ON THE COMPLEXITY OF A PROBLEM ON MONADIC STRING REWRITING SYSTEMS

F.L. ŢIPLEA AND ERKKI MÄKINEN

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

UNIVERSITY OF TAMPERE

REPORT A-2001-4

UNIVERSITY OF TAMPERE
DEPARTMENT OF COMPUTER AND
INFORMATION SCIENCES
SERIES OF PUBLICATIONS A
A-2001-4, JUNE 2001

ON THE COMPLEXITY OF A PROBLEM ON MONADIC STRING REWRITING SYSTEMS

F.L. ŢIPLEA AND ERKKI MÄKINEN

University of Tampere
Department of Computer and Information Sciences
P.O.Box 607
FIN-33014 University of Tampere, Finland

ISBN 951-44-5135-X
ISSN 1457-2060

ON THE COMPLEXITY OF A PROBLEM ON MONADIC STRING REWRITING SYSTEMS[1]

Ferucio Laurenţiu Ţiplea[2]
Faculty of Computer Science, "Al.I.Cuza" University of Iaşi
Iaşi, Romania
e-mail: `fltiplea@mail.dntis.ro`

and

Erkki Mäkinen[3]
Department of Computer and Information Sciences, Tampere University
P.O. Box 607, Tampere, FIN-33014, Finland
e-mail: `em@cs.uta.fi`

ABSTRACT

Computing the set of descendants of a regular language L with respect to a monadic string rewriting system has proved to be very useful in developing decision algorithms for various problems on finitely-presented monoids and context-free grammars. Recently, Esparza et al. [6] proved $\mathcal{O}(ps^3)$ time and $\mathcal{O}(ps^2)$ space bounds for this problem, where p is the number of rules in the monadic string rewriting system and s is the number of states in the automaton accepting L.

Using *synchronized extension systems* [11, 9, 10] we provide a new insight to the problem and allow an $\mathcal{O}(pr)$ time and space solution, where p is as above and r is the number of the rules in the grammar generating L.

Keywords: formal languages, decision problems, monadic string rewriting systems, synchronized extension systems, computational complexity

1. Introduction and Preliminaries

In [3] (see also [2]), Book and Otto show, among many other results, that if the rewriting rules of a monadic string rewriting system T are applied to the strings of a regular set L, the set $\Delta_T^*(L)$ so obtained (the set of descendants of L with respect to T) is also regular. This result can be used to develop a methodology for designing decision algorithms for various problems on finitely-presented monoids and context-free grammars. The complexity of the decision algorithms obtained in this way depends highly, among others factors, on the complexity of the transformation of a finite automaton accepting a language L into an automaton accepting the set of descendants of L. The original transformation algorithm by Book and Otto was later improved by Bouajjani et al. [4] and Esparza et al. [5, 6]. Esparza et al. [6] proved $\mathcal{O}(ps^3)$ time and $\mathcal{O}(ps^2)$ space bounds where p is the number of rules in the monadic string rewriting system and s is the number of states in the automaton accepting L.

[1]To be presented at the Third Workshop on Descriptional Complexity of Automata, Grammars and Related Structures, July 20 - 22, 2001, Vienna, Austria.

[2]While visiting Carnegie Mellon University (Pittsburgh, Pennsylvania) by a Fulbright research grant.

[3]Work supported by the Academy of Finland (Project 35025).

Using *synchronized extension systems* [11, 9, 10], a new powerful and elegant rewriting formalism, we provide a new insight to the problem and allow an $\mathcal{O}(pr)$ time and space solution, where p is as above and r is the number of the rules in the grammar generating L. Notice that synchronized extension systems deal the problems in different level of granularity than finite automata used by the authors citied above.

For a self-contained and elegant treatment of this problem, we will survey to some extent the Book and Otto's methodology, as well as some algorithms for computing the set of descendants.

2. An Approach for Designing Decision Algorithms

Recall first some concepts from [3] related to string rewriting systems. A *string rewriting system over an alphabet* Σ (shortly, an *STS over* Σ) is an non-empty subset $T \subseteq \Sigma^* \times \Sigma^*$. Each element $(\alpha, \beta) \in T$ is called a *(rewrite) rule*; they are usually denoted by $\alpha \to \beta$. The *rewriting* (or *step derivation*) *relation induced by* T is the binary relation $\Rightarrow_T$ on Σ^* given by

$$u \Rightarrow_T v \quad \text{iff} \quad u = u_1\alpha u_2 \ \wedge \ v = u_1\beta u_2 \ \wedge \ \alpha \to \beta \in T,$$

for all $u, v \in \Sigma^*$. The reflexive and transitive closure of $\Rightarrow_T$, denoted by $\stackrel{*}{\Rightarrow}_T$, is called the *derivation relation induced by* T.

A *(non-empty) derivation of* u *into* v by T is a sequence of step derivations

$$u = u_0'\alpha_1 u_0'' \Rightarrow_T u_0'\beta_1 u_0'' = u_1 = u_1'\alpha_2 u_1'' \Rightarrow_T \cdots \Rightarrow_T u_{n-1}'\beta_n u_{n-1}'' = u_n = v,$$

where $n \geq 1$ and $r_i : \alpha_i \to \beta_i \in T$ for all $1 \leq i \leq n$. Sometimes we will write $u \xRightarrow{r_1 \cdots r_n}_T v$ to denote the fact that u is rewritten into v by the rules $r_1, \ldots, r_n$ used in this order (this notation is ambiguous because it does not take into consideration the places where the rules are applied. However, we will use it in conjunction with a derivation explicitly given as above in order to simplify the notation and, therefore, the ambiguities will be avoided.)

For a language L over an alphabet Σ and an STS T over Σ, we denote by $\Delta_T^*(L)$ the language

$$\Delta_T^*(L) = \{v \in \Sigma^* | \exists u \in L : \ u \stackrel{*}{\Rightarrow}_T v\}.$$

A *monadic STS* (over an alphabet Σ) is an STS having the property $|\alpha| > |\beta|$ and $|\beta| \leq 1$, for each rule $\alpha \to \beta$.

In [3] (p. 91), the following important result has been proved.

Theorem 1 *For any non-deterministic finite automaton A and monadic STS T one can construct in polynomial time a non-deterministic finite automaton B such that $L(B) = \Delta_T^*(L(A))$.*

Theorem 1 has many important consequences concerning the decidability of various problems on rewriting systems. The main strategy in using this theorem is to transform the decision problem we want to study into an equivalent one on regular languages. As an example, consider the "extended word problem" for a confluent monadic STS (for definition, see [3], p. 11) T given by:

Instance: Two regular sets R_1 and R_2 specified by nondeterministic finite-state automata.
Question: Do there exist $x \in R_1$ and $y \in R_2$ such that $x \stackrel{*}{\leftrightarrow}_T y$?

One can easily see that

$$(\exists x \in R_1)(\exists y \in R_2)(x \stackrel{*}{\leftrightarrow}_T y) \ \text{ iff } \ \Delta_T^*(R_1) \cap \Delta_T^*(R_2) \neq \emptyset.$$

Since $\Delta_T^*(R_1)$ and $\Delta_T^*(R_2)$ are regular, and the question

$$\Delta_T^*(R_1) \cap \Delta_T^*(R_2) \stackrel{?}{=} \emptyset$$

is decidable, the extended word problem for T is decidable.

The decision problems studied so far by means of Theorem 1 can be grouped into two classes:

1. Decision problems on monoids presented by finite confluent monadic STS's. This class includes problems like the extended word problem, the power problem, the left/right divisibility problem, the submonoid problem, the independent set problem, the subgroup problem, the left/right/two-sided ideal problem, Green's relations decision problem etc.
2. Decision problems on context-free languages. This class includes problems like the emptiness problem, the finiteness problem, the membership problem, the useless variable problem, and the nullable variable problem.

The first class of decision problems has been studied to a great extent by Book [1]. Some limitations of the Book method (called the *method of linear sentences* – see [3], p. 104) are studied in [8]. For a uniform treatment the reader is referred to [3]. Book and Otto also pointed out (see the end of Chapter 6 in [3]) the applicability of their method to certain problems on context-free grammars, but they did not go into details. These were set up in [4, 5, 6].

The complexity of a decision algorithm based on the method of linear sentences depends on (a) the complexity of designing a non-deterministic finite automaton B as in Theorem 1 and (b) the complexity of the resulting decision problem on regular languages.

Book and Otto have discussed the complexity of the decision problem they studied only from a qualitative point of view (polynomial time/space). In [6], a quantitative point of view is considered, in order to compare the efficiency of the new decision algorithms with respect to well-known algorithms (for example, CYK or Earley's algorithms). The discussion focuses on (a) above. We will discuss three methods for (a), one being better than the others.

3. Computing the Set of Descendants

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and T be a monadic STS on Σ. The algorithm proposed by Book and Otto for computing a finite automaton recognizing the set $\Delta_T^*(L(A))$ changes the transition relation δ by adding certain transitions and by turning certain non-final states into final ones.

Algorithm 1

Input: A and T as above;
Output: $A' = (Q, \Sigma, \Delta', q_0, F)$ accepting $\Delta_T^*(L(A))$;
begin
  $\delta' := \delta$;
  repeat
    for $q, q' \in Q$ and $\alpha \to \beta \in T$ **do**
      if $q' \in \bar{\delta}'(q, \alpha)$ **then** $\delta' := \delta' \cup \{(a, \beta, q')\}$
  until δ' does not change any more
end

It is not difficult to prove the correctness of this algorithm (see for details [5]). The running time of this algorithm can be estimated as follows. Let p be the number of rules in T, l be the maximal length of the left hand side of a rule, and let s be the number of states in

A. The repeat-until loop in Algorithm 1 is performed at most $\mathcal{O}(p \cdot s^2)$ times because the transitions added to δ' are labelled by the right hand sides of rules in T. The for-loop is performed $\Theta(p \cdot s^2)$ times, and checking whether $q' \in \bar{\delta}'(q, \alpha)$ holds can be done by simulating A on input α, which requires time $\mathcal{O}(l \cdot s^2)$. Adding an element to the relation δ' requires a constant time. Therefore, the running time of Algorithm 1 is $\mathcal{O}(l \cdot p^2 \cdot s^6)$.

In [5], Esparza et al. proposed a new algorithm, whose running time is $\mathcal{O}(p \cdot s^4)$. Later in [4], the bounds were improved to $\mathcal{O}(p \cdot s^3)$ time and space. The space complexity was the subject of a new improvement to $\mathcal{O}(p \cdot s^2)$ in [6]. All these versions work on context-free grammars which are special cases of monadic STS's. Moreover, the final version obtained in [6] requires the Chomsky normal form extended with unit productions and λ-productions. This is not a real restriction because such a normal form can be obtained in linear time and with a linear growth in the size of the productions. For unambiguous grammars, the time complexity is $\mathcal{O}(p \cdot s^2)$, as proved in [6].

Excepting the (conceptually) very simple method by Book and Otto, the above solutions require complicated computations. The solution we propose in the next section is fairly trivial and superior from the complexity point of view. It works directly with the original monadic STS. For convenience, the regular language is consider to be specified by a regular grammar.

4. The New Solution

In this section we first recall the basic definitions related to synchronized extension systems. This recently introduced rewriting formalism is then used in creating a new efficient solution for the problem discussed in the previous sections.

4.1. Synchronized Extension Systems

Synchronized extension systems (SE-systems, for short) are introduced in [11] as 4-tuples $G = (V, L_1, L_2, S)$, where V is an alphabet and L_1, L_2, and S are languages over V. L_1 is called the *initial language*, L_2 the *extending language*, and S the *synchronization set* of G. For an SE-system G, define the binary relations $\Rightarrow_{G,r}$ and $\Rightarrow_{G,r^-}$ over V^* as follows (the leftmost versions $\Rightarrow_{G,l}$ and $\Rightarrow_{G,l^-}$ can be defined analogously):

- $u \Rightarrow_{G,r} v$ iff $(\exists w \in L_2)(\exists s \in S)(\exists x, y \in V^*)(u = xs \ \wedge \ w = sy \ \wedge \ v = xsy)$;
- $u \Rightarrow_{G,r^-} v$ iff $(\exists w \in L_2)(\exists s \in S)(\exists x, y \in V^*)(u = xs \ \wedge \ w = sy \ \wedge \ v = xy)$.

In an SE-system $G = (V, L_1, L_2, S)$, the words in S act as synchronization words. They can be kept or neglected in the final result, and r, r^-, l, and l^- are called (basic) *modes of synchronizations.* In what follows, we restrict ourselves to the mode r^-.

We say that an SE-system $G = (V, L_1, L_2, S)$ is *of type* (p_1, p_2, p_3) if the L_1, L_2, and S are languages having the properties p_1, p_2, and p_3, respectively. We use the abbreviations f and reg for the properties of finiteness and regularity, respectively.

A derivation $u \stackrel{*}{\Rightarrow}_{r^-} v$ is called an *r^--derivation.* The *language of type* r^- generated by an *SE-system* $G = (V, L_1, L_2, S)$ is defined as

$$L^{r^-}(G) = \{v \in V^* | \exists u \in L_1 : \ u \stackrel{*}{\Rightarrow}_{G,r^-} v\}.$$

The following result is essential for this paper.

Theorem 2 ([11]) *For any SE-system G of type (reg, reg, f), the language $L^{r^-}(G)$ is regular.*

4.2. Left-to-Right Derivations in STS's

In this section we give technical results concerning the form of derivations in finite STS's. The concept of a "derivation from u on x within the decomposition $u = u_1xu_2$" is aimed to capture the idea that the subwords u_1 and u_2 are not used (neither partially nor totally) in a derivation. Alternatively, one can say that the derivation $u = u_1xu_2 \stackrel{*}{\Rightarrow}_T u_1yu_2$ from u on x is obtained from the (normal) derivation $x \stackrel{*}{\Rightarrow}_T y$ by catenating to each step the word u_1 to the left and the word u_2 to the right.

Definition 1 *Let T be a finite STS over an alphabet Σ, $u \in \Sigma^+$ and x a subword of u. A* derivation from u on x within a decomposition $u = u_1xu_2$ *is defined inductively as follows:*

- *if $u = u_1xu_2 = u_1x'\alpha x''u_2$ and $\alpha \to \beta \in T$, then*

$$u = u_1xu_2 = u_1x'\alpha x''u_2 \Rightarrow_T u_1x'\beta x''u_2$$

is a derivation on x;

- *if $u = u_1xu_2 \stackrel{*}{\Rightarrow}_T u_1yu_2$ is a derivation on x and*

$$u_1yu_2 = u_1y'\alpha y''u_2 \Rightarrow_T u_1y'\beta y''u_2$$

is a derivation on y (within the decomposition u_1yu_2), then

$$u = u_1xu_2 \stackrel{*}{\Rightarrow}_T u_1yu_2 = u_1y'\alpha y''u_2 \Rightarrow_T u_1y'\beta y''u_2$$

is a derivation on x.

Now we are ready to define left-to-right derivations of a finite STS.

Definition 2 *Let T be a finite STS over an alphabet Σ, $u \in \Sigma^+$, and let $\mathcal{D}$ the derivation*

$$\mathcal{D}: \quad u \xRightarrow{r_1 \cdots r_{i-1}} u_1\alpha_i u_2 \stackrel{r_i}{\Rightarrow} u_1\beta_i u_2 \xRightarrow{r_{i+1} \cdots r_n} v.$$

Let $i < j \leq n$.

(1) The step j of $\mathcal{D}$ is said to be to the left of the step i *if there is a decomposition $u_1 = u_1'u_1''$ of u_1 such that the derivation*

$$u_1\beta_i u_2 = u_1'u_1''\beta_i u_2 \xRightarrow{r_{i+1} \cdots r_{j-1}} xy$$

is on u_1' or $u_1''\beta_i u_2$ and the step j (of $\mathcal{D}$) is on x.

(2) The step j of $\mathcal{D}$ is said to be to the right of the step i *if there is a decomposition $u_2 = u_2'u_2''$ of u_2 such that the derivation*

$$u_1\beta_i u_2 = u_1\beta_i u_2'u_2'' \xRightarrow{r_{i+1} \cdots r_{j-1}} xy$$

is on $u_1\beta_i u_2'$ or u_2'' and the step j (of $\mathcal{D}$) is on y.

(3) The step j of the derivation $\mathcal{D}$ is said to be dependent on the step i *if it is neither to the left nor to the right of the step i.*

(4) The derivation $\mathcal{D}$ is called a left-to-right (right-to-left) derivation of u into v *if for every i, $1 \leq i < n$, the step $i+1$ is not to the left (right) of the step i.*

The following lemma states that it is sufficient to consider left-to-right derivations in finite STS's (see also [7] for a classical treatment of a similar topic).

Lemma 3 *Let T be a finite STS over an alphabet Σ. Then, for every derivation $\mathcal{D}$ of a word u into a word v one can effectively construct a left-to-right derivation $\mathcal{D}'$ of u into v. Moreover, the derivation $\mathcal{D}'$ can be obtained by changing only the order of steps in the original derivation $\mathcal{D}$.*

Proof. Let $\mathcal{D}$ be a derivation of u into v,

$$\mathcal{D}: \quad u \stackrel{s}{\Rightarrow}_T v,$$

where $s = r_1 \dots r_n \in T^+$.

Define inductively a sequence $s' = r_{i_1} \cdots r_{i_n}$, where $i_1, \dots, i_n \in \{1, \dots, n\}$ are pairwise distinct, as follows:

1. Initially, set $s' := r_1$;
2. Assume that s' is the sequence obtained by rearranging the subsequence $r_1 \cdots r_k$ of s, where $k < n$;
3. Consider the rule r_{k+1} and the following possible cases:
   (a) r_{k+1} is to the left of all rules in s'. Then, define $s' = r_{k+1}s'$;
   (b) r_{k+1} does not depend on any rule in s'. Then, find the biggest j such that r_{k+1} is to the right of $s'(j)$ and insert r_{k+1} immediately after $s'(j)$ (that is, $s' := s'(1) \cdots s'(j) r_{k+1} s'(j+1) \cdots s'(p)$, where $|s'| = p$);
   (c) r_{k+1} depends on some rule in s', and let j be the biggest index such that r_{k+1} depends on $s'(j)$. Then, insert r_{k+1} immediately after $s'(j)$ (as above).

The fact that $\mathcal{D}'$ defined by s' is a left-to-right derivation follows directly from the construction above.

□

Remark 1 In [3], Book and Otto introduce the concept of a *leftmost derivation* for STS's. Let T be an STS over an alphabet Σ. A derivation step $u \Rightarrow_T v$ is called a *leftmost derivation step* if the following hold:

(i) there is a rule $\alpha \to \beta \in T$ such that $u = u_1\alpha u_2$ and $v = u_1\beta u_2$;
(ii) for every rule $\alpha' \to \beta' \in T$ such that $u = u'_1\alpha' u'_2$ we have
   - $u_1\alpha$ is a proper prefix of $u'_1\alpha'$, or
   - $u_1\alpha = u'_1\alpha'$ and u_1 is a proper prefix of u'_1, or
   - $u_1 = u'_1$ and $\alpha = \alpha'$.

A derivation is called *leftmost* if each step of it is a leftmost derivation step.

Every two consecutive leftmost derivation steps have the property that the latter one is not to the left of the first one (otherwise, (ii) is contradicted). Therefore, every leftmost derivation is a left-to-right derivation, but the converse does not hold. As a conclusion, derivations of STS are not generally equivalent to leftmost derivations.

4.3. The SES-solution

If the step j (using the rule $r_j : \alpha_j \to \beta_j$) of a derivation depends on a step i (using the rule $r_i : \alpha_i \to \beta_i$), then α_j uses, directly or indirectly, subwords of β_i.

Left-to-right derivations of monadic STS have the interesting property that whenever a step j depends on a step i then it uses all of the right hand side of the rule r_i. This property is crucial for the results to be proved in this section.

Let $G = (V_N, V_T, X_0, P)$ be a regular (right-linear) grammar without unit productions (i.e., rules $A \to B$ where $A, B \in V_N$), and let T be a finite monadic STS over V_T. We consider the SE-system $H = (V, L_1, L_2, S)$, where

- $V = V_N \cup V_T$,
- $L_1 = \{X_0\}$,
- $L_2 = \{A\beta | A \to \beta \in P\} \cup \{\alpha A \beta A | A \in V_N \ \wedge \ \alpha \to \beta \in T\}$,
- $S = V_N \cup \{\alpha A | A \in V_N \ \wedge \ (\exists \beta)(\alpha \to \beta \in T)\}$.

Then, H is an SE-system of type (f, f, f) and, from Theorem 2 it follows that $L^{r^-}(H)$ is regular.

With the notation above we have;

Theorem 4 $\Delta_T^*(L(G)) = L^{r^-}(H) \cap V_T^*$.

Proof. A derivation in G,

$$X_0 \Rightarrow_G a_1 A_1 \Rightarrow_G \cdots \Rightarrow_G a_1 \cdots a_{n-1} A_{n-1} \Rightarrow_G a_1 \cdots a_{n-1} a_n,$$

is simulated in H by synchronized extensions to the right, that is

$$X_0 \Rightarrow_{r^-} a_1 A_1 \Rightarrow_{r^-} \cdots \Rightarrow_{r^-} a_1 \cdots a_{n-1} A_{n-1} \Rightarrow_{r^-} a_1 \cdots a_{n-1} a_n$$

(for some variables $A_1, \ldots, A_{n-1}$).

The action of T on $u = a_1 \cdots a_n$ is simulated at the time of generating u. Assume that u is rewritten into v by the sequnce $s = r_1 \cdots r_m$ of rules of T, and let $\mathcal{D}$ be the derivation

$$\mathcal{D}: \quad X_0 \overset{*}{\Rightarrow}_G u \overset{s}{\Rightarrow}_T v.$$

By Lemma 3 we may assume that the derivation of u into v is left-to-right. Then, $\mathcal{D}$ can be simulated by a derivation in H using the following remarks:

(1) if $u = u_1 \alpha u_2 \alpha' u_3$ and $r : \alpha \to \beta, r' : \alpha' \to \beta' \in T$, then the derivation

$$X_0 \overset{*}{\Rightarrow}_G u = u_1 \alpha u_2 \alpha' u_3 \overset{rr'}{\Longrightarrow}_T u_1 \beta u_2 \beta' u_3$$

can be simulated in H by

$$\begin{aligned} X_0 &\overset{*}{\Rightarrow}_{r^-} u_1 \alpha A \\ &\Rightarrow_{r^-} u_1 \beta A \\ &\overset{*}{\Rightarrow}_{r^-} u_1 \beta u_2 \alpha' B \\ &\Rightarrow_{r^-} u_1 \beta u_2 \beta' B \\ &\overset{*}{\Rightarrow}_{r^-} u_1 \beta u_2 \beta' u_3, \end{aligned}$$

for some variables A and B;

(2) if $u = u_1 u_2 \alpha u_3 u_4$, $r : \alpha \to \beta$, $\alpha' = u_2 \beta u_3$ and $r' : \alpha' \to \beta' \in T$, then the derivation

$$X_0 \overset{*}{\Rightarrow}_G u = u_1 u_2 \alpha u_3 u_4 \overset{r}{\Rightarrow}_T u_1 u_2 \beta u_3 u_4 \overset{r'}{\Rightarrow}_T u_1 \beta' u_4$$

can be simulated in H by

$$\begin{aligned} X_0 &\overset{*}{\Rightarrow}_{r^-} u_1 u_2 \alpha A \\ &\Rightarrow_{r^-} u_1 u_2 \beta A \\ &\overset{*}{\Rightarrow}_{r^-} u_1 u_2 \beta u_3 B \\ &\Rightarrow_{r^-} u_1 \beta' B \\ &\overset{*}{\Rightarrow}_{r^-} u_1 \beta' u_4; \end{aligned}$$

(3) since the right hand side of each rule in T is either a symbol or the empty word, every two rules r and r' that are applied successively can be related either as in (1) or as in (2).

Therefore, every derivation in G from X_0 to a word $u \in V_T^*$ followed by a derivation in T from u into a word v can be simulated by a derivation in H from X_0 into v.

Conversely, it is trivial to see that every derivation in H leading to a word $v \in V_T^*$ is a combination between a derivation in G from X_0 into a word $u \in V_T^*$ followed then by a derivation in T from u into v.

As a conclusion, $\Delta_T^*(L(G)) = L^{r^-}(H) \cap V_T^*$.

□

Corollary 5 *For every right-linear grammar $G = (V_N, V_T, X_0, P)$ and every monadic STS T over V_N, the language $\Delta_T^*(L(G))$ is regular.*

Consider now the complexity of our construction. To the extending language L_2 we take a string for each production in G, and a string for each production in G and a rule in T. Hence, the cardinality of L_2 is $\mathcal{O}(p \cdot r)$, where p and r are the number of rules in T and G, respectively. To the synchronization set S we take a string for each nonterminal in G, and a string for each nonterminal in G and a left hand side of a rule in T. Thus, the sum of the cardinalities of L_2 and S is $\mathcal{O}(p \cdot r)$. This gives the time and space bounds for our construction.

Theorem 6 *Let G and T be as above. Then, an SE-system H of type (f, f, f) simulating the computation of $\Delta_T^*(L(G))$ can be constructed in $\mathcal{O}(|P| \cdot |T|)$ time and space.*

Theorem 6 gives the complexity of *constructing* H. Note that the complexity is dominated by the size of the output; the algorithm itself is straightforward. Implementing the computation defined by H is also possible to perform very efficiently. Namely, we can store V_N, V_T, S, and T in arrays, and maintain the current string in a stack. A simulation step simply rewrites the right hand side end of the current string.

References

[1] R.V. Book, Decidable sentences of Church-Rosser congruences. *Theoret. Comput. Sci.* **24** (1983), 301–312.

[2] R. V. Book, F. Otto, Cancellation rules and extended word problems. *Inform. Process. Lett.* **20** (1985), 5–11.

[3] R. V. Book, F. Otto, *String Rewriting Systems.* Springer-Verlag, 1993.

[4] A. Bouajjani, J. Esparza, A. Finkel, O. Maler, P. Rossmanith, B. Willems, P. Wolper, An efficient automata approach to some problems on context-free grammars. *Inform. Process. Lett.* **74** (2000), 221–227.

[5] J. Esparza, P. Rossmanith, An automata approach to some problems on context-free grammars. In: *Foundations of Computer Science: Potential, Theory, Cognition*, LNCS **1337**, Springer-Verlag, 1997, 143–152.

[6] J. Esparza, P. Rossmanith, S. Schwoon, A uniform framework for problems on context-free grammars. *Bull. EATCS* **72** (2000), 169–177.

[7] T. V. Griffiths, Some remarks on derivations in general rewriting systems. *Inform. Control* **12** (1968), 27–54.

[8] F. Otto, Some undecidability results for non-monadic Church-Rosser Thue systems. *Theoret. Comput. Sci.* **33** (1984), 261–278.

[9] F. L. Ţiplea, E. Mäkinen, A Note on synchronized extension systems. *Inform. Process. Lett.* **79** (2001), 7–9.

[10] F.L. Ţiplea, E. Mäkinen, A note on SE-systems and regular canonical systems. *Fundam. Inf.* (to appear).

[11] F.L. Ţiplea, E. Mäkinen, C. Apachiţe. Synchronized extension systems. *Acta Inform.* **37** (2001), 449-465.