# On Reducing Automata and Their Normalizations

Martin Procházka

Abstract

The *reducing automaton*, a variant of the restarting automaton, is introduced and its normalizations are studied. They provide useful features such as *prefix-correctness* and *state minimality*. The LR(0) grammar generating the same language is constructed for any *monotone* reducing automaton. This grammar is used to construct an equivalent monotone reducing automaton that is *prefix-correct*. The minimization of a set of states of any reducing automaton using the method already invented in the theory of Moore machines is described. Both normalizations can be applied to the monotone reducing automaton sequentially so that the obtained automaton is both prefix-correct and state-minimal.

**Keywords**

reducing automata, LR(0) grammars, prefix-correctness, state-minimality

## 1. Introduction

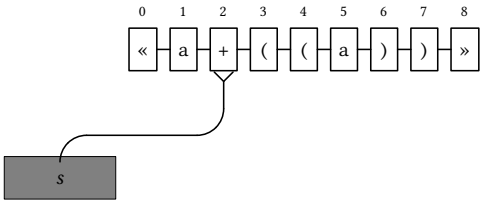
This paper is a revised translation of the author's dissertation thesis [1]. Some parts are reworded, some supplemented, others omitted.

The *reducing automaton* (first mentioned in [2]), like its predecessor and ideal, the *restarting automaton* (see [3]), is a device suitable for describing a syntax of both formal and natural languages. It is based on the notion of *reduction analysis*, i.e., a gradual truncation of the analyzed string that preserves both its incorrectness and correctness[1]. The following example of a gradual simplification of the arithmetic expression a + ( ( a ) ) shows that the reduction analysis is quite natural.

```
a + ( ( a ) )
a + (   a   )
a +     a
a
```

The reducing automaton differs from the restarting automaton in several details: 1. It lacks a fixed size lookahead window that is actually moved to the control unit. 2. The sequence of the head movement and the state change is reversed. 3. Positions of the reduced worklist items are precisely determined. It allows us to reuse some of the concepts and methods invented in standard grammar and automata theory and simplifies the techniques presented here: (a) construction of an LR(0) grammar that generates the same language as a given monotone reducing automaton (Section 3), (b) construction of an equivalent prefix-correct monotone reducing automaton, and (c) construction of a strongly equivalent state-minimal reducing automaton (Section 4). On the other hand the mentioned differences does not prevent us to adopt results reached for restarting automata as any reducing automaton can be simulated by a restarting automaton and vice versa while preserving (combination of) properties like determinism or monotony. Here we focus on *monotone* reducing automata, which recognize *deterministic context-free languages*, as shown for their predecessors, deterministic monotone restarting automata in [3].

**Figure 1:** Reducing automaton with a state $s$ in its control unit and a working head scanning an item of the working list at position 2 that contains a symbol +.




## 2. Basic notions and properties

The *reducing automaton* is shown in Figure 1. It processes a finite *list of items*. The first and the last item of the list contains the *delimiter* « and » respectively. Any other item contains a symbol of the finite *input alphabet* different from both delimiters and a natural number, the *position* of the item in the list. The positions of the items from left to right form an increasing sequence. The position of any item does not change during the computation. The reducing automaton resembles a finite-state automaton. It consists of a *control unit* and a *working head*. At any given time the control unit is in one of the finitely

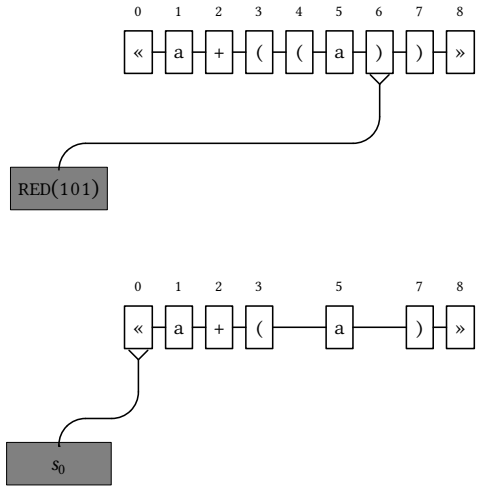
*ITAT'22: Information technologies – Applications and Theory, September 23–27, 2022, Zuberec, Slovakia*

martproc@gmail.com (M. Procházka)



[1] Restarting automata preserve correctness if they are deterministic. Reducing automata are deterministic by definition, as we shall see below.

**Figure 2:** Reducing the working list.




many states and the working head reads one item of the list. The automaton starts its computation in the *initial state* with the head placed on the very first item of the list containing the left delimiter «. At each step of the computation the automaton moves its working head to the right onto the next item and changes its state according to the current state and the symbol contained in the scanned item of the list. The new state is determined by a *transition function*.

Some states of the automaton are *final*, some others are *reducing*. The final states are ACC, the *accepting state*, and ERR the *rejecting state*. Reaching the ACC state means finishing the computation and accepting the word contained in the working list. By switching into the ERR state, the automaton terminates the computation by rejecting the word being processed. RED($n$), $n \in 1 \cdot (1|0)^*$, is the *reducing state*. Just after transition into this state, the automaton truncates the working list as prescribed by the *reducing sequence* $n$, moves its working head to the beginning of the working list, and transfers its control unit to the initial state. The reducing sequence determines items to be removed from the list in the following way: If its $i$-th symbol from the right equals to 1 then the automaton removes the $i$-th item from the list counting from the item under the working head to the left (counting the item under the working head as the first one). If it equals to 0 the corresponding item remains in the list. An example of truncation of the word « a+((a)) » by the reducing sequence 101 can be seen in the figure 2. Final state closes a certain *stage* of automaton computation. By stage, we mean any part of the computation that starts in the initial state with the head on the left delimiter and finishes in any final or reducing state.

Formally, we define the reducing automaton $M$ (red-automaton, for short) as a 7-tuple

$$M = (\Sigma, «, », S, s_0, F, f),$$

where $\Sigma$ is the finite *input alphabet*, «,» $\notin \Sigma$ is the *left* and *right delimiter* of an input word, respectively, $S$ is the finite set of *transition states*, $s_0 \in S$ is the *initial state*, $F$ is the finite set of *final* and *reducing states*, $f : S \times (\Sigma \cup \{»\}) \longrightarrow (S \cup F)$ is the *transition function*. Sets $F$ and $S$ are disjunctive. $F$ optionally contains accepting and rejecting final states ACC and ERR and finitely many reducing states of a form RED($n$), where $n$ is a sequence of 0 and 1 that begins with 1 ($n \in 1 \cdot (1|0)^*$). 0 means "*leave a symbol at a corresponding position in a working list*", 1 means "*delete a symbol at a corresponding position from a working list*". If $f(s, a) = s'$, we say that the automaton $M$ *moves* from the state $s$ over the symbol $a$ to the new state $s'$, or also that the transition function *switches the control unit* of the automaton from the state $s$ over the symbol $a$ to the new state $s'$.

We extend the transition function $f$ to the domain $(S \cup F \cup \{\text{RED}\}) \times (\Sigma^* \cdot \{\lambda, »\})$, where RED is the new auxiliary state different from all states from $S$ and $\lambda$ is an *empty word*. We mark the new extended function as $\delta$. The function $\delta$ equals to the transition function $f$ for all pairs $(s, a) \in S \times (\Sigma \cup \{»\})$. For any other pair from its domain it is defined as follows:

$$\delta(\text{ACC}, a) = \text{ACC} \qquad \delta(\text{RED}(n), a) = \text{RED}$$
$$\delta(\text{ERR}, a) = \text{ERR} \qquad \delta(\text{RED}, a) = \text{RED}$$

We define the reflexive and transitive closure $\delta^*$ of the function $\delta$ in the usual way. We consider only automata with the extended transition function satisfying the following conditions for any $s \in S$, $a \in \Sigma \cup \{»\}$, $s' \in S \cup F$, and $u \in \Sigma^* \cdot \{\lambda, »\}$:

$$\begin{aligned}
&\delta(s, ») = s' && \Longrightarrow\ s' \in F \\
&\delta(s, ») = \text{RED}(n) && \Longrightarrow\ 0 \text{ is the suffix of } n \\
&\delta(s, a) = \text{ACC} && \Longrightarrow\ a = » \\
&\delta^*(s_0, u) = \text{RED}(n) && \Longrightarrow\ |u| \geq |n|
\end{aligned}$$

*Characteristic constant* of the automaton $M$ is the length of the longest binary sequence contained in reducing states of the automaton $M$,

$$k(M) = \max\{|n| \mid \text{RED}(n) \in F\} \tag{1}$$

We describe the reduction of a word by a reduction sequence using the reduction operation / which we define as follows:

$$a/0 = a \qquad \lambda/n = \lambda \qquad (u \cdot a)/\lambda = u \cdot a$$
$$a/1 = \lambda \qquad u/\lambda = u \qquad (u \cdot a)/(n \cdot i) = (u/n) \cdot (a/i)$$

where $u \in \Sigma^*$, $a \in \Sigma^* \cup \{\text{»}\}$, $n \in (10^*)^*$ and $i \in \{0, 1\}$. Here we restrict neither the word $u$ nor the reduction sequence $n$ by any constant, $u$ can be longer than $n$ and vice versa. The truncation of the words (a), +a» by the reduction sequences 101, 110 respectively looks like this:

| | ( | a | ) |
|---|---|---|---|
| / | 1 | 0 | 1 |
| = | | a | |

| | + | a | » |
|---|---|---|---|
| / | 1 | 1 | 0 |
| = | | | » |

Based on the reduction operation, we introduce *reduction relation*. This allows us to describe how the automaton successively rewrites the processed word $w \in \Sigma^*$. The reducing automaton $M$ *reduces* the word «$w$» to the word «$w'$»,

$$\text{«}w\text{»} \Rightarrow_M \text{«}w'\text{»},$$

if $\delta^*(s_0, u) = \text{RED}(n), w\text{»} = uv, w'\text{»} = (u/n)\cdot v$ for some $u \in \Sigma^* \cdot \{\lambda, \text{»}\}$. Obviously, $|w| > |w'|$ and the above reduction is *shortening*. We refer to the reflexive and transitive closure of the reduction relation as $\Rightarrow_M^*$. By *reduction analysis* of the automaton $M$ we mean any sequence of reductions

$$\text{«}w_1\text{»} \Rightarrow \text{«}w_2\text{»} \Rightarrow \ldots \Rightarrow \text{«}w_n\text{»},$$

which cannot be extended further. If $\delta^*(w_n\text{»}) = \text{ACC}$, we are talking about an *accepting* reduction analysis, otherwise it is a *rejecting* reduction analysis. We often use the shorter term *analysis* instead of reduction analysis.

The words over the alphabet $\Sigma$ accepted by the reducing automaton $M$ in one stage form its *simple language*

$$L_0(M) = \{w \in \Sigma^* \mid \delta^*(s_0, w\text{»}) = \text{ACC}\}.$$

It is obvious that $L_0(M)$ is regular. We define the language *accepted* or also *recognized* by the reducing automaton $M$ as the set of all words over the alphabet $\Sigma$ for which there exists an accepting analysis of the automaton $M$,

$$L(M) = \{w \in \Sigma^* \mid \text{«}w\text{»} \Rightarrow_M^* \text{«}w'\text{»} \in \text{«}L_0(M)\text{»}\}.$$

Let us assume that $M_1 = (\Sigma_1, \text{«}, \text{»}, S_1, s_1, F_1, f_1)$ and $M_2 = (\Sigma_2, \text{«}, \text{»}, S_2, s_2, F_2, f_2)$ are any reducing automata. We say that $M_1$ and $M_2$ are *equivalent*, if

$$L(M_1) = L(M_2).$$

If for any $w, w' \in (\Sigma_1^* \cup \Sigma_2^*)$

$$L_0(M_1) = L_0(M_2) \quad \text{and}$$
$$\text{«}w\text{»} \Rightarrow_{M_1} \text{«}w'\text{»} \iff \text{«}w\text{»} \Rightarrow_{M_2} \text{«}w'\text{»},$$

then these automata are *reductionally equivalent*. They are *strongly equivalent* if for any prefix «$u$ of the word «$w$», the following holds

$$\delta_1^*(s_1, u) = \text{RED}(n) \iff \delta_2^*(s_2, u) = \text{RED}(n)$$
$$\delta_1^*(s_1, u) = \text{ACC} \iff \delta_2^*(s_2, u) = \text{ACC}$$
$$\delta_1^*(s_1, u) = \text{ERR} \iff \delta_2^*(s_2, u) = \text{ERR}$$

Using the reduction relation, we can express a basic property of reducing automata called *error and correctness preserving property*.

**Lemma 2.1.** *If* «$w_1$» $\Rightarrow_M$ «$w_2$», *then* $w_1 \in L(M)$, *iff* $w_2 \in L(M)$.

*Proof.* Let's assume that «$w_1$» $\Rightarrow_M$ «$w_2$». 1. If $w_2 \in L(M)$, then «$w_2$» $\Rightarrow_M^*$ «$w$» $\in$ «$L_0(M)$» for some $w$ and «$w_1$» $\Rightarrow_M$ «$w_2$» $\Rightarrow_M^*$ «$w$» is the accepting analysis for the word $w_1$. 2. If $w_1 \in L(M)$, then «$w_1$» $\Rightarrow_M^*$ «$w$» $\in$ «$L_0(M)$» for some $w$. This analysis starts by «$w_1$» $\Rightarrow_M$ «$w_2$», otherwise $M$ wouldn't be deterministic. So, we have «$w_2$» $\Rightarrow_M^*$ «$w$» $\in$ «$L_0(M)$». □

Similarly to [3], we introduce *monotony* for reducing automata. A reducing automaton is *monotone* reducing automaton (mon-red-automaton, for short), if – at any stage – it visits all the items visited at the previous stage that remain in the working list. Formally, the reducing automaton $M$ is monotone if the following condition is satisfied for each $w \in \Sigma^* \cdot \{\lambda, \text{»}\}$:

$$\delta^*(s_0, w) = \text{RED}(n) \implies \delta^*(s_0, (w/n)) \notin \{\text{ERR}, \text{RED}\}$$

Monotony is important for characterizing the deterministic context-free languages (DCFL for short). Deterministic monotone restarting automata (det-mon-R-automata) recognize just all deterministic context-free languages. In the same way, we can characterize the DCFL class using mon-red-automata. An R-automaton differs from a red-automaton only in that its working head is extended by a so-called lookahead window, which scans a continuous subword of fixed length to the right of the working head, and that it reduces the symbols scanned by the working head and the lookahead window. Thus, any R-automaton can be simulated by a red-automaton that stores the contents of the lookahead window in its control unit. Conversely, any red-automaton $M$ can be simulated by an R-automaton with a lookahead window of size $k(M)$. It is obvious that at each stage the simulated and simulating automaton visit the same items in the working list, so that monotony is preserved.

*Representation.* A red-automaton can be represented by a transition table in the same way as a finite-state machine. A transition table of a red-automaton recognizing a language $\{a^n b^n \mid n \geq 0\}$ is shown in Table 1. Each state encodes a regular expression mentioned in the first column of the Table. So, a lookahead window of R-automata is in case of red-automata moved into their states.

**Table 1**
Transition table.

| | $f$ | a | b | » |
|---|---|---|---|---|
| → | $s_0 = «$ | $s_1$ | ERR | ACC |
| | $s_1 = «a^*a$ | $s_1$ | $s_2$ | ERR |
| | $s_2 = «a^*ab$ | ERR | RED(110) | RED(110) |

## 3. Grammars

For any mon-red-automaton $M$, we construct a grammar $G_M$ whose derivation trees correspond closely to computations of a given automaton $M$. We show that this grammar generates the language recognized by the automaton $M$ and that it is an LR(0) grammar.

Let's assume that $M = (\Sigma, «, », S, s_0, F, f)$ is a mon-red-automaton. The grammar of the automaton $M$ is the grammar $G_M$, which is obtained by reducing the grammar

$$G = \big(V, N, S, P\big),$$

where $V = \Sigma \cup \{«, »\}$ is the *set of terminals*; $N$ is the *set of nonterminals* containing 5-tuples $(a, s, u, o, v)$, where $a \in \Sigma \cup \{«, »\}$, $s \in S \cup (F \setminus \{\text{ERR}\})$, $u \in \{\lambda, «\} \cdot \Sigma^* \cdot \{\lambda, »\}$, $|u| \le k(M)$, $v \in \Sigma^* \cdot \{\lambda, »\}$, $|v| \le k(M)$, and $o \in \{\text{ACC}\} \cup \{\text{RED}(n) \mid \exists n' : \text{RED}(nn') \in F\}$; $S = («, s_0, \lambda, \text{ACC}, \lambda) \in N$ is the *initial nonterminal*; $P$ is the *set of rules* defined as follows:

$$\begin{aligned}
X \to aY \in P, \quad & \text{if} \quad X = (a, s, \lambda, \text{RED}(n), (b/i) \cdot v) \\
& \qquad\; Y = (b, f(s,b), \lambda, \text{RED}(n \cdot i), v) \\
& \text{or} \quad X = (a, s, \lambda, \text{ACC}, \lambda) \\
& \qquad\; Y = (b, f(s,b), \lambda, \text{ACC}, \lambda), \\
X \to Y \in P, \quad & \text{if} \quad X = (a, s, au, \text{RED}(n), (b/i) \cdot v) \\
& \qquad\; Y = (b, f(s,b), u, \text{RED}(n \cdot i), v) \\
& \text{or} \quad X = (a, s, au, \text{ACC}, \lambda) \\
& \qquad\; Y = (b, f(s,b), u, \text{ACC}, \lambda), \\
X \to aYZ \in P, \quad & \text{if} \quad X = (a, s, \lambda, o, v) \\
& \qquad\; Y = (b, f(s,b), \lambda, \text{RED}(1), x) \\
& \qquad\; Z = (a, s, ax, o, v), \\
X \to YZ \in P, \quad & \text{if} \quad X = (a, s, au, o, v) \\
& \qquad\; Y = (b, f(s,b), u, \text{RED}(1), x) \\
& \qquad\; Z = (a, s, ax, o, v), \\
X \to a \in P, \quad & \text{if} \quad X = (a, \text{RED}(n), \lambda, \text{RED}(n), \lambda) \\
& \text{or} \quad X = (a, \text{ACC}, \lambda, \text{ACC}, \lambda), \\
X \to \lambda \in P, \quad & \text{if} \quad X = (a, \text{RED}(n), a, \text{RED}(n), \lambda) \\
& \text{or} \quad X = (a, \text{ACC}, a, \text{ACC}, \lambda),
\end{aligned}$$

for any $s \in S \cup (F \setminus \{\text{ERR}\})$, $u \in \{\lambda, «\} \cdot \Sigma^* \cdot \{\lambda, »\}$, $|u| \le k(M)$, $v \in \Sigma^* \cdot \{\lambda, »\}$, $|v| \le k(M)$, $n \in (10^*)^+$, $i \in \{0, 1\}$, $a \in \Sigma \cup \{«\}$, and $b \in \Sigma \cup \{»\}$. $G$ (and therefore $G_M$ as well) is obviously a context-free grammar.

**Table 2**
Graphical representation of grammar rules.

| rule | classical representation | our representation |
|---|---|---|
| $X \to aY$ | $X$<br>↙ ↘<br>$a$ $Y$ | $a$<br>↑<br>$X \to Y$ |
| $X \to Y$ | $X$<br>↓<br>$Y$ | $X \to Y$ |
| $X \to aYZ$ | $X$<br>↙↓↘<br>$a$ $Y$ $Z$ | $a$<br>↑<br>$X \to Y$<br>↓<br>$Z$ |
| $X \to YZ$ | $X$<br>↙ ↘<br>$Y$ $Z$ | $X \to Y$<br>↓<br>$Z$ |
| $X \to a$ | $X$<br>↓<br>$a$ | $X$<br>↑<br>$a$ |
| $X \to \lambda$ | $X$<br>↓<br>$\lambda$ | $X$ |

We draw derivation trees of grammar $G_M$ differently than we are used to for context-free grammars. We start with the initial nonterminal $S_0$ at the top right and expand each nonterminal as indicated in Table 2. For example the rule $X \to aYZ$ is drawn so that its right hand side is written clockwise around the left hand side nonterminal $X$, the terminal $a$ at 12 o'clock, the nonterminal $Y$ at 3 o'clock and the nonterminal $Z$ at 6 o'clock. This yields derivation tree pictures in which each maximum nonterminal branch resembles a working list visited by the automaton during the stage of its computation. When drawing the derivation tree we prevent the edges from crossing. We omit $\lambda$-rules.

Suppose $w \Rightarrow_M w'$ for some words $w$ and $w'$. We show how to move from a derivation tree that gives the word $w$ to a derivation tree that gives the word $w'$. First, we introduce the following notation: Let

$$X = (a, s, u, o, v)$$

be any nonterminal of the grammar $G_M$. Then

$$X^\lambda = (a, s, \lambda, o, v).$$

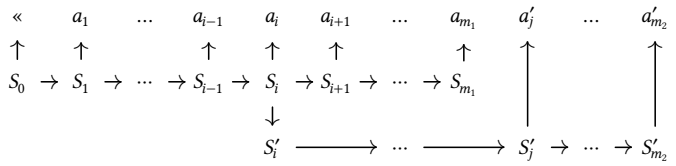
The following implications follow directly from the definition of the rules of the grammar $G_M$, where $a$ is a symbol in the first component of $X$ and $\omega \in N \cup \{\lambda\}$:

$$X \in V \implies X^\lambda \in V \tag{2}$$

$$X \to aYZ \in P \implies X = Z^\lambda \tag{3}$$

$$X \to Y\omega \in P \implies X^\lambda \to aY^\lambda\omega \in P \tag{4}$$

**Figure 3:** Function $R$ applied to a derivation tree $T$ with the first two branches shown in Fig. 4a. The function $R$ transforms these two branches into the single one shown in Fig. 4b, the rest of tree keeping unchanged.



(a) the first two maximum nonterminal paths of the original tree $T$

(b) the first maximum nonterminal path of the resulting tree $R(T)$

$$X \to \lambda \in P \implies X^\lambda \to a \in P \tag{5}$$

$$X \to aY\omega \in P \implies X = X^\lambda \text{ and } Y = Y^\lambda \tag{6}$$

Next, we introduce a function $R$ that assigns a tree $R(T)$ to any derivation tree $T$ of the grammar $G_M$ with at least two maximum nonterminal paths. We show that the resulting tree is a derivation tree of the grammar $G_M$. Suppose that the first two maximum nonterminal paths of the tree $T$, together with terminals they generate, are drawn in Figure 4a. Note that the vertex denoted by the nonterminal $S_i$ is the branching point of these nonterminal paths. We construct the tree $R(T)$ in the following way:

1. Remove the terminal vertices $a_i$, ..., $a_{m_1}$ and $a'_j$, ..., $a'_{m_2}$.
2. Remove the vertices $S_i, \dots, S_{m_1}$.
3. For each $l \in \{i, \dots, m_2\}$ do the following:
   a) replace the vertex $S'_l$ with the vertex $S'^{\lambda}_l$
   b) add the terminal vertex $a'_l$, where $a'_l$ is the symbol in the first component of the nonterminal $S'^{\lambda}_l$
   c) add an edge from $S'^{\lambda}_l$ to $a'_l$
4. Add a horizontal edge from the vertex $S_{i-1}$ to the vertex $S'^{\lambda}_l$.

Function $R$ converts the first two maximum nonterminal paths of the tree $T$ (including the terminals they generate) in Figure 4a to the path drawn in Figure 4b.

**Lemma 3.1.** *For every derivation tree $T$ of the grammar $G_M$ that lies in the domain of the function $R$ (contains at least two maximal nonterminal paths), the following statements hold:*

*(i) $R(T)$ is the derivation tree of the grammar $G_M$,*

*(ii) $R(T)$ contains one less maximal nonterminal path than $T$,*

*(iii) if $T$ gives the word $w$ and $R(T)$ gives the word $w'$, then $w \Rightarrow_M w'$.*

*Proof.* Suppose that the first two maximum nonterminal paths of the tree $T$, together with the generated terminals, look like paths depicted in Figure 4a.

*(i).* It follows from (2) that the vertices $S'^{\lambda}_i$, ..., $S'^{\lambda}_j$, ..., $S'^{\lambda}_{m_2}$ in Figure 4b are nonterminals of the grammar $G_M$.

The following rules are used in the first maximal nonterminal path of the tree $T$ for some $\omega \in V \cup \{\lambda\}$:

$$\begin{aligned} S_{i-1} &\to a_{i-1}\, S_i\, \omega \\ S_i &\to a_i\, S_{i+1}\, S'_i \end{aligned}$$

According to (3), $S_i = S'^{\lambda}_i$. So

$$S_{i-1} \to a_{i-1}\, S'^{\lambda}_i\, \omega$$

is a rule of grammar $G_M$. The following rules are used in the second maximal nonterminal path of the tree $T$ for some $\omega_{i+1}$, ..., $\omega_{m_2} \in V \cup \{\lambda\}$:

$$\begin{aligned} S'_i &\to S'_{i+1}\, \omega_{i+1} \\ &\vdots \\ S'_{j-1} &\to S'_j\, \omega_j \\ S'_j &\to a'_j\, S'_{j+1}\, \omega_{j+1} \\ &\vdots \\ S'_{m_2-1} &\to a'_{m_2-1}\, S'_{m_2}\, \omega_{m_2} \\ S'_{m_2} &\to a'_{m_2} \end{aligned}$$

According to (4), (5) and (6) the following rules

$$\begin{aligned} S'^{\lambda}_i &\to a'_i\, S'^{\lambda}_{i+1}\, \omega_{i+1} \\ &\vdots \\ S'^{\lambda}_{j-1} &\to a'_{j-1}\, S'^{\lambda}_{j}\, \omega_{j} \\ S'^{\lambda}_j &\to a'_j\, S'^{\lambda}_{j+1}\, \omega_{j+1} \\ &\vdots \\ S'^{\lambda}_{m_2-1} &\to a'_{m_2-1}\, S'^{\lambda}_{m_2}\, \omega_{m_2} \\ S'^{\lambda}_{m_2} &\to a'_{m_2} \end{aligned}$$

are rules of the grammar $G_M$. The tree $R(T)$ is therefore the derivation tree of the grammar $G_M$.

*(ii).* The construction of the tree $R(T)$ from the tree $T$ involves the removal of the suffix of the first nonterminal path just behind the branching point with the second nonterminal path. All other nonterminals are eventually replaced by other nonterminals, and all other edges between nonterminals in the tree remain unchanged. Thus, the derivation tree $R(T)$ contains one less maximal nonterminal path than the derivation tree $T$.

*(iii).* If the tree $T$ gives the word $w$ and its first two maximum nonterminal paths are shown in Figure 4a, then obviously

$$w = a_0 a_1 \dots a_i a_{i+1} \dots a_{m_1} x$$

for some $x$. The nonterminal $S_i$ is the last common nonterminal of the first two nonterminal paths of the tree $T$ and is rewritten to $a_i\, S_{i+1}\, S'_i$ using the grammar $G_M$ rule. The definition of the grammar $G_M$ rule for each $l \in \{i, \dots, m_1 - 1\}$ implies that $s_{l+1} = f(s_l, a_{l+1})$, where $s_l$ is the state contained in the second component of the nonterminal $S_l$, $s_{m_1} = \text{RED}(n)$ for some reduction sequence $n$ is the state contained in the second component of the nonterminal $S_{m_1}$,

$$\begin{array}{c} a_i \\ \uparrow \\ (a_i, s_i, \lambda, o, v) = S_i \longrightarrow S_{i+1} = (a_{i+1}, s_{i+1}, \lambda, \text{RED}(1), u) \\ \downarrow \\ (a_i, s_i, a_i \cdot u, o, v) = S'_i \end{array}$$

for some words $u$ and $v$ and some operation $o$, and $u = (a_{i+1} \dots a_{m_1})/n = a'_{i+1} \dots a'_{j-1}$. Together we get that $w' = a_0 a_1 \dots a_i u x$, and hence $w \Rightarrow_M w'$. □

**Lemma 3.2.** *For any derivation tree $T$ of the grammar $G_M$ and any words «w» and «w'», such that $T$ gives the word «w» and «w'» $\Rightarrow_M$ «w», there is a derivation tree $T'$ of the grammar $G_M$ that gives the word «w'» and $R(T') = T$.*

*Proof.* If «w'» $\Rightarrow_M$ «w», then for some $u$, $v$, and $n$ the word $uv$ is a prefix of w'», $\delta^*(s, uv) = \text{RED}(n)$, and $|v| = |n|$. Let $S_0 \to S_1 \to \dots \to S_m$ be the first maximal nonterminal path of the derivation tree $T$ and the first components of nonterminals on this path contain the terminal symbols «, $a_0$, $a_1$, ..., $a_m$. Since the automaton $M$ is monotone, $m \geq |uv/n| = l$ and $uv/n = a_1 \dots a_k \dots a_l$, where $k = |u|$.

The tree $T'$ is obtained from the tree $T$ in the following way:

1. Remove the terminal symbols $a_k$, ..., $a_l$.
2. Replace each nonterminal $S_j$, $k \leq j \leq l$, on the first nonterminal path with the nonterminals $S'_j$, which we get from $S_j$ by replacing the empty word $\lambda$ in the third component with the word $a_j \dots a_l$.
3. Add new nonterminals

$$\begin{aligned} S''_k &= (a_k, s_k, \lambda, o, v) = S_k, \\ S''_{k+i} &= (b_i, s_{k+i}, \lambda, \text{RED}(n'_i), v_i/n_i), \end{aligned}$$

   where $b_i$ is the $i$-th symbol of the word $v$ from the left, $v'_i$ is the prefix of the word $v$ of length $i$, $v'_i v_i = v$, $n'_i$ is the prefix of the reduction sequence of length $i$, $n'_i n_i = n$ and $s_{k+i} = \delta(s_k, v'_i)$ for all $i \in \{1, \dots, |v|\}$.
4. Add a terminal $a_k$ and an edge from the $S''_k$ terminal to this terminal. Similarly, we add the terminals $b_1$, ..., $b_{|v|}$ and connect them to the new derivation tree with an edge leading along the line from the $S''_{k+1}$, ..., $S''_{k+|v|}$.
5. We join the nonterminal $S''_k$ to the new derivation tree by an edge leading from the nonterminal $S_{k-1}$. We use the grammar rules $S_{k-1} \to a_{k-1}\, S''_k S'_k$. Similarly, we join the nonterminals $S''_{k+1}$, ..., $S''_{k+|v|}$ to the derivation tree using the rules $S''_k \to a_k\, S''_{k+1}$, ..., $S''_{k+|v|-1} \to a_{k+|v|-1}\, S''_{k+|v|}$.

The above construction yields a derivation tree $T'$ which gives the word «w'» and $R(T') = T$. We can check the correctness of this construction using the definition of nonterminals and the rules of the grammar $G_M$. □

**Lemma 3.3.** *Let $p$ be any natural number, $T_1$, ..., $T_p$ be any derivation trees of the grammar $G_M$, and $w_1$, ..., $w_p$ any words from « $\Sigma^*$ ». If at the same time*

- *$T_i = R(T_{i+1})$ for all $i \in \{1, \dots, p-1\}$,*
- *$T_i$ gives the word $w_i$ for all $i \in \{1, \dots, p\}$,*
- *$T_1$ contains exactly one maximal nonterminal path,*

*then at the same time*

- *$w_{i+1} \Rightarrow_M w_i$ for all $i \in 1, \dots, p-1$,*

- $w_1$ *is accepted by the M automaton in a single stage.*

*Proof.* We prove the theorem by induction on the natural number $p$. Obviously, $T_i$ contains just $i$ maximal nonterminal paths.

$p = 1$. Let $T_1$ be the derivation tree of the grammar $G_M$, which gives the word $w_1$ and contains exactly one maximal nonterminal path

$$S_0 \longrightarrow S_1 \longrightarrow \cdots \longrightarrow S_{i-1} \longrightarrow S_i \longrightarrow \cdots \longrightarrow S_m.$$

This means that no rule with two nonterminals on the right-hand side is used in this tree. Thus, only the following rules are used:

$$S_{i-1} \to a_i \mid a_i\, S_i \mid S_i \mid \lambda$$

The word in the third component of the nonterminal $S_i$ is empty (for all $i \in \{1, \dots, m\}$) because it is empty in the initial nonterminal $S_0$ and for all rules $S_{i-1} \to a_i\, S_i \mid S_i$, the word in the third component of the nonterminal $S_i$ is not longer than the word in the third component of the nonterminal $S_{i-1}$. Thus, only the following rules are used in the tree $T_1$:

$$S_{i-1} \to a_i \mid a_i\, S_i$$

and the calculation tree $T_1$ for the word $w_1$ looks like this:

$$\begin{array}{ccccccccc} « & & a_1 & \dots & a_{i-1} & & a_i & \dots & a_m \\ \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\ S_0 & \longrightarrow & S_1 & \longrightarrow \cdots \longrightarrow & S_{i-1} & \longrightarrow & S_i & \longrightarrow \cdots \longrightarrow & S_m \end{array}$$

It follows immediately from the definition of the grammar $G_M$ rules that $f(s_{i-1}, a_i) = s_i$ for all $i \in \{1, \dots, m\}$, where $s_0$ is the state in the second component of the initial nonterminal $S_0$ and also the initial state of the automaton $M$, $s_i$ is the state in the second component of the nonterminal $S_i$.

Because the fourth components of the nonterminals $S_{i-1}$ and $S_i$ are the same for all $i \in \{1, \dots, m\}$ and there is ACC in the fourth component of the initial nonterminal $S_0$, the operation ACC is contained in all nonterminals of the tree $T_1$.

The automaton $M$ only accepts after moving over the right delimiter », so $a_m =$ ». We get $a_0 a_1 \dots a_{m-1} a_m = w_1$ and

$$\delta^*(s_0, a_1 \dots a_{m-1} a_m) = \text{ACC}.$$

*Induction step.* Suppose the theorem holds for $p = q \geq 1$. We show that it is also true for $p = q + 1$. Let $T$ be any derivation tree of the grammar $G_M$ with $q + 1$ maximal nonterminal paths, which yields the word $w$. Then, by 3.1, $T' = R(T)$ is a derivation tree of the grammar $G_M$ with $q$ maximal nonterminal paths, and if it gives the word $w'$, then $w \Rightarrow_M w'$. For the tree $T'$ the theorem by induction holds, so it also holds for the tree $T$. □

**Lemma 3.4.** *Let $p$ be any natural number and $w_1$, ..., $w_p$ any words from « $\Sigma^*$ ». If at the same time*

- $w_{i+1} \Rightarrow_M w_i$ *for all* $i \in \{1, \dots, p-1\}$,
- $w_1$ *is accepted by the M automaton in a single stage.*

*then for some derivation trees $T_1$, ..., $T_p$ of the grammar $G_M$*

- $T_i = R(T_{i+1})$ *for all* $i \in \{1, \dots, p-1\}$,
- $T_i$ *gives the word* $w_i$ *for all* $i \in \{1, \dots, p\}$.

*Proof.* We prove the theorem by induction on the natural number $p$.

$p = 1$. Suppose that the word $w$ is accepted by the automaton $M$ in one stage. So for some $s_0$, ..., $s_m$ ($s_0$ is the initial state of $M$, $s_m =$ ACC) and $a_0$, $a_1$, ..., $a_m$ ($a_0 =$ «, $a_m =$ »), and for each $i \in \{1, \dots, m\}$

$$\delta(s_{i-1}, a_i) = s_i.$$

It follows that

$$S_i = (a_i, s_i, \lambda, \text{ACC}, \lambda)$$

is the nonterminal of the grammar $G_M$, $S_0$ is the initial nonterminal of the grammar $G_M$ and

$$S_{i-1} \to a_{i-1}\, S_i \qquad\qquad S_m \to a_m$$

are the rules of the grammar $G_M$. So we can construct the following derivation tree of the grammar $G_M$, which gives the word $w =$ «$a_1 \dots a_{m-1}$»:

$$\begin{array}{ccccccccc} « & & a_1 & \dots & a_{i-1} & & a_i & \dots & » \\ \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\ S_0 & \longrightarrow & S_1 & \longrightarrow \cdots \longrightarrow & S_{i-1} & \longrightarrow & S_i & \longrightarrow \cdots \longrightarrow & S_m \end{array}$$

*Induction step.* Assume that the theorem holds for $p = q \geq 1$. We show that it is also true for $p = q + 1$. Let $w$ be an arbitrary word of the language « $L(M)$ » that the automaton $M$ accepts in exactly $q$ stages. Let further $w'$ be any word such that $w' \Rightarrow_M w$. Thus, by the induction assumption, some derivation tree $T$ of the grammar $G_M$ yields the word $w$. Then, according to 3.2, there is a derivation tree $T'$ which gives the word $w'$ and $T = R(T')$. So the theorem holds for the word $w'$, which the automaton $M$ accepts in $q + 1$ stages. □

The following theorem is a direct consequence of lemmata 3.3 and 3.4.

**Theorem 3.1.** $L(G_M) =$ «$L(M)$».

The next theorem says that the grammar $G_M$ of the monotone reducing automaton $M$ is suitable for constructing a classical syntactic parser for the language recognized by this automaton.

**Theorem 3.2.** *$G_M$ is an* LR(0) *grammar.*

*Proof.* First, let us recall some classical notions introduced in the context of LR(0) grammars, as given, for example, in [4].

We call any expression of type

$$X \to \alpha . \beta,$$

an *item* of the context-free grammar $G$, where $X \to \alpha\beta$ is a rule of the grammar $G$. In particular, $X \to .$ is an item if $X \to \lambda$ is a rule of grammar $G$. For each rule $X \to \gamma$, we call $X \to \gamma .$ a *complete item*. We say that $A \to \alpha . \beta$ is a *valid item for the string* $\omega$ if there exists a right sentential form $\xi Au$ ($u$ is a string of terminals) such that $\xi\alpha = \omega$. We use the notation $I(\omega)$ for the set of all valid items for the string $\omega$.

A context-free grammar $G = (V, N, S, P)$ is called an LR(0) *grammar* if it satisfies the following conditions:

1. The initial symbol $S$ does not appear on the right-hand side of any rule.
2. No *reduce*/*reduce* conflict. For any string $\gamma \in (V \cup N)^*$, there is at most one complete item in the set $I(\gamma)$.
3. No *shift*/*reduce* conflict. If a complete item occurs in $I(\gamma)$, then there is no item with a terminal to the right of the dot in $I(\gamma)$.

The first condition is satisfied for grammar $G_M$, because the initial nonterminal $S$ does not occur on the right-hand side of any rule of $G_M$.

We prove the remaining two conditions using a method taken from [4] which is based on the notion of *characteristic*. We define the characteristic sequentially, first for terminal and nonterminal symbols, then for strings of terminal and nonterminal symbols, then for item sets of the grammar, and finally for item sets obtained by constructing the states of the item automaton of the grammar $G_M$. We show that each item set that is a state of the item automaton has a *continuous characteristic*, which results in the absence of both types of conflicts.

For each terminal or nonterminal symbol $x$ of grammar $G_M$, we define the *characteristic* $[x]$ *of the symbol* $x$ as follows:

$$[x] = \begin{cases} \texttt{initial}, & \text{if } x = S, \\ \texttt{terminal}, & \text{if } x \in V \\ (a, s, u), & \text{if } x = (a, s, u, o, v) \in N \setminus \{S\} \\ & \text{for some } o \text{ and } v. \end{cases}$$

The *characteristic of the strings* $\alpha$ formed by terminals and nonterminals of the grammar $G_M$ is defined by the following rule:

$$[\alpha] = \begin{cases} \texttt{empty}, & \text{if } \alpha = \lambda, \\ [x], & \text{if } x \in V \cup N \text{ and } \alpha = x\alpha' \\ & \text{for some } \alpha'. \end{cases}$$

We define the characteristic of the item $p = X \to \alpha . \beta$ as follows:

$$[p] = ([X], \alpha, [\beta])$$

The *characteristic of the set of items* $I$ of the grammar $G_M$ is defined as the following set:

$$[I] = \{[p] \mid p \in I\}$$

Checking all types of rules of the grammar $G_M$, we get that the first two components of item's characteristic uniquely determine its third component. More precisely, if $(c_{1,1}, \alpha_1, c_{1,2})$ and $(c_{2,1}, \alpha_2, c_{2,2})$ are characteristics of two items of the grammar $G_M$, then

$$\big(c_{1,1} = c_{2,1} \quad \text{and} \quad \alpha_1 = \alpha_2\big) \quad \Longrightarrow \quad c_{1,2} = c_{2,2}$$

A set of items $I$ has a *continuous characteristic* if

$$[I] = \{(c_0, \alpha, c_1), (c_1, \lambda, c_2), \ldots, (c_{n-1}, \lambda, c_n)\}$$

for some $c_0$, $c_1$, $c_2$, ..., $c_{n-1}$, $c_n$ and $\alpha$.

By verifying all types of rules of the grammar $G_M$, we can prove the validity of the following statements, which say that construction of the states of the item automaton creates only sets of items with a continuous characteristic.

1. $I(\lambda)$ has a continuous characteristic.
2. If $I$ has a continuous characteristic, $p = A \to \alpha . B\beta \in I$ and $I_p = \{B \to . \gamma \mid B \to \gamma \in G_M\}$, then $I \cup I_p$ has a continuous characteristic.
3. If $I$ has a continuous characteristic, then for each symbol $x$ of the grammar $G_M$, the set $I_x = \{A \to \alpha x . \beta \mid A \to \alpha . x\beta \in I\}$ also has a continuous characteristic.

Thus, any state $I(\gamma)$ has a continuous characteristic. The definition of a set of items with continuous characteristic implies that such a set can contain at most one triple whose third component is `empty` or `terminal`. Each complete item $A \to \alpha .$ has characteristic $([A], \alpha, \texttt{empty})$, and each item $A \to \alpha . \beta$ containing a terminal to the right of the dot has characteristic $([A], \alpha, \texttt{terminal})$. Therefore, any set of items with a continuous characteristic can contain at most one complete item, and if it does, it no longer contains any item with a terminal to the right of the dot.

Thus, each set $I(\gamma)$ satisfies the second and third conditions of the definition of LR(0) grammar and $G_M$ is an LR(0) grammar. □

# 4. Normalizations

Normalized mon-red-automata form a subclass of reducing automata. A normalized mon-red-automaton is any mon-red-automaton that is both *prefix-correct* and *state minimal*. In this section, we show how to construct an equivalent normalized mon-red-automaton to any mon-red-automaton that accepts a non-empty language.

We use a grammar of mon-red-automaton constructed in the previous section to construct an equivalent prefix-correct mon-red-automaton. Then we show how to minimize the set of states of any red-automaton including the elimination of unreachable states.

## 4.1. Prefix-correctness

Suppose that $M = (\Sigma, \langle\!\langle, \rangle\!\rangle, S, s_0, F, f)$ is a monotone reducing automaton. We say that $M$ is *prefix-correct* if for any word $u \in \Sigma^*$ and reduction sequence $n$ the following implications hold:

$$\begin{aligned} \delta^*(s_0, u) \in S &\implies \exists v \in \Sigma^* : uv \in L(M) \\ \delta^*(s_0, u) = \mathrm{RED}(n) &\implies \exists v \in \Sigma^* : uv \in L(M) \\ \delta^*(s_0, u\rangle\!\rangle) = \mathrm{RED}(n) &\implies u \in L(M) \end{aligned}$$

The first two implications say that any word $u \in \Sigma^*$, over which the automaton $M$ moves its working head while switching its control unit from the initial state $s_0$ to any transition or reducing state, is a prefix of some word from the language $L(M)$. The third implication states that any word $u\rangle\!\rangle \in \Sigma^*\rangle\!\rangle$, over which the automaton $M$ moves its working head while switching its control unit from the initial state $s_0$ to any reducing state, is a word of the language $L(M)\rangle\!\rangle$. However, the automaton may need a few more stages to formally accept it.

**Theorem 4.1.** *For each mon-red-automaton, an equivalent prefix-correct mon-red-automata can be constructed.*

*Proof.* Suppose that $M = (\Sigma, \langle\!\langle, \rangle\!\rangle, S, s_0, F, f)$ is a mon-red-automaton. First, we construct an LR(0) grammar $G_M$. Next, we construct a reducing automaton $M' = (\Sigma, \langle\!\langle, \rangle\!\rangle, S', s_0', F', f')$ based on the grammar $G_M$. Finally, we show that $M'$ is monotone, prefix-correct, and equivalent to the original reducing automaton $M$.

*Construction of the reducing automaton $M'$* resembles a construction of an item automaton for the grammar $G_M$. Transition states of the automaton $M'$ are defined as sets of items of this grammar, in the same way as in case of an item automaton. However, we are interested in those grammar rules only that can be used in the first nonterminal path of $G_M$'s derivation tree. While the grammar $G_M$ contains rules of the following types

$$\begin{array}{lll} X \to aYZ & X \to aY & X \to a \\ X \to YZ & X \to Y & X \to \lambda \end{array}$$

the first nonterminal path of any of grammar's derivation tree uses rules of types listed in the first row only, i.e. the rules with the right hand side starting with the terminal. Thus, the last rule used in the first maximal nonterminal path of any derivation tree of grammar $G_M$ is of type $X \to a$, and the first complete item encountered by the item automaton is of type $X \to a.$. Therefore, we construct the states of the reducing automaton $M'$ only from the following types of items

$$\begin{array}{lll} X \to .aYZ & X \to .aY & X \to .a \\ X \to a.YZ & X \to a.Y & X \to a. \end{array}$$

and we consider the transitions between item sets over terminal symbols only.

Now let us describe a construction of the prefix-correct automaton using just mentioned principles. For any $u \in \Sigma^*$ and $a \in \Sigma \cup \{\langle\!\langle, \rangle\!\rangle\}$ we define the item sets using the following rules

$$\begin{aligned} I(\lambda) &= \{S \to .\langle\!\langle\gamma \mid S \to \langle\!\langle\gamma \in P\}, \\ I(ua) &= \{X \to a.\alpha \mid X \to .a\alpha \in I(u)\} \\ &\quad \cup \{Y \to .b\beta \mid X \to .aY\gamma \in I(u) \text{ a } Y \to b\beta \in P\}, \end{aligned}$$

where $S$ is the starting nonterminal of the grammar $G_M$. We obtain the above item sets utilizing slightly modified method already used in classical theory of parsing to construct an item automaton for a given LR(0) grammar. Here, we consider only items of the above types. Thus, our item sets are subsets of states of the classical item automaton, and hence contain neither *shift*/*reduce* nor *reduce*/*reduce* conflict. A set of all nonterminal states of the reducing automaton $M'$ consists of all constructed item sets except $I(\lambda)$ and sets containing only a complete item. Its initial state is the set $I(\langle\!\langle)$. The reducing states are just all $\mathrm{RED}(n)$ for which there exists a complete item $X \to a.$ of the grammar $G(M')$ and $X = (a, \mathrm{RED}(n), \lambda, \mathrm{RED}(n), \lambda)$. We define the transition function as follows:

$$f'\big(I(u), a\big) = \begin{cases} \mathrm{RED}(n), & \text{if } I(ua) = \{X \to a.\} \text{ and} \\ & X = (a, \mathrm{RED}(n), \lambda, \mathrm{RED}(n), \lambda) \\ & \text{for some } n, \\ \mathrm{ACC}, & \text{if } a = \rangle\!\rangle \text{ and} \\ & I(u\rangle\!\rangle) = \{X \to \rangle\!\rangle.\} \text{ and} \\ & X = (\rangle\!\rangle, \mathrm{ACC}, \lambda, \mathrm{ACC}, \lambda), \\ \mathrm{ERR}, & \text{if } I(ua) = \emptyset, \\ I(ua), & \text{otherwise.} \end{cases}$$

*Equivalence of automata $M'$ and $M$.* Apparently, $\langle\!\langle L(M)\rangle\!\rangle = L(G_M)$. We show that

$$L(G_M) = \langle\!\langle L(M')\rangle\!\rangle.$$

Suppose that $\langle\!\langle w\rangle\!\rangle \in L(G_M)$ and $T$ is the derivation tree of the grammar $G_M$, which gives $\langle\!\langle w\rangle\!\rangle$. By induction on

the number $p$ of maximal nonterminal paths in the tree $T$, we prove that $w \in L(M')$.

If the tree $T$ contains only one maximal nonterminal path, then $I(\text{«}w\text{»}) = \{X \to \text{»}\,.\,\}$ and $X = (\text{»}, \text{ACC}, \lambda, \text{ACC}, \lambda)$, so $\delta'^*(I(\text{«}), w\text{»}) = \text{ACC}$ and thus $w \in L(M')$.

Assume that the statement holds for all derivation trees of the grammar $G_M$ with at most $p$ maximal nonterminal paths. We then prove that it also holds for derivation trees with $p+1$ maximal nonterminal paths. If $T$ is a derivation tree with $p+1$ maximal nonterminal paths, which gives the word «$w$», then by Lemma 3.3, $R(T)$ is a tree with $p$ maximal nonterminal paths that yields the word «$w'$» such that «$w$» $\Rightarrow_M$ «$w'$». By the induction assumption $w' \in L(M')$. Thus, $\delta^*(s_0, ua) = \text{RED}(n)$ for some prefix «$ua$ of «$w$», where $I(\text{«}ua) = \{X \to a\,.\,\}$ and $X = (a, \text{RED}(n), \lambda, \text{RED}(n), \lambda)$. So $\delta'^*(I(\text{«}u), a) = \text{RED}(n)$ and «$w$» $\Rightarrow_{M'}$ «$w'$». Therefore $w \in L(M')$.

We can prove the reverse inclusion ($\text{«}L(M')\text{»} \subseteq L(G_M)$) in a similar way by induction on the number of stages of the reduction analysis of «$w$» by the automaton $M'$ using Lemma 3.2.

*Prefix correctness.* Suppose first that the automaton $M'$ moves from the initial state $I(\text{«})$ over the prefix $u \in \Sigma^*$ of the word $w$» to the state $I(\text{«}u)$, formally

$$\delta'^*(I(\text{«}), u) = I(\text{«}u).$$

This means that the items listed below in the left column can be selected from the sets $I(\text{«})$, ..., $I(\text{«}u)$ and used in the derivation by $G_M$ listed in the right column

$$\begin{array}{ll} S \Rightarrow \text{«}\,.\,S_1\,\gamma_1 & S \Rightarrow \text{«}\,S_1\,\gamma_1 \\ S_1 \to a_1\,.\,S_2\,\gamma_2 & \Rightarrow \text{«}\,a_1\,S_2\,\gamma_2\gamma_1 \\ \quad\vdots & \quad\vdots \\ S_k \to a_k\,.\,S_{k+1}\,\gamma_{k+1} & \Rightarrow \text{«}\,a_1 \ldots a_k\,S_{k+1}\gamma_{k+1}\,\gamma_k \cdots \gamma_2\,\gamma_1 \end{array}$$

where $a_1 \ldots a_k = u$. Since the grammar $G_M$ is reduced, all nonterminals in $S_{k+1}\,\gamma_{k+1}\,\gamma$ can be rewritten into some terminal strings, so that $u$ is a prefix of some word from $L(M)$.

Another possibility is that $\delta'^*(I(\text{«}), u) = \text{RED}(n)$ for some reduction sequence $n$. Then $I(\text{«}u) = \{X \to a_k\,.\,\}$, where $X = (a_k, \text{RED}(n), \lambda, \text{RED}(n), \lambda)$. Just replace the last item $S_k \to a_k\,.\,S_{k+1}\,\gamma_{k+1}$ in the list above with the complete item $S_k \to a_k\,.$ to get the derivation $S_k \Rightarrow^*_{G_M} \text{«}\,u\,\gamma$. If $a_k \neq \text{»}$, then $u$ is a prefix of some word of the language $L(M)$. If $a_k = \text{»}$, then $\gamma \Rightarrow^*_{G_M} \lambda$ (otherwise the grammar $G_M$ would generate a word outside «$\Sigma^*$») and $u \in L(M)$».

*Monotony.* Suppose the automaton $M'$ moves from the state $I(\text{«})$ via the prefix $uv$ of the word $w$» to the state $\text{RED}(n) \in F'$, $|v| = |n|$, and that the word $u$ consists of the symbols $a_1$, ..., $a_k \in \Sigma$ and the word $v$ consists of the symbols $a_{k+1}$, ..., $a_{l-1} \in \Sigma$, $a_l \in \Sigma \cup \{\text{»}\}$. As in the proof of prefix correctness, we can now select the items listed in the left column from the sets $I(\text{«})$, ..., $I(\text{«}uv)$ and then use their rules in the derivation listed in the right column:

$$\begin{array}{ll} S \Rightarrow \text{«}\,.\,S_1\,\gamma_1 & S \Rightarrow \text{«}\,S_1\,\gamma_1 \\ S_1 \to a_1\,.\,S_2\,\gamma_2 & \Rightarrow \text{«}\,a_1\,S_2\,\gamma_2\,\gamma_1 \\ \quad\vdots & \quad\vdots \\ S_{k-1} \to a_{k-1}\,.\,S_k\,\gamma_k & \Rightarrow \text{«}\,a_1 \ldots a_{k-1}\,S_k\,\gamma_k \cdots \gamma_2\,\gamma_1 \\ S_k \to a_k\,.\;S_{k+1}\,X_0 & \Rightarrow \text{«}\,a_1 \ldots a_{k-1}\,a_k\,S_{k+1}\,X_0\,\gamma \\ S_{k+1} \to a_{k+1}\,.\,S_{k+2} & \Rightarrow \text{«}\,u\,a_{k+1}\,S_{k+2}\,X_0\,\gamma \\ \quad\vdots & \quad\vdots \\ S_l \to a_l\,. & \Rightarrow \text{«}\,u\,a_{k+1} \ldots a_{l-1}\,a_l\,X_0\,\gamma, \end{array}$$

where $\gamma = \gamma_k \ldots \gamma_1$, $u = a_1 \ldots a_k$, and $v = a_{k+1} \ldots a_l$. Since $G_M$ is reduced, some terminal string can be derived from $X_0$. Suppose that $v/n = b_1 \ldots b_m$. From the definition of the grammar $G_M$, it follows that some of its nonterminals $X_1$, $X_2$, ..., $X_m$ contain the symbols $b_1$, $b_2$, ..., $b_m$ in their first component, and the grammar contains the following rules

$$\begin{array}{ll} S_k \to a_k\,S_{k+1}\,X_0 & S_k = X_0^\lambda \\ X_0 \to X_1\,\gamma_1' & X_0^\lambda \to a_k\,X_1^\lambda\,\gamma_1' \\ X_1 \to X_2\,\gamma_2' & X_1^\lambda \to b_1\,X_2^\lambda\,\gamma_2' \\ X_2 \to X_3\,\gamma_3' & X_2^\lambda \to b_2\,X_3^\lambda\,\gamma_3' \\ \quad\vdots & \quad\vdots \\ X_m \to \gamma' & X_m^\lambda \to b_m\,\gamma' \end{array}$$

The equality of the nonterminals $S_k$ and $X_0^\lambda$ together with the existence of the rules in the second column follows from the application of implications (3), (4), and (5). These rules allow the automaton $M'$ to move from the state $I(\text{«}u)$ over the word $v/n = b_1 b_2 \ldots b_m$ to the transition state $I(\text{«}uv/n)$ or to ACC or to some reducing state. Thus the reducing automaton $M'$ is monotone. □

## 4.2. State minimality

We show how to minimize the set of transition, final, and reducing states of a reducing automaton while preserving reduction analysis and, moreover, visited working list positions in each stage.

Suppose $M = (\Sigma, \text{«}, \text{»}, S, s_0, F, f)$ is any reducing automaton and $s \in S \cup F$. We call the state $s$ *reachable* if $\delta^*(s_0, w) = s$ for some $w \in \Sigma^* \cdot \{\lambda, \text{»}\}$. A state that is not reachable is called *unreachable*.

**Theorem 4.2.** *An equvalent reducing automaton with only reachable states can be constructed to any reducing automaton.*

*Proof.* A reducing automaton can be viewed as a finite-state machine with an alphabet $\Sigma \cup \{\text{»}\}$, a set of states

$S \cup F \cup \{\text{RED}\}$, an initial state $s_0$, a set of final states $F$, and a transition function $\delta$. Thus, we can use the construction of a finite-state machine containing only reachable states from the theory of finite-state machines. □

Let $M_1 = (\Sigma, «, », S_1, s_1, F_1, f_1)$ and $M_2 = (\Sigma, «, », S_2, s_2, F_2, f_2)$ be reducing automata with the same input alphabet $\Sigma$. We define the relation $\sim \subseteq S_1 \times S_2$ in the following way: $s \sim s'$, iff for each word $w \in \Sigma^* \cdot \{\lambda, »\}$ from $\delta_1^*(s, w) \in F_1$ or $\delta_2^*(s', w) \in F_2$ it follows that $\delta_1^*(s, w) = \delta_2^*(s', w)$. If $M_1 = M_2$, then the relation $\sim$ is the equivalence on the set of transition states $S_1$ of the automaton $M_1$. We call the transition states, that are in the relation $\sim$, *stage-equivalent*. We call reducing automata $M_1$ and $M_2$ stage-equivalent, if their initial states are stage-equivalent ($s_1 \sim s_2$). Two stage-equivalent reducing automata either both accept, both reject, or both reduce any word, in all three cases at the same place in the working list, and in the case of reduction also according to the same reduction sequence. Thus, any two stage-equivalent reducing automata are obviously strongly equivalent and vice versa. Since the stage equivalence and strong equivalence name the same phenomenon, we henceforth use only the term strong equivalence.

Let $M$ be any reducing automaton. An automaton $M$ is *state minimal* if (i) all its states are reachable, and (ii) no its different transition states are equivalent. A reducing automaton $M'$ is called a *reduct* of a reducing automaton $M$ if (i) $M'$ is strongly equivalent to $M$, and (ii) $M'$ is state minimal.

**Theorem 4.3.** *A reduct can be constructed to any reducing automaton.*

*Proof.* Let $M = (\Sigma, «, », S, s_0, F, f)$ be a reducing automaton. We show how to construct its reduct $M'$.

We convert the problem of constructing a reduct of a reducing automaton to the problem of constructing a reduct of a Moore machine, which is already solved in the theory of finite-state machines and finite-state transducers.

We can look at a reducing automaton as a finite-state machine extended with the ability to reduce a working list. The reducing automaton $M$ computes a transition function over a word contained in a working list to determine whether and where to accept, reject, or reduce a word, and in the case of reduction, how to reduce it. Moore machine is also, in principle, a finite-state machine augmented by marking the input word. It also computes a transition function over an input word and continuously marks the input word based on its value. The reduct $A'$ of Moore machine $A$ is a *state-minimal* machine that marks any word in the same way as the original machine $A$. So the idea is to design a Moore machine $A$ for a reducing automaton $M$ which, by its output function, marks in the input word whether and where to accept, reject or reduce it (and how to reduce it) in the same way as the reducing automaton $M$. We then construct its reduct $A'$ (with the same behaviour as $A$) and finally convert it back to the reducing automaton $M$.

The Moore machine $A = (S_A, s_A, \Sigma_A, \Gamma_A, \delta_A, \mu_A)$, where $S_A$ is a set of states, $s_A$ is an initial state, $\Sigma_A$ is an input alphabet, $\Gamma_A$ is an output alphabet, $\delta_A$ is a transition function, and $\mu_A$ is an output function, is defined in the following way:

$$
\begin{aligned}
S_A &= S \cup F \cup \{\text{RED}\} \\
s_A &= s_0 \\
\Sigma_A &= \Sigma \cup \{»\} \\
\Gamma_A &= F \cup \{\text{RED}\} \cup \{s_0\} \\
\delta_A &= \delta \\
\mu_A(s) &= \begin{cases} s_0, & \text{if } s \in S \\ s, & \text{if } s \in F \cup \{\text{RED}\} \end{cases}
\end{aligned}
$$

For the Moore machine $A$ defined in this way, we construct its reduct $A' = (S_{A'}, s_{A'}, \Sigma_{A'}, \Gamma_{A'}, \delta_{A'}, \mu_{A'})$ by the construction given, for example, in [5].

Finally, we move from the reduct $A'$ back to the reducing automaton $M' = (\Sigma', «, », S', s_0', F', f')$ defined as follows:

$$
\begin{aligned}
\Sigma' &= \Sigma_{A'} \setminus \{»\} = \Sigma_A \setminus \{»\} = \Sigma \\
S' &= \{s \mid \mu_{A'}(s) = s_0\} \\
s_0' &= s_{A'} = s_A = s_0 \\
F' &= \{s' \in F \mid \exists s \in S_{A'} : s' = \mu_{A'}(s)\} \\
f'(s, a) &= \begin{cases} s', & \text{if } s' = \delta_{A'}(s, a) \in S' \\ \mu_{A'}(s'), & \text{if } s' = \delta_{A'}(s, a) \notin S' \end{cases}
\end{aligned}
$$

Now it is not hard to see that the constructed reducing automaton $M'$ is a reduct of the automaton $M$.

The strong equivalence of the reducing automata $M$ and $M'$ follows from the following facts: 1. The set $F'$ contains exactly all reachable final or reducing states of the reducing automaton $M$. 2. The initial state $s_{A'}$ of the Moore machine $A'$ is behaviorally equivalent to the initial state $s_A$ of the Moore machine $A$. Thus, for any word $w \in \Sigma^* \cdot \{\lambda, »\}$ and the reachable reducing state $\text{RED}(n)$ of the reducing automaton $M$, the following equivalence holds:

$$
\begin{aligned}
\delta'^*(s_0', w) = \text{RED}(n) &\iff \\
\mu_{A'}\big(\delta_{A'}^*(s_{A'}, w)\big) = \text{RED}(n) &\iff \\
\mu_A\big(\delta_A^*(s_A, w)\big) = \text{RED}(n) &\iff \\
\delta^*(s_0, w) = \text{RED}(n)
\end{aligned}
$$

The same equivalences holds if we replace the reducing state $\text{RED}(n)$ with the accepting or rejecting states ACC or ERR.

The state-minimality of the automaton $M'$ follows directly from the state-minimality of Moore machine $A'$ and from the way we obtained the automaton $M'$ from the machine $A'$. The reducing automaton $M'$ is therefore a reduct of the reducing automaton $M$. □

We say that reducing automata $M$ and $M'$ with the same alphabet $\Sigma$ are *isomorphic* if there is a one to one mapping $h$,

$$h : S \cup F \longrightarrow S' \cup F',$$

such that

(i) $h(s) = s$ for any $s \in F$;

(ii) $\delta(s, a) = s' \iff \delta'\big(h(s), a\big) = h(s')$ for any $s \in S$, $a \in \Sigma \cup \{»\}$, and $s' \in S \cup F$.

**Theorem 4.4.** *Any two reducts of the same reducing automaton are isomorphic.*

This theorem can obviously be proved by modifying the proof of the same theorem for Moore machines.

**Corollary 4.1.** *No reducing automaton has fewer states than its reduct.*

As the reduct is strongly equivalent to the reducing automaton for which it is constructed, the construction retains both monotony and prefix-correctness. So, for any monotone reducing automaton we can construct an equivalent monotone reducing automaton, which is both prefix-correct and state-minimal.

## 5. Conclusion

We introduced the reducing automaton and described two of its normalizations. Further, we plan to propose a construction of an equivalent mon-red-automaton with only *repeatable reductions*. A reduction is repeatable if «$xuyvz$» $\Rightarrow_M$ $xyz$ implies «$xu^{n+1}yv^{n+1}z$» $\Rightarrow_M$ «$xu^nyv^nz$» for each $n \in \mathbb{N}_0$. Further, we would like to distinguish repeatable reductions reducing *regular contexts* (either $u$ or $v$ is empty) and *linear contexts* (neither $u$ nor $v$ is empty) and normalize a monotone reducing automaton so that it reduces regular contexts from the right.

## Acknowledgments

This paper was created as part of the *Parsing and Syntactic Analysis seminar* led by *František Mráz* and *Martin Plátek* at the *Faculty of Mathematics and Physics of Charles University in Prague.*

## References

[1] M. Procházka, Redukční automaty a syntaktické chyby, Ph.D. thesis, Univerzita Karlova v Praze, fakulta Matematicko-fyzikální, Praha, 2012.

[2] M. Procházka, M. Plátek, Redukční automaty, monotonie a redukovanost, ITAT, 2002.

[3] P. Jančar, F. Mráz, M. Plátek, J. Vogel, Restarting automata, Lecture Notes in Computer Science 965 (1995) 283–292.

[4] M. Chytil, Automaty a gramatiky, SNTL Praha, Praha, 1984.

[5] E. F. Moore, Gedanken-experiments on sequential machines, Annals of Mathematics studies 34 (1956) 129–153.