follows that $(T_{|\alpha'|}, a, T_{|\alpha|}) \in \Delta$, and therefore $(T_{|\alpha'|}, a, T_{|\alpha|}) \in \Delta_n$ for some $n \geq 0$ ($\Delta$ is the least upper bound of the sequence $(\Delta_i)_{i=0}^{\infty}$). Let $m < n$ be the largest index such that $(T_{|\alpha'|}, a, T_{|\alpha|}) \notin \Delta_m$. Then on step $m + 1$ of the construction, this triple was added to $\Delta_m$, because $T_{|\alpha'|} \xrightarrow{\Delta_a} T_{|\alpha|}$. Now because $q \in T_{|\alpha|}$, there is $q' \in T_{|\alpha'|}$, such that $(q', a, q) \in \Delta$. By the induction hypothesis the lemma holds for $|\alpha'|$, which implies that there exists an execution $q_0, q_1, \ldots, q'$ of $A$ over $\alpha'$, complying with the requirements. We append $q$ and receive the execution of $A$ over $\alpha$ we were looking for.

\[\square\]

Proposition 4.1. Let $A$ be a FSA and $A^D$ is received from it by determinization. Then $A$ and $A^D$ are equivalent.
Proof. Let $\alpha \in \mathcal{L}(A)$. Then there exists a successful execution $q_0, q_1, \ldots, q_{|\alpha|} \in F$ of $A$ over $\alpha$. According to Lemma 4.2 there exists an execution $T_0, T_1, \ldots, T_{|\alpha|}$ of $A^D$, such that $q_i \in T_i$ for $i \in [0, |\alpha|]$. On the other hand $q_{|\alpha|} \in T_{|\alpha|}$ and $T_{|\alpha|} \cap F \neq \emptyset$, and therefore $T_{|\alpha|} \in F^D$, e.g. $A^D$ has a successful execution over $\alpha$ e.g. $\alpha \in \mathcal{L}(A^D)$.

Conversely, let $\alpha \in \mathcal{L}(A^D)$. This means that there exists a successful execution $T_0, T_1, \ldots, T_{|\alpha|}$ of $A^D$ over $\alpha$. On the other hand $T_{|\alpha|} \in F^D$ and that’s why $T_{|\alpha|} \cap F \neq \emptyset$, e.g. there exists $q \in T_{|\alpha|}$ which is a final state of $A$. Using Lemma 4.3 we conclude that there is an execution $q_0, q_1, \ldots, q$ of $A$ over $\alpha$, which shows to be successful ($q \in F$) and it follows that $\alpha \in \mathcal{L}(A)$. 

**Corollary 4.1.** For every FSA $A$ there exists a DFA $A^D$, such that $\mathcal{L}(A) = \mathcal{L}(A^D)$. 

14
5 Bimachines

Definition 5.1. We define a bimachine as

\[ B = (A_L, A_R, \psi) \]

where \( A_{L,R} = (\Sigma, Q_{L,R}, q_{L,R}, \delta_{L,R}) \) are deterministic finite state automata without any accepting states (respectively left and right), and \( \psi : Q_L \times \Sigma \times Q_R \rightarrow \Sigma^* \) is an output function.

The bimachines are abstract machines which work over input word (bidirectionally) and produce an output word based on executions of their automata and the input word. Their operational semantics is defined by the transitive closure of \( \psi \), \( \psi^* : Q_L \times \Sigma^* \times Q_R \rightarrow \Sigma^* \), defined as follows

- \( \psi^*(q_1, \epsilon, q_2) = \epsilon \)
- \( \psi^*(q_1, a\alpha, q_2) = \psi(q_1, a, \delta_R(q_2, \tilde{\alpha})) \cdot \psi^*(\delta_L(q_1, a), \alpha, q_2) \)

Based on the functional nature of the bimachines we’ll often use the \( B : \Sigma^* \rightarrow \Sigma^* \) notation, in which for any \( \alpha \in \Sigma^* \) we define as \( B(\alpha) = \psi^*(q_L, \alpha, q_R) \). We’ll say that for any particular word \( \alpha \in \Sigma^* \) over the bimachine’s input alphabet, \( B(\alpha) \) is the result from \( B \)’s execution over \( \alpha \).

Behind this long definition of bimachine’s execution result lies an intuitively simple strategy. We might assume that the bimachine reads its input word and for any character outputs a word over its alphabet. The result from the bimachine’s execution is the concatenation of all output words. On every step the output function decides on its output according to the current character and the two states which would have been reached respectively by the left and right automata exactly before they consume this same character.

![Figure 2: Execution of a bimachine](image)

It becomes clear that the so defined bimachines work only over their input word. Let’s examine a modified version of this strategy which only differs in the operational semantics used.

Definition 5.2. (Left) Output-driven bimachine is

\[ B = (A_L, A_R, \psi) \]

where \( A_{L,R} \) are again DFA, \( \psi \) is an output function and the operational semantics \( B \) of the bimachine is defined as \( B(\alpha) = \psi^{**}(q_L, \alpha, q_R) \), where
• \( \psi^*(q_1, \epsilon, q_2) = \epsilon \)

• \( \psi^*(q_1, a, q_2) = \psi(q_1, a, \delta_R(q_2, \alpha)) \cdot \psi^*(\delta_L^i(q_1, \psi(q_1, a, \delta_R(q_2, \alpha))), \alpha, q_2) \)

An important property of the output-driven bimachines is the fact that their left automaton doesn’t read the input word directly but the output produced by the output function instead. Absolutely symmetrically one might define a right output-driven bimachine.

Now we proceed to demonstrate that every output-driven bimachine can be simulated by an equivalent classical one (working purely over its input).

**Construction 5.1.** Let \( B \) is an output-driven bimachine. We define a FSA

\[
A_L^N = \langle \Sigma, Q_L \times Q_R, \{q_L\} \times Q_R, \Delta_L \rangle
\]

where the transition relation is defined as follows:

\[
\langle (p_1, q_1), a, (p_2, q_2) \rangle \in \Delta_L \iff \delta_R(q_2, a) = q_1 \land \delta_L^i(p_1, \psi(p_1, a, q_2)) = p_2
\]

Now we construct the left DFA \( A_L' \) by determinizing \( A_L^N \) (Construction 4.1). We take \( A_R' = A_R \) for right DFA of the bimachine and define the output function \( \psi' : Q_L^N \times \Sigma \times Q_R \) as follows:

\[
\psi'(L, a, r) = \beta \iff (\exists (p, q) \in L)(\delta_R(r, a) = q \land \psi(p, a, r) = \beta)
\]

Thus we completed the construction of \( B' = (A_L', A_R', \psi') \).

In order to be sure that \( B' \) is a classical bimachine, equivalent to \( B \), we should first check that it is correctly defined and that for any \( \alpha \in \Sigma^* \), \( B'(\alpha) = B(\alpha) \).

The correctness proof requires only to show that \( \psi' \) is actually a function.

**Proposition 5.1.** Let \( L \in Q_L' \) is a state of \( A_L' \) and \( (p_k, q_k) \in L(k = 1, 2) \). Then \( q_1 = q_2 \Rightarrow p_1 = p_2 \)

**Proof.** Let \( Q_{L_k} = \cup Q_i \) is the least upper bound of the sequence \( \{Q_i\}_{i=0}^{\infty} \) from Construction 4.1. We demonstrate a proof based on complete induction on the least index \( i \), such that \( L \in Q_i \) (\( i \) exists because \( Q_{L_k} \) is the least upper bound of the sequence).

\( i = 0 \) Obviously \( p_1 = p_2 = q_L \), because \( L = \{q_L\} \times Q_R \).

\( i + 1 \) Let \( i + 1 \) is the least index such that \( L \in Q_{i+1} \). Then (from Construction 4.1) there exists \( L' \in Q_i \), such that \( \delta_L^i(L', a) = L \) for some \( a \in \Sigma \). Let \( (p_k, q_k) \in L' \). Hence there exist \( \langle p_k^q, q_k^q \rangle \in L' \), such that \( \langle (p_k^q, q_k^q), a, (p_k, q_k) \rangle \in \Delta_L \) (for \( k = 1, 2 \)). Now let’s assume that \( q_1 = q_2 \). Therefore \( q_1' = \delta_R(q_1, a) = \delta_R(q_2, a) = q_2' \). Let \( i' < i + 1 \) is the least index such that \( L' \in Q_{i'} \). From the induction hypothesis for \( i' \) we deduce that \( p_1' = p_2' \). Finally, \( p_1 = \delta_L^i(p_1', \psi(p_1', a, q_1)) = \delta_L^i(p_2', \psi(p_2', a, q_2)) = p_2 \), which is what we wanted to show.

\( \square \)
Corollary 5.1. \( \psi' \) is correctly defined function

Proof. Let’s assume that \( \psi'(L, a, r) = \beta_{1,2} \). By the definition of \( \psi' \) it would follow that there exist \( (p_i, q_i) \in L \) for \( i = 1, 2 \), such that \( \delta_R(r, a) = q_i \), and additionally \( \psi(p_i, a, r) = \beta_2 \). But then \( q_1 = \delta_R(r, a) = q_2 \) and from Proposition 5.1 it follows that \( p_1 = p_2 \). This leads us to the fact that \( \beta_1 = \psi(p_1, a, r) = \psi(p_2, a, r) = \beta_2 \). Thus, we showed that \( \psi' \) is correctly defined function. \( \square \)

It remains to show that the constructed classical bimachine is equivalent to the output-driven bimachine.

Proposition 5.2. Let \( t = \alpha \beta \in \Sigma^* \), \( \delta_L^*(q_L', \alpha) = L \). Then

\[
(\delta_L^*(q_L, \psi^{**}(q_L, \alpha, \delta_R(q_R, \beta))), \delta_R(q_R, \beta)) \in L
\]

Proof. The proof uses a simple induction on \( \alpha \).

\( \alpha = \epsilon \) When \( \alpha = \epsilon \), by the definition of \( \psi^{**} \) we have that \( \psi^{**}(q_L, \alpha, \delta_R(q_R, \beta)) = \epsilon \) therefore \( \delta_L^*(q_L, \psi^{**}(q_L, \alpha, \delta_R(q_R, \beta))) = p_1 \). \( \delta_R(q_R, \beta) = r \), and \( \delta^*_R(r, a) = r_1 \). By induction hypothesis we might deduce that \( (p_1, r_1) \in L' \).

On the other hand \( L = \delta_L(L', a) \), and because \( \delta_R(r, a) = r_1 \) we show that \( (p_1, r_1, \delta^*_R(p_1, \psi(p_1, a, r)), r) \in \Delta_L \). Hence (by Construction 4.1) it follows that \( (\delta_L^*(p_1, \psi(p_1, a, r)), r) \in L \). Now by the definition of \( \psi^{**} \), \( \delta_L^*(p_1, \psi(p_1, a, r)) = \delta_L^*(q_L, \psi^{**}(q_L, \alpha, \delta_R(q_R, \beta))) \), which proves the proposition. \( \square \)

Corollary 5.2. For any \( \alpha \in \Sigma^* \), \( B(\alpha) = B'(\alpha) \)

Proof. Let’s start with the case when \( \alpha = \epsilon \). By the definitions of \( \psi \) and \( \psi' \) it follows that \( B(\alpha) = \psi^{**}(q_L, \alpha, q_R') = \psi^{**}(q_L', \alpha, q_R') = B'(\alpha) \).

If \( \alpha = a_1a_2\ldots a_n \) we should prove that for \( i = 1, 2, \ldots, n \) and \( r = \delta_R(q_R, a_1a_2\ldots a_{i-1}) \)

\[
\psi(\delta_L^*(q_L, \psi^{**}(q_L, a_1a_2\ldots a_{i-1}, \delta_R(r, a_i))), a_i, r) = \psi'(\delta_L^*(q_L', a_1a_2\ldots a_{i-1}), a_i, r)
\]

But this is exactly the case because according to the already proven Proposition 5.2 and the definition of \( \psi' \):

\[
(\delta_L^*(q_L, \psi^{**}(q_L, a_1a_2\ldots a_{i-1}, \delta_R(r, a_i))), \delta_R(r, a_i)) \in \delta_L^*(q_L', a_1a_2\ldots a_{i-1})
\]

This is how we showed that in every position of the input word \( B \) and \( B' \) output equal results. Hence \( B(\alpha) = B'(\alpha) \). \( \square \)
6 Direct Construction

Let $E \rightarrow \beta / L_R$ is a context-sensitive rewrite rule over finite alphabet $\Sigma$. We construct the finite automata $A_L$, $A_E$ and $A_R$, such that

$$
A_L = \mathcal{A}(\Sigma^* L) = (\Sigma, Q_L, S_L, F_L, \Delta_L), \quad \mathcal{L}(A_L) = \mathcal{L}(\Sigma^* L) \\
A_E = \mathcal{A}(E) = (\Sigma, Q_E, S_E, F_E, \Delta_E), \quad \mathcal{L}(A_E) = \mathcal{L}(E) \\
A_R = \mathcal{A}(R\Sigma^*) = (\Sigma, Q_R, S_R, F_R, \Delta_R), \quad \mathcal{L}(A_R) = \mathcal{L}(R\Sigma^*)
$$

Even though we haven’t fixed a particular construction of FSA from regular expression we assume the implicit requirement that $A_L$, $A_E$ and $A_R$ are in normal form.

![Figure 3: Finite automata in normal form constructed respectively from the regular expressions $\Sigma^*x$, $xy|yz$ and $z\Sigma^*$](image)

After we constructed $A_L$, $A_E$ and $A_R$, we concatenate them to obtain

$$
A = (\Sigma, Q, S, F, \Delta) = A_L \cup A_E \cup A_R
$$

![Figure 4: Finite automaton obtained by the concatenation $A = A_L \cup A_E \cup A_R$, from the rewrite rule $xy|yz \rightarrow \epsilon / x\_\_z$ from Example 2.2](image)

From the properties of the FSA’s normal form and the concatenation to the left/right we can deduce the following

**Property 6.1.** Every successful execution of $A$ over $t \in \Sigma^*$ is of the type

$$
q_0, q_t, \ldots, q_{|u|−1}, q_{e_1}, \ldots, q_{e_{|v|}}, q_{r_1}, q_{r_2}, \ldots, q_{r_{|w|}}
$$

where $(u, v, w) \in C(t; L, E, R)$, $q_0 \in S_E$, $q_{e_i} \in F_E$, $q_{r_{|w|}} \in F_R$ for every $i \in [0, |v|]$, $q_{e_i} \in Q_L$ for every $i \in [0, |u|]$, $q_{r_i} \in Q_R - S_R - F_R$ for every $i \in ([|u|, |vw|])$, and $q_{r_{|w|}} \in F_R$. We’ll call such an execution of $A$ — an execution for the context $(u, v, w)$. We denote $X(t; L, E, R) = \{\sigma|\sigma$ is an execution for some $(u, v, w) \in C(t; L, E, R)\}$
Property 6.2. For every context \( \langle u, v, w \rangle \in \mathcal{C}(t; L, E, R) \) there exists and execution \( \sigma \in \mathcal{X}(t; L, E, R) \) for \( \langle u, v, w \rangle \).

We'll denote \( \mathcal{X}_u(t; L, E, R) = \{ \sigma | \sigma \text{ is an execution for some } \langle u, v, w \rangle \in \mathcal{C}_u(t; L, E, R) \} \)

In order to construct the classical bimachine in question, we first construct an output-driven bimachine \( \mathcal{B} = (A_L, A_R, \psi) \), working only over its output. It will resolve any possible context ambiguities.

6.1 The Left Automaton

We extend the input alphabet \( \Sigma \) into \( \Sigma^c = \Sigma \cup \{ a^c | a \in \Sigma \} \). Informally speaking, \( \Sigma^c \) extends \( \Sigma \) by adding a cloned version of every character in \( \Sigma \). In order to construct the left automaton we first define

\[
A_1^N = (\Sigma^c, Q, S, F, \Delta_1^N)
\]

where \( \Delta_1^N = \Delta \cup \{ (q_1, a^c, q_2) | (q_1, q_2) \in \Delta \land q_2 \not\in S_E \} - Q \times \Sigma^c \times Q_R. \]

![Figure 5: Non-deterministic version \( A_1^N \) of the left automaton \( A_1 \), constructed from the rewrite rule \( xyyz \rightarrow \epsilon \) / \( xz \)](image)

Intuitively, \( A_1^N \) behaves exactly as \( A_L \cup A_E \) with the only difference that whenever a cloned character is consumed no transitions into initial states of \( A_E \) are allowed. Thus, no executions are possible for contexts, whose focus is about to be processed. On the other hand, the output function of the bimachine will be responsible to output cloned characters only when focuses of valid contexts are processed.

We then determinize \( A_1^N \) (Construction 4.1) and receive the left automaton \( A_1 \) of the bimachine.

6.2 The Right Automaton

We first reverse \( A \) to receive its mirror \( A_2^N = \hat{A} \). Then we construct the right automaton \( A_2 \) of the bimachine by determinizing \( A_2^N \) (Construction 4.1).

6.3 The Output Function

The output function \( \psi \) of the bimachine \( \mathcal{B} \) is defined as follows:

\[
\psi(L, a, R) = \begin{cases} 
  a^c & \text{if } \delta_1(L, a) \cap R \cap (Q_E - S_E - F_E) \neq \emptyset \\
  a & \text{if } \delta_1(L, a) \cap R \cap (Q_E - S_E - F_E) = \emptyset
\end{cases}
\]
This finishes the definition of $B = \langle A_L, A_R, \psi \rangle$ - an output-driven bimachine. Before we finish the construction of a bimachine for context-sensitive rewrite rule, let’s show several interesting properties of $B$.

**Lemma 6.1.** Let $t = \alpha \beta$, $\delta_2^c(q_2, \tilde{\beta}) = R$ and $\delta_1^c(q_1, \psi^*(q_1, \alpha, R)) = L$. Also let $\sigma$ is an execution of $A$ over $\alpha$, such that $\sigma(|\alpha|) \in Q_L$. Then $\sigma(|\alpha|) \in L$.

**Proof.** Induction on the length of $\alpha$.

$\alpha = \epsilon$ It is obvious that $L = \delta^c_1(q_1, \psi^*(q_1, \epsilon, R)) = \delta^c_1(q_1, \epsilon) = q_1 = S$. Hence $\sigma(|\alpha|) = \sigma(0) \in S = L$.

$\alpha = \alpha'a$ Since $\sigma(|\alpha|) \in Q_L$ and because of Property 6.1 we might assert that $\sigma(|\alpha'|) \in Q_L$. Let’s denote $R' = \delta_2(R, a)$ and $L' = \delta_1^c(q_1, \psi^*(q_1, \alpha', R))$. Then by the induction hypothesis it follows that $\sigma(|\alpha'|) \in L'$. As $\sigma$ is an execution of $A$, it is true that $\langle \sigma(|\alpha'|), a, \sigma(|\alpha|) \rangle \in \Delta$. It remains to note that $L = \delta_1^c(L', \psi(L', a, R))$, moreover $\sigma(|\alpha|) \notin S_E \cup Q_R$ and from the definition of $\delta_1$ we may conclude that $\sigma(|\alpha|) \in L$.

**Lemma 6.2.** Let $t = \alpha \beta$ and $\delta_2^c(q_2, \tilde{\beta}) = R$ and $\delta_1^c(q_1, \psi^*(q_1, \alpha, R)) = L$. Then if $q \in L$ then there exists $\sigma$ - an execution of $A$ over $\alpha$, such that $\sigma(|\alpha|) = q$.

**Proof.** Induction on $\alpha$:
\[ x = \epsilon \]

For simplicity, transitions going into \{9\} have been omitted.

<table>
<thead>
<tr>
<th>(R_0)</th>
<th>(R_1)</th>
<th>(R_2)</th>
<th>(R_3)</th>
<th>(R_4)</th>
<th>(R_5)</th>
<th>(R_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>9</td>
<td>(9.7)</td>
<td>(9.5)</td>
<td>(9.7,6)</td>
<td>(9.4)</td>
<td>(9.5,4)</td>
</tr>
</tbody>
</table>

Figure 7: The right automaton \(A_R\) constructed from the rewrite rule \(xy|yz \rightarrow \epsilon / x\_y\). For simplicity, transitions going into \{9\} have been omitted.

\[ \alpha = \epsilon \]

Then \(L = q_1 = S\). Now if \(q \in L\) we examine an execution \(\sigma\) consisting of a single state \(q\). \(\sigma\) is obviously an execution of \(A\) and \(\sigma(|\alpha|) = q\).

\[ \alpha = \alpha' a \]

Let \(\delta_2(R, a) = R'\) and \(\delta_1(q_1, \psi^{*+}(q_1, \alpha', R')) = L'\). Now \(L = \delta_1(L', \psi(L', a, R))\).

We notice that \(\psi(L', a, R) \in \{a, a^2\}\), and therefore \(L = \delta_1(L', \psi(L', a, R))\).

Let \(q \in L\). There exists \(q' \in L', \) such that \((q', \psi(L', a, R), q) \in \Delta^N\), hence by the definition of \(\Delta^N\) it follows that \((q', a, q) \in \Delta\). By the induction hypothesis it follows that there exists \(\sigma' - \) an execution of \(A\) over \(\alpha'\), such that \(\sigma'(|\alpha'|) = q'\). We define

\[ \sigma(i) = \begin{cases} \sigma'(i) & i \leq |\alpha'| \\ q & i = |\alpha| \end{cases} \]

It is easily shown that \(\sigma'\) is exactly the wanted execution of \(A\) over \(\alpha\).
Proposition 6.1. Let \( t = \alpha \beta \in \Sigma^* \), \( \delta_2(q_2, \tilde{\beta}) = R \), and \( \delta_1(q_1, \psi^*(q_1, \alpha, R)) = L \). Then for every \( q \in Q_E \), \( q \in L \cap R \iff (\exists \sigma \in \mathcal{X}) (\sigma([\alpha]) = q) \).

Proof. Before we proceed with the actual proof, let's verify that the proposition might be reduced to those \( q \in Q_E \) for which there exists an execution \( \sigma \in \mathcal{X} \), such that \( \sigma([\alpha]) = q \). Indeed, if \( q \in L \cap R \), then \( q \in L \) and by Lemma 6.2 an execution \( \sigma_L \) of \( \mathcal{A} \) over \( \alpha \) would exist, such that \( \sigma_L([\alpha]) = q \). On the other hand \( q \in R \) hence by Lemma 4.3 there exists an execution \( \sigma_R \) of \( \mathcal{A} \) over \( \beta \), such that \( \sigma_R([\beta]) = q \). Then \( \sigma \), defined as follows:

\[
\sigma(i) = \begin{cases} 
\sigma_L(i) & i \leq |\alpha| \\
\sigma_R(|t| - i) & i > |\alpha| 
\end{cases}
\]

is a successful execution of \( \mathcal{A} \) and therefore \( \sigma \in \mathcal{X} \).

The proof in the opposite direction is obvious because \( \mathcal{X}_v \subseteq \mathcal{X} \).

Let us now define the set

\[
A = \{ \langle \alpha, q \rangle | \alpha \subseteq t \land q \in Q_E \land (\exists \sigma \in \mathcal{X}) (\sigma([\alpha]) = q) \}
\]

and a relation "\( \prec \)" in \( A \), such that

\[
\langle \alpha_1, q_1 \rangle \prec \langle \alpha_2, q_2 \rangle \iff \alpha_1 \subset \alpha_2 \lor \alpha_1 = \alpha_2 \land q_1 \in (Q_E - S_E) \land q_2 \in S_E
\]

This is a partial ordering of \( A \) which turns \( A \) into a well-founded set.

We proof the proposition by induction on the structure of \( A \). Let's fix an element \( \langle \alpha, q \rangle \in A \) and assume that for any preceding element \( \langle \alpha', q' \rangle \in A \) ((\( \langle \alpha', q' \rangle \prec \langle \alpha, q \rangle \)) the proposition is valid. We show that it remains valid for \( \langle \alpha, q \rangle \).

\( \alpha = \epsilon \Rightarrow \) Let \( q \in L \cap R \). But \( \alpha = \epsilon \), therefore \( q = \sigma([\alpha]) = \sigma(0) \in S \cap Q_E \), and hence \( q \in S_E \). This means that there exists a context \( \langle \epsilon, \nu, \omega \rangle \in \mathcal{C} \) and \( \sigma \) is an execution for it. But \( \langle \epsilon, \nu, \omega \rangle \in \mathcal{C}_v \) (by the construction of \( \mathcal{C}_v \)), and therefore \( \langle \epsilon, \nu, \omega \rangle \in \mathcal{C}_v \), and \( \sigma \in \mathcal{X}_v \).

\( \Leftarrow \) Let \( \sigma \in \mathcal{X}_v \). Then \( q = \sigma([\alpha]) = \sigma(0) \in S \land q_1 = L \). On the other hand \( \tilde{\sigma} \) is an execution of \( \tilde{A} \) and according to Lemma 4.2 \( q \in \delta_2(q_2, \tilde{\beta}) = R \), which leads us to \( q \in L \cap R \).

\( \alpha = \alpha' \). Let's denote \( R' = \delta_2(R, a) \), \( L' = \delta_1(q_1, \psi^*(q_1, \alpha', R')) \).

\( \Rightarrow \) Let \( q \in L \cap R \). We have two options for \( q \):

- \( q \in Q_E - S_E \). Then (by Construction 4.1) there exists \( q' \in L' \), such that \( \langle q', a, q \rangle \in \Delta \) and hence \( q' = \delta_2(R, a) = R' \). By property 6.1 we may assert that \( q' \in Q_E \) hence \( q' \in L' \cap R' \cap Q_E \). The induction hypothesis for \( \langle \alpha', q' \rangle \) states that \( \sigma' \in \mathcal{X}_v \) and \( \sigma'([\alpha']) = q' \). As \( q \in R \) there exists an execution \( \sigma_R \) of \( \mathcal{A} \) over \( \beta \), such that \( \sigma_R(0) \in F \) and
\[ \sigma(\bar{\beta}) = q \text{ (Lemma 4.3).} \] Now we may define \( \sigma \) – a successful execution of \( \mathcal{A} \) as follows:

\[
\sigma(i) = \begin{cases} 
\sigma'(i) & \text{if } i < |\alpha| \\
\sigma_R(|t| - i) & \text{if } i \geq |\alpha| 
\end{cases}
\]

It is easily seen that if \( \sigma' \) was an execution for \( (u_0, v_0, w_0) \in \mathcal{C}_v \), then \( \sigma \) is an execution for \( (u_0, v, w) \in \mathcal{C} \). Now from Proposition 2.4 it follows that \( (u_0, v, w) \in \mathcal{C}_v \) is also a valid context, e.g. \( \sigma \in \mathcal{X}_v \).

\( q \in \mathcal{S}_E \) The first fact, we immediately notice is that \( \psi(L', a, R) \not\in (\Sigma^e - \Sigma) \) (this is true because if it weren’t one could possibly have stated that \( L \cap \mathcal{S}_E = \emptyset \)). As a direct consequence from the definition of \( \psi \) it follows that \( L \cap R \cap (Q_E - S_E - F_E) = \emptyset \). Now let \( \sigma \in \mathcal{X} \) is such that \( \sigma([\alpha]) = q \), and this is an execution for the context \( (u, v, w) \in \mathcal{C} \). We show that this context is a valid one \( (u, v, w) \in \mathcal{C}_v \). Indeed, let’s assume that \( (u, v, w) \not\in \mathcal{C}_v \). Then there certainly exists \( (u', v', w') \in \mathcal{C}_v \), such that \( u' \subseteq u \subseteq u'' \) (Proposition 2.3), and therefore there is an execution \( \sigma' \in \mathcal{X}_v \) for \( (u', v', w') \), such that \( \sigma'([\alpha]) \in Q_E - S_E - F_E \) (Property 6.1). Then \( \langle \alpha, \sigma'([\alpha]) \rangle \not\in \mathcal{C} \) and by the induction hypothesis \( \sigma'([\alpha]) \in L \cap R \), which leads us to \( L \cap R \cap (Q_E - S_E - F_E) \neq \emptyset \), which is a contradiction with our assumption that \( (u, v, w) \not\in \mathcal{C}_v \). Therefore \( \sigma \in \mathcal{X}_v \).

\( \Rightarrow \) Let \( \sigma \in \mathcal{X}_v \) and \( \sigma([\alpha]) = q \). As \( \sigma \) is a successful execution of \( \mathcal{A} \) over \( \alpha \beta \), there should exist another execution \( \sigma_R \) of \( \mathcal{A} \) over \( \bar{\beta} \) and \( \sigma_R([\bar{\beta}]) = q \). This means that \( q \in R \) (Lemma 4.2). It only remains to show that \( q \in L \).

We again examine two cases:

\( q \in Q_E - \mathcal{S}_E \) From Property 6.1 it follows that \( \sigma([\alpha']) \in Q_E \) and then by induction hypothesis one might deduce that \( \sigma([\alpha']) \in L' \cap R' \). Since \( q \not\in \mathcal{S}_E \cup Q_R \) whatever the value of \( \psi(L', a, R) \) is we’ll have \( q \in \delta_1(L', \psi(L', a, R)) = L \).

\( q \in \mathcal{S}_E \) Having in mind how the executions of \( \mathcal{A} \) look like (Property 6.1) we have that \( \sigma([\alpha']) \in Q_L \). Then by Lemma 6.1, \( \sigma([\alpha']) \in L' \). Let’s assume that \( q \not\in L \). This could only be possible if \( \psi(L', a, R) = \alpha' \) which on the other hand can happen only if \( \delta_1(L', a) \cap R \cap (Q_E - S_E - F_E) \neq \emptyset \). Let \( q_0 \) be an evidence for that \( (q_0 \in \delta_1(L', a) \cap R \cap (Q_E - S_E - F_E)) \). Therefore \( q_0 \in L \cap R \) and \( q_0 \in (Q_E - S_E - F_E) \). Since \( \langle \alpha, q_0 \rangle \not\in \mathcal{C} \) we can apply the induction hypothesis for \( \langle \alpha, q_0 \rangle \) and conclude that there exists \( \sigma_0 \in \mathcal{X}_v \) such that \( \sigma_0([\alpha]) = q_0 \). From Property 6.1 and the fact that \( q_0 \in (Q_E - S_E - F_E) \), we deduce that \( (u_0, v_0, w_0) \in \mathcal{C}_v \) and \( u_0 \subseteq \alpha \subseteq v_0 w_0 \). But if \( \sigma \) is an execution for \( (u, v, w) \not\in \mathcal{C}_v \) it easily seen that \( \alpha = u \) hence \( (u, v, w) \not\in \mathcal{C}_v \) (there exists a valid
context overlapping it and Proposition 2.2). This contradicts our assumption that \( q \not\in L \). Thus we proved that \( q \in L \cap R \).

\[
\]

6.4 Bimachine For Context-Sensitive Rewrite Rule

Finally, we construct \( B^f \) - a classical bimachine, working over its input, equivalent to \( B \). Based on \( B^f = (A_1', A_2', \psi^f) \) we construct the final bimachine for \( E \rightarrow \beta / \text{L}_R \):

\[
B' = (A_1', A_2', \psi')
\]

where \( A_1' = A_1, A_2' = A_2 \), and the only difference with \( B^f \) is the output function \( \psi' \), defined as follows

\[
\psi'(L, a, R) = \psi_1'(L, a, R) \cdot \psi_2'(L, a, R)
\]

where

\[
\psi_1'(L, a, R) =
\begin{cases}
\epsilon & \text{if } \langle L', \delta_2'(R, a) \rangle \in L \land L' \cap \delta_1'(R, a) \cap (Q_E - S_E - F_E) \neq \emptyset \\
\beta & \text{if } \langle L', \delta_2'(R, a) \rangle \in L \land L' \cap \delta_1'(R, a) \cap (S_E - F_E) \neq \emptyset \\
\beta a & \text{if } \langle L', \delta_2'(R, a) \rangle \in L \land L' \cap \delta_1'(R, a) \cap (S_E \cap F_E) \neq \emptyset \\
a & \text{otherwise}
\end{cases}
\]

\[
\psi_2'(L, a, R) =
\begin{cases}
\beta & \text{if } \langle L', R \rangle \in \delta_1'(L, a) \land L' \cap R \cap S_E \neq \emptyset \land R = \delta_2'(q_2) \\
\epsilon & \text{otherwise}
\end{cases}
\]

Lemma 6.3. Let \( t = \alpha \beta \in \Sigma^* \), \( \delta_1^*(q_1', \alpha) = L \), \( \delta_2^*(q_2', \beta) = R' \). Then \( \langle L', R' \rangle \in L \) and \( q \in L' \cap R' \cap Q_E \) iff there exists a context \( \langle u, v, w \rangle \in C_v(t; L, E, R) \) and execution for it \( \sigma \in \mathcal{X}_v(t; L, E, R) \), such that \( \sigma(\alpha) = q \).

Proof. Let \( \langle L', R' \rangle \in L \) and \( q \in L' \cap R' \cap Q_E \). Then from Propositions 5.1 and 5.2 it immediately follows that \( R' = \delta_2^*(q_2, \beta) \) and \( L' = \delta_1^*(q_1, \psi^* (q_1, \alpha, R')) \). Now since \( q \in L' \cap R' \cap Q_E \), from Proposition 6.1 we may deduce that there exists a valid context \( \langle u, v, w \rangle \in C_v \) and execution for it \( \sigma \in \mathcal{X}_v \), such that \( \sigma(\alpha) = q \).

The proof is analogous in the opposite direction.

Now we are ready to show that the so constructed bimachine actually works as expected, namely realizes substitution according to the context-sensitive rewrite rule \( E \rightarrow \beta / \text{L}_R \).

Proposition 6.2. Let \( E \rightarrow \beta / \text{L}_R \) is a rewrite rule and \( B' \) is the bimachine constructed from it. Let also \( \alpha \in \Sigma^* \) is a word over the rule’s alphabet \( \Sigma \). Then \( B'(\alpha) \) is exactly the result of the application of the context rule over \( \alpha \).
Figure 9. The left automaton of $B'$ for the rule $xy|yz \rightarrow \beta / x \_ z$. Incoming transitions of $L_0$ have been omitted for brevity

**Proof.** In the case when $\alpha = \epsilon$, by Definition 2.6 the result from the application of $E \rightarrow \beta / L \_ R$ is exactly $\epsilon = B'(\alpha)$.

Let $\alpha = a_1a_2 \ldots a_n$ where $a_i \in \Sigma$ for $1 \leq i \leq n$. Let $(\omega_1\pi_1)(\omega_2\pi_2)\ldots(\omega_n\pi_n)$ is the result from applying the rule defined by Definition 2.6. We'll show that for any $i \in [1, n]$ if $\delta_i^L(q'_1, a_1a_2 \ldots a_{i-1}) = L$ and $\delta_i^R(q'_2, a_na_{n-1} \ldots a_{i+1}) = R$ then $\psi_i(L, a_i, R) = \omega_i$ and $\psi_i^2(L, a_i, R) = \pi_i$.

According to $\psi_i$’s definition four cases have to be examined:

1. **Case I** Assume that $(L', \delta_i^L(R, a)) \in L \& L' \cap \delta_i^R(R, a) \cap (Q_E - S_E - F_E) \neq \emptyset$. Then by Lemma 6.3 there exists a valid context $(u, v, w) \in C_v$ and execution for it $\sigma \in X_v$ such that $\sigma(i - 1) \in (Q_E - S_E - F_E)$. Based on the type of the executions of $A$ (Property 6.1) we deduce that $|u| < i - 1 < |uv|$. This (according to Definition 2.6) means that $\omega_i = \epsilon$.

2. **Case II** Assume that $(L', \delta_i^L(R, a)) \in L \& L' \cap \delta_i^R(R, a) \cap (S_E - F_E) \neq \emptyset$. Then by Lemma 6.3 there exists a valid context $(u, v, w) \in C_v$ and execution for it $\sigma \in X_v$ such that $\sigma(i - 1) \in (S_E - F_E)$. Based on the type of the executions of $A$ (Property 6.1) we deduce that $|u| = i - 1 < |uv|$. This (according to Definition 2.6) means that $\omega_i = \beta$.

3. **Case III** Assume that $(L', \delta_i^L(R, a)) \in L \& L' \cap \delta_i^R(R, a) \cap (S_E \cap F_E) \neq \emptyset$. Then by
Figure 10: Output function of $B'$ for the rewrite rule $xy|yz \rightarrow \beta / x_\_z$

Lemma 6.3 there exists a valid context $\langle u, v, w \rangle \in C_v$ and execution for it $\sigma \in \mathcal{X}_v$, such that $\sigma(i-1) \in (S_E \cap F_E)$. Based on the type of executions of $\mathcal{A}$ (Property 6.1) we deduce that $|u| = i - 1$. Since haven’t fallen into case II it follows that there doesn’t exists $\langle u, v, w \rangle \in C_v$, such that $|v| \neq \epsilon$ and therefore (by Definition 2.6) $\omega_i = \beta a_i$.

IV case Being here means didn’t fall into any of the above cases. Now let’s assume there exists $\langle u, v, w \rangle \in C_v$, such that $|u| \leq i - 1 < |w|$ or $|u| = i - 1 = |w|$. Then there exists an execution for it $\sigma \in \mathcal{X}_v$, such that $\sigma(i - 1) \in Q_E - F_E$ or $\sigma | i - 1 \rangle \in E \cap F_E$. Then (by Lemma 6.3) $\langle L', \delta_2(R, a) \rangle \in L'$ and $\sigma(i - 1) \in L' \cap \delta_2(R, a) \cap Q_E - F_E$ or $\sigma(i - 1) \in L' \cap \delta_2(R, a) \cap S_E \cap F_E$, which contradicts to the fact that we didn’t fall into any of the above cases. Thus we conclude that $\omega_i = a_i$.

We showed that $\psi'_i(L, a_i, R) = \omega_i$ for $i = 1, 2, \ldots, n$. It remains to show that $\psi'_i(L, a_i, R) = \pi_i$ for any $i = 1, 2, \ldots, n$.

Let $\psi'_i(L, a_i, R) = \beta$. Then $\langle L', R \rangle \in \delta'_2(L, a)$ and $L' \cap R \cap S_E \neq \emptyset$. According to Lemma 6.3 there will exist $\langle u, v, w \rangle \in C_v$ and execution for it $\sigma \in \mathcal{X}_v$, such that $\sigma(i) \in S_E$. On the other hand $R = q_{2\_R}^F$ and because of $\mathcal{A}$’s normal form it follows that $\alpha = t$ and hence $\langle u, v, w \rangle = \langle \alpha, \epsilon, \epsilon \rangle \in C_v$. Now we can immediately conclude that $\pi_i = \beta$.

By analogy we show that if $\pi_i = \beta$ then $\psi'_i(L, a_i, R) = \beta$.

Thus we finished our proof that $\psi'(L, a_i, R) = \omega_i \pi_i$ and therefore $B'(\alpha)$ is exactly the result from the application of $E \rightarrow \beta / L_\_R \overline{a} \alpha$.

Example 6.1. Let’s run the bimachine constructed for $xy|yz \rightarrow B / x_\_z$ over the input word $xyz x z y z z x z z x z z x z z x z z x z z$. The execution of $A'_1$ will be $L_0, L_2, L_4, L_6, L_8, L_1, L_3, L_5, L_7, L_9, L_1, L_3, L_5, L_7, L_9, L_1$. The right automaton $A'_2$ will produce $R_0, R_2, R_4, R_6, R_8, R_1, R_3, R_5, R_7, R_9, R_1, R_3, R_5, R_7, R_9, R_1$. Reading the input in the reverse direction. According to Fig. 10 the output of $B'$ is exactly $xBzzBzz$.  

26
7  Algorithms

In this section we present some the most interesting algorithms we need for the
binachine construction.

The algorithms are all presented without proof of correctness as they follow
without any modifications the respective constructions shown earlier.

7.1 Concatenation to the Left

The algorithm lconcat for concatenation to the left corresponds to Definition
3.8.

```plaintext
subroutine lconcat(Automaton A, Automaton B)
    let C = new Automaton

    foreach state in A.states
        let cloned_state = C.add_state(state)
        if state.is_start then
            cloned_state.set_start(true)
        end if
    endforeach

    foreach state in B.states
        let cloned_state = C.add_state(state)
        if state.is_start and A.accepts("") then
            cloned_state.set_start(true)
        end if
        if state.is_final then
            cloned_state.set_final(true)
        end if
    endforeach

    foreach trans in A.transitions
        C.add_transition(trans.source, trans.char, trans.target)
        if trans.target.is_final then
            foreach state in B.start_states
                C.add_transition(trans.source, trans.char, state)
            endforeach
        end if
    endforeach

    foreach trans in B.transitions
        C.add_transition(trans.source, trans.char, trans.target)
    endforeach

    return C
end subroutine
```
7.2 Translation of an Output-Driven Bimachine

The following algorithm output_to_input corresponds to Construction 5.1 which constructs a classical input-driven bimachine from an equivalent output-driven one.

subroutine output_to_input(Bimachine B)
    let psi = B.output_function
    let AL = B.left_automaton
    let AR = B.right_automaton

    let AN = new Automaton
    let states = new Table
    foreach state1 in AL.states
        foreach state2 in AR.states
            states[state1][state2] = AN.add_state()
        endforeach
    endforeach

    foreach p1 in AL.states
        foreach q1 in AR.states
            foreach p2 in AL.states
                foreach q2 in AR.states
                    foreach a in alphabet
                        if AR.trans_function(q2, a) = q1 and
                           AL.trans_function(p1, psi(p1, a, q2)) = p2 then
                            AN.add_transition(states[p1][q1], a, states[p2][q2])
                        end if
                    endforeach
                endforeach
            endforeach
        endforeach
    endforeach

    let A1 = AN.build_deterministic()

    let new_psi = new Function
    foreach L in A1.states
        foreach a in alphabet
            foreach r in AR.states
                foreach p in AL.states
                    foreach q in AR.states
                        if states[p][q] in L and
                           AR.trans_function(r, a) = q then
                            new_psi.define(L, a, r, psi(p, a, r))
                        end if
                    endforeach
                endforeach
            endforeach
        endforeach
    endforeach
end foreach
end foreach
end foreach

return new Bimachine(A1, AR, new_psi)
end subroutine

7.3 Direct Construction of a Bimachine for Context-Sensitive Rewrite Rule

The algorithm construct_bimachine directly reflects the construction from section 6 and constructs a bimachine realizing some given rewrite rule.

routine construct_bimachine(L, E, R, word)
    let AL = new Automaton("\.*\+L")
    let AE = new Automaton(E)
    let AR = new Automaton(R+"\.*")

    let A = rconcate(lconcat(AL, AE), AR)
    let A2 = A.reverse().build_deterministic()
    let AN = A.clone()

    foreach tr in AN.trans
        if tr.target in AR.states then
            AN.remove_transition(tr)
        else
            if tr.target not in AE.start_states then
                AN.add_transition(tr.source, tr.char + alphabet.size, tr.target)
            end if
        end if
    endforeach

    let A1 = AN.build_deterministic()

    let psi = new Function
    let inter = AE.states - AE.start_states - AE.final_states
    foreach L in A1.states
        foreach R in A2.states
            foreach a in alphabet
                if A1.trans_function(L, a).intersect(R).intersect(inter) then
                    psi.define(L, a, R, a + alphabet.size)
                else
                    psi.define(L, a, R, a)
                end if
            end foreach
        end foreach
    endforeach

    let AN

end foreach
end foreach

let B = output_to_input(new Bimachine(A1, A2, psi))

let final_psi = new Function
let intermediate = AE.states - AE.start_states - AE.final_states
let startnonfinal = AE.start_states - AE.final_states
let startandfinal = AE.start_states.intersect(AE.final_states)
foreach L in B.left_automaton.states
  foreach R in B.right_automaton.states
    foreach a in alphabet
      let result = a
      foreach (p, q) in L
        continue unless q = B.right_automaton.trans_function(R, a)
        let common = p.intersect(q)
        if common.intersect(intermediate).length > 0 then
          result = ""
          break
        else if common.intersect(startnonfinal).length > 0 then
          result = word
          break
        else if common.intersect(startandfinal).length > 0 then
          result = word + a
          break
      end if
    if R = B.right_automaton.start_state then
      foreach (p, q) in B.left_automaton.trans_function(L, a)
        continue unless q = R
        if p.intersect(q).intersect(AE.start_states).length > 0 then
          result = result + word
        end if
      end foreach
    end if
  end if
final_psi.define(L, a, R, result)
end foreach
end foreach

B.output_function = final_psi

return B
end subroutine
8 Complexity

In this section we explore the upper bound of the constructed bimachine’s size.

Proposition 8.1. Let \( B = (\mathcal{A}_L, \mathcal{A}_R, \psi) \) is an output-driven bimachine and \( B' = (\mathcal{A}'_L, \mathcal{A}'_R, \psi') \) is an equivalent input-driven bimachine, constructed by Construction 5.1. Then

a) \( |Q'_R| = |Q_R| \)

b) \( |Q'_L| \leq (|Q_L| + 1)^{|Q_R|} \)

Proof. a) By Construction 5.1 we immediately deduce that \( \mathcal{A}_R = \mathcal{A}'_R \) and hence \( |Q'_R| = |Q_R| \) is trivially valid.

b) Because of Proposition 5.1 we may look at the states in \( Q'_L \) as if they were partial functions from \( Q_R \) to \( Q_L \). This means that \( |Q'_L| \) is no larger than the number of different partial functions defined from \( Q_R \) to \( Q_L \) therefore \( |Q'_L| \leq (|Q_L| + 1)^{|Q_R|} \).

Let’s now calculate the upper bound of the constructed bimachine’s space complexity. In other words we shall estimate the number of states in the bimachine’s automata in terms of the context rule.

Let’s fix a context rule \( E \rightarrow \beta / \_ R \) and denote \( l = |L|, e = |E| \) and \( r = |R| \). Then the number of states in \( \mathcal{A}_L \), \( \mathcal{A}_E \) and \( \mathcal{A}_R \) will be respectively \( O(l) \), \( O(e) \) and \( O(r) \). Hence it is easily seen that the number of states in \( \mathcal{A} \) is bounded by \( O(l + e + r) \).

This gives us an upper bound for the states of \( \mathcal{A}_2 \), namely \( O(2^{l+e+r}) \).

As \( \mathcal{A}_2 \) is obtained from \( \mathcal{A} \) through transition relation modifications, its number of states remains as much as \( O(l + e + r) \). This number grows exponentially to \( O(2^{l+e+r}) \) after the determinization of \( \mathcal{A}_1 \). Finally, according to Proposition 8.1, Construction 5.1 constructs \( \mathcal{A}_1 \) with \( O(2^{(l+e+r)(l+e+r)}) \) states.

Thus we obtained the final estimate of the bimachine’s space complexity – respectively \( O(2^n 2^n) \) and \( O(2^n) \) for the left and right automata, where \( n = l + e + r \) is the total length of the regular expressions in the context-sensitive rewrite rule.
References


