On efficient reduction-algorithms for some trace rewriting systems

Michael Bertol and Volker Diekert

Universität Stuttgart, Institut für Informatik
Breitwiesenstr. 20-22, D-70565 Stuttgart

Abstract. We consider some basic problems on the decidability and complexity of trace rewriting systems. The new contribution of this paper is an $\mathcal{O}(n \log(n))$ algorithm for some computing irreducible normal forms in the case of certain one-rule systems.

1 Introduction

The notes of this paper are based on the Font Romeu Lecture and on an invited lecture at FCT-93 conference [11] of the second author. In the first part of the paper we give some basic background about trace rewriting systems. There is some overlap with the published notes from FCT-93. However, the second part is original and constitutes a new contribution to the theory of trace rewriting systems.

The theory of rewriting over free partially commutative monoids (trace rewriting) combines combinatorial aspects from string rewriting (modulo a congruence) and graph rewriting. This restriction of graph rewriting leads to feasible algorithms, but some interesting complexity questions are still open. For example, a challenging open problem is to improve the known quadratic time bound for the non-uniform complexity of length-reducing systems.

We present an $\mathcal{O}(n \log(n))$ algorithm for computing irreducible normal forms in the case of certain one-rule systems. The time bounds known before were either quadratic or linear.

We assume the reader is familiar with the concept of Mazurkiewcz traces [16]. For the background material we refer to [1, 9, 17] and to the forthcoming book on traces [12]. For basic notions on rewriting systems see [2, 14] or any other chapter of the present lecture notes.

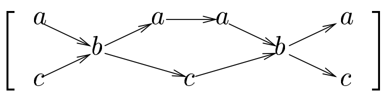
2 Notations

By Σ we denote a finite alphabet, $D \subseteq \Sigma \times \Sigma$ is a reflexive and symmetric dependence relation. The complement $I = \Sigma \times \Sigma \setminus D$ is called the independence relation and the quotient monoid $\mathbb{M} = \mathbb{M}(\Sigma, D) = \Sigma^* / \{ab = ba \mid (a, b) \in I\}$ is called the free partially commutative monoid. This monoid has been introduced in combinatorics by Cartier and Foata [3]. In computer science it is known as trace monoid, introduced by Mazurkiewicz [16]. Accordingly an element $t \in \mathbb{M}$

is a trace. We denote by $|t|$ the length of a trace t, by $|t|_a$ its a-length for some $a \in \Sigma$, and by $\mathrm{alph}(t) = \{a \in \Sigma \mid |t|_a \geq 1\}$ its alphabet. It is convenient to extend the independence relation to a relation over $\mathbb{M}$. For $u, v \in \mathbb{M}$ we define $(u, v) \in I$, if we have $\mathrm{alph}(u) \times \mathrm{alph}(v) \subseteq I$. A trace, i.e., a congruence class, may be viewed as a dependence graph as follows.

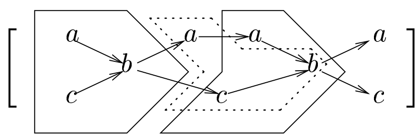
Let $a_1 \cdots a_n \in \Sigma^*$ be a word, $a_i \in \Sigma$ for $1 \leq i \leq n$. Then the corresponding trace $[a_1 \cdots a_n] \in \mathbb{M}$ is given by the isomorphism class of a node-labeled acyclic graph $[V, E, \lambda]$ where the set of vertices is any n-point set, say $V = \{1, \ldots, n\}$ with the labeling $\lambda(i) = a_i$, and arcs are from i to j if both $i < j$ and $(a_i, a_j) \in D$. Since the whole informaition is contained in the Hasse diagram we can drop the transitive suplementable edges.

Example 1. Let the dependence alphabet (Σ, D) be $(\{a, b, c\}, \{\{a, b\}, \{b, c\}\}) = [a-b-c]$ and the trace $t = [acbacabca] = \{acbaacbac, acbaacbca, \ldots, cabcaabca\}$. Then the corresponding graph looks like

$t = [acbacabca] \cong$ 

It is convenient to distinguish between the notions of subtrace and factor. A *subtrace* s of t is a subgraph of the dependence graph of t such that all directed paths starting and ending in s are entirely contained in s. Every subtrace s of t is a factor, i.e., we can write $t = usv$, but this factorization is not unique, in general. The other way, if $t = usv$ then the traces u, s and v can be identified as subtraces of t. Of course, different subtraces may be equal as elements of $\mathbb{M}$. To be precise, if we speak of a subtrace $s \subseteq t$, we mean that we have identified the vertices belonging to s in the graph of t.

Example 2. In the graph of the trace in the example above we find two subgraphs that are subtraces of t that represents the trace $l = [acb]$. The occurrences are shaped by polygons. The dotted polygon is a subgraph which is no factor.

$t = [acb][a][acb][ca] \cong$ 

A dependence alphabet (Σ, D) (independence alphabet (Σ, I) resp.) will be viewed as an undirected graph where edges are between different dependent (independent resp.) letters. A trace is called *connected*, if the dependence graph t is connected or, equivalently, if $\mathrm{alph}(t)$ induces a connected subgraph in (Σ, D).

Example 3. The trace t in the example above is connected since it contains a letter $b \in \Sigma$ and the induced subgraph of the dependence alphabet $[a - b - c]$ is also connected. The trace t' with $\mathrm{alph}(t') = \{a, c\}$, i.e., $t' = [ac]$ is not connected since there is no path from a vertex marked with a to a c-marked. The induced graph of the dependence alphabet $(\{a, c\}, \emptyset) = [a \quad c]$ is disconnected, too.

3 Trace rewriting systems

In this section we give some overview of known results and open questions for trace rewriting systems.

Definition 1 Trace-rewriting system. A trace-rewriting system is a (finite) set of rules $\mathcal{R} \subseteq \mathbb{M}(\Sigma, D) \times \mathbb{M}(\Sigma, D)$.

A trace-rewriting system $\mathcal{R}$ defines a reduction relation $\underset{\mathcal{R}}{\Longrightarrow}$ by $x \underset{\mathcal{R}}{\Longrightarrow} y$ if $x = ulv, y = urv$ for some $u, v \in \mathbb{M}$ and $l \to r \in \mathcal{R}$. The n-fold iteration of $\underset{\mathcal{R}}{\Longrightarrow}$ is denoted by $\overset{n}{\underset{\mathcal{R}}{\Longrightarrow}}$ for $n \in \mathbb{N}$. By $\overset{+}{\underset{\mathcal{R}}{\Longrightarrow}}$ ($\overset{*}{\underset{\mathcal{R}}{\Longrightarrow}}$ resp. , $\overset{*}{\underset{\mathcal{R}}{\Longleftrightarrow}}$ resp.) we denote the transitive (reflexive transitive resp., reflexive symmetric and transitive resp.) closure of $\underset{\mathcal{R}}{\Longrightarrow}$. The relation $\overset{*}{\underset{\mathcal{R}}{\Longleftrightarrow}}$ is a congruence and its quotient monoid is denoted by $\mathbb{M}/\mathcal{R}$.

A trace-rewriting system $\mathcal{R}$ is called:

- length-reducing, if $|l| > |r|$ for all $l \to r \in \mathcal{R}$
- Noetherian, if there is no infinite derivation chain $x_0 \underset{\mathcal{R}}{\Longrightarrow} x_1 \underset{\mathcal{R}}{\Longrightarrow} x_2 \cdots$
- confluent, if $\overset{*}{\underset{\mathcal{R}}{\Longleftrightarrow}} \subseteq \overset{*}{\underset{\mathcal{R}}{\Longrightarrow}} \circ \overset{*}{\underset{\mathcal{R}}{\Longleftarrow}}$
- locally confluent, if $\underset{\mathcal{R}}{\Longleftarrow} \circ \underset{\mathcal{R}}{\Longrightarrow} \subseteq \overset{*}{\underset{\mathcal{R}}{\Longrightarrow}} \circ \overset{*}{\underset{\mathcal{R}}{\Longleftarrow}}$
- The set of irreducible traces is $\mathrm{Irr}(\mathcal{R}) = \{t \in \mathbb{M} \mid \neg\exists s : t \underset{\mathcal{R}}{\Longrightarrow} s\}$.

It is well-known that confluence implies local confluence. If the system $\mathcal{R}$ is Noetherian, then confluence becomes equivalent to local confluence.

A classical result says that it is undecidable whether a finite semi-Thue system is Noetherian. More precisely, we can state:

Proposition 2. *It is decidable whether a finite trace-rewriting system is Noetherian if and only if* $\mathbb{M}$ *is commutative, i.e.* $\mathbb{M} = \Sigma^{\mathbb{N}}$.

Proof. If $\mathbb{M}$ is not commutative, we may use an encoding of semi-Thue systems over a two-letter alphabet. If $\mathbb{M}$ is commutative then the Noetherian property can be expressed by some Presburger formula.

The minimal number of rules which we need to have undecidability is not known. In fact, the following problem is open:

Problem 3. Given an one-rule system $\mathcal{R} = \{l \to r\}, l, r \in \mathbb{M}$. Is it decidable whether $\mathcal{R}$ is Noetherian?

For one-rule semi-Thue systems this is an outstanding open problem and for one-rule term rewriting systems the property of being Noetherian is undecidable [7]. Of course, a positive solution to Problem 3 would imply the semi-Thue case. However, due to the commutation rules it may happen that there is a negative answer in the trace case and a positive answer in the word case. This is similar to what happens for the decidability of the confluence of Noetherian systems.

Indeed, for finite Noetherian semi-Thue systems (local) confluence is decidable, since a finite system has finitely many *critical pairs*, only. For trace-rewriting systems a fully satisfactory notion of a critical pair is not available. Whatever definition we use, there must be finite Noetherian systems without any finite computable set of critical pairs. This is due to the following result of Narendran and Otto:

Proposition 4 [18]. *There exists an alphabet (Σ, D) with exactly one pair of independent letters such that the confluence of finite length-reducing trace-rewriting systems is recursively undecidable.*

Problem 5. Let (Σ, D) be a dependence alphabet such that confluence of finite Noetherian systems is decidable and let (Σ', D') be an induced subgraph of (Σ, D). Is the confluence of finite Noetherian systems decidable for (Σ', D')?

A positive solution of Problem 5 is not obvious, since the confluence of a system depends on the whole alphabet and not only on the letters occurring in the system.

Example 4. Let

$$(\Sigma, D) = [\, a — b — c \,]\ , \quad (\Sigma', D') = \begin{bmatrix} a & — & b \\ | & & | \\ d & — & c \end{bmatrix} pp$$

and $\mathcal{R} = \{ab \to ba, bc \to cb\}$. We may view $\mathcal{R}$ as a system over $\mathbb{M} = \mathbb{M}(\Sigma, D)$ or over $\mathbb{M}' = \mathbb{M}(\Sigma', D')$. The system $\mathcal{R}$ is Noetherian in both cases. This can be seen by mapping it to $\{a, b\}^* \times \{b, c\}^*$. The system $\mathcal{R} \subseteq \mathbb{M} \times \mathbb{M}$ is confluent, since every trace reduces to an unique normal form in $c^* b^* a^*$. Viewing $\mathcal{R} \subseteq \mathbb{M}' \times \mathbb{M}'$ over the monoid $\mathbb{M}'$, the confluence disappears:

$$\mathrm{Irr}(\mathcal{R}) \ni badc \underset{\mathcal{R}}{\Longleftarrow} abdc \underset{\mathcal{R}}{\Longrightarrow} adcb \in \mathrm{Irr}(\mathcal{R})$$

The minimal number of letters required to achieve the undecidability is not known. In particular, we do not know the answer for three letters.

Problem 6. Let $(\Sigma, D) = [a — b — c]$. Is it decidable whether a finite Noetherian system is confluent?

A more general problem is to find good (decidable) sufficient conditions for Noetherian systems such that confluence becomes decidable. Further details can be found in [9, Chapt.3].

4 A simple reduction algorithm

Let $\mathcal{R} \subseteq \mathbb{M} \times \mathbb{M}$ be a finite Noetherian trace rewriting system. Assume for the sake of simplicity that the system $\mathcal{R}$ is length-reducing and not part of the input. Then on input $t \in \mathbb{M}$ of length $n = |t|$ there are at most n reduction steps possible. We consider the time complexity of computing irreducible descendants $t \xRightarrow[\mathcal{R}]{*} \hat{t} \in \mathrm{Irr}(\mathcal{R})$. To be precise, we consider the non-uniform problem where the system $\mathcal{R}$ is fixed and we measure the complexity in the length of the input trace t.

> **FUNCTION** reduce(t)
>   **begin**
> $v := 1$;
>   **while** $t \neq 1$ **do**
>   let $t = at'$ for some $a \in \Sigma, t' \in \mathbb{M}$;
> $v := va; t := t'$;
>   **if** $v = v'l$ for some $l \to r \in \mathcal{R}$ **then** $v := v'; t := rt$
>   **endwhile**
>   **return** v
>   **end**

The complexity to evaluate the function reduce is linear time, $\mathcal{O}(|t|)$; if $\mathbb{M}$ is free or commutative, then we have reduce$(t) \in \mathrm{Irr}(\mathcal{R})$ for all $t \in \mathbb{M}$. Thus, if $\mathcal{R} \subseteq \mathbb{M} \times \mathbb{M}$ is a finite, confluent and length-reducing system over a free or commutative monoid $\mathbb{M}$ (i.e., $\mathbb{M} = \Sigma^*$ or $\mathbb{M} = \mathbb{N}^{\Sigma}$), then the word problem in the quotient monoid is solvable in linear time. For trace monoids in general no such positive result is known.

The basic problem is due to the fact that this procedure does not compute irreducible descendants, in general. In [10, 8] cones and blocks are introduced as some more general decidable and sufficient conditions such that irreducible descendants can be computed following the algorithmic scheme above.

In general we can use pattern matching to obtain a correct algorithm. The idea is that $\mathrm{Red}(\mathcal{R}) = \mathbb{M} \setminus \mathrm{Irr}(\mathcal{R})$ is a recognizable set. Hence, if $v \in \mathrm{Red}(\mathcal{R})$, then this can be detected by some finite state control and we can also compute a factorization $v = v'lv''$ for some $l \to r \in \mathcal{R}$ via protocols. The reduction step is then to define $v := v'$ and $t := rv''t$. We obtain a correct algorithm, but its worst case behavior is now square time.

Proposition 7. *Let $\mathcal{R} \subseteq \mathbb{M} \times \mathbb{M}$ be a finite confluent and length reducing trace rewriting system. Then the word problem of the quotient monoid $\mathbb{M}/\mathcal{R}$ is solvable in square time.*

The most interesting problem in this area is therefore:

Problem 8. Let $\mathcal{R} \subseteq \mathbb{M} \times \mathbb{M}$ be as in proposition 3. What is the time complexity of solving the word problem of the monoid $\mathbb{M}/\mathcal{R}$?

Of course, the problem above is equivalent with knowing the time complexity for computing irreducible descendants. More precisely, we can ask:

Problem 9. Let $\mathcal{R}$ be a finite length-reducing trace-rewriting system. Does there exist some linear time algorithm for computing irreducible descendants?

The representation of a trace as a tuple of words leads to efficient algorithms for factorizing and reducing traces. Since there is a couple of fast factorization algorithms (pattern matching algorithms) and reducing algorithms on free monoids Σ^*, we are done if we can synchronize our actions on each of the components. Therefore we use a well known embedding of a free commutative monoid $\mathbb{M}$ into a product of free monoids.

Lemma 10 Projection Lemma. *Let $(\Sigma, D) = \bigcup_{A\in\mathcal{C}}(A, A \times A)$ be a dependence alphabet, which is covered by a set of cliques $\mathcal{C}$. For $A \in \mathcal{C}$ let $\pi_A : \mathbb{M}(\Sigma, D) \to A^*$ denote the canonical projection which is defined by erasing all letters from $\Sigma \setminus A$. Then the (in real time computable) homomorphism*

$$\begin{aligned} \pi : \mathbb{M}(\Sigma, D) &\longrightarrow \textstyle\prod_{A\in\mathcal{C}} A^* \\ t &\longmapsto (\pi_A(t))_{A\in\mathcal{C}} \end{aligned}$$

is injective.

The Projection Lemma has different independent sources. It can be found in [15, Lemma 2.5], [4], [5, Lemma 3.1], [6, Prop. 1.1]. A more categorial form is in [13, Thm. 2.1] and [9, Thm. 1.4.4]. Here we use it for transforming traces in a convenient data structure, i.e., representing traces as n-tuples of words. Note for that this structure depends on the dependence alphabet and the clique covering.

To find an occurrence of a left hand side of some rule, we have to find a synchronous pattern in each projection of the trace. Synchronous is equivalent to the second condition in the following characterization, which is an immediate consequence of the Projection Lemma.

Corollary 11. *Let $l, t \in \mathbb{M}(\Sigma, D)$ be traces. Then the trace t contains the factor l, i.e., $t = u \cdot l \cdot v$, if and only if for all $A \in \mathcal{C}$*

- *there are factorizations $\pi_A(t) = u_A \pi_A(l) v_A$*
- *such that for all $c \in A \cap B$ the prefixes u_A and u_B have the same c-length, i.e., $|u_A|_c = |u_B|_c$.*

5 On the complexity of reducing by one-rule system

In this section we consider the time complexity of computing irreducible descendants in the case of a fixed one-rule system $\mathcal{R} = \{l \to r\}$. We will assume that $\mathcal{R}$ is Noetherian. In fact we require for the complexity results that $\mathcal{R}$ is length-reducing. As in the other sections let (Σ, D) be the underlying dependence alphabet, and $(\Sigma_A)_{A\in\mathcal{C}}$ be a family of covering cliques, i.e., $(\Sigma, D) =$

$(\bigcup_{A\in\mathcal{C}} \Sigma_A, \bigcup_{A\in\mathcal{C}} \Sigma_A \times \Sigma_A)$. We denote by $\pi_A : \mathbb{M}(\Sigma, D) \to A^*$ the corresponding projections. We restrict ourself to the one-rule case, because this case is the most easy to explain. The general idea becomes clear already in the special case. In order to have a general solution we have to solve some additional technical problems. This will (hopefully) be done elsewhere and is not part of the present notes.

The first technical assumption is that we require $\pi_A(l) \neq 1$ for all $A \in \mathcal{C}$; the second is that the left-hand side l is a connected trace. We prove that the time complexity of computing an irreducible descendant $t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t} \in \mathrm{Irr}(\mathcal{R})$ of an input trace with length $|t| = n$ is of order $\mathcal{O}(n \log n)$. This is a considerable improvement to the former known best time bound $\mathcal{O}(n^2)$.

Our algorithm is based on the notion of visible letters. If all letters of the right-hand side are visible, then we can even provide a linear algorithm. The well-known linear time algorithm in the free case is then a special case of our algorithm.

The basic data structure is the representation of a trace t as a tuple of words $(\pi_A)_{A\in\mathcal{C}}$. Furthermore, we use pointers to identify the i-th a in $\pi_A(t)$ with the i-th a in $\pi_B(t)$ for $a \in A \cap B$. Obviously, this structure can be computed in linear time from t, if t is given e.g. as a sequence of letters.

Note that we are not able to compute the complete dependence graph of t in linear time. This is in general impossible since it contains $\Theta(|t|^2)$ edges. In some sense this is the main difficulty, since we never be able to rely on the full information given by the dependence graph.

Lemma 12. *Let l be connected and $\pi_A(l) \neq 1$ for all $A \in \mathcal{C}$. Assume that $ulv = u'lv'$ for some $u, u', v, v' \in \mathbb{M}(\Sigma, D)$. Then the following implications hold:*

1. *if $|u| = |u'|$ then $u = u'$,*
2. *if $|u| \leq |u'|$ then $|\pi_A(u)| \leq |\pi_A(u')|$ for all $A \in \mathcal{C}$.*

Proof. Let t be the dependence graph of ulv and $u'lv'$. Identify the factors u and u' with a subtrace (prefix) of t. Assume that u' contains some vertex with label $a \in \Sigma$ such that this vertex is not in u. Then we have $|u|_a < |u'|_a$. Since l is connected, this implies $|u|_a < |u'|_a$ for all $a \in \mathrm{alph}(l)$. Finally, since $\pi_A(l) \neq 1$, we obtain $|u|_a < |u'|_a$ for all $a \in \Sigma$.

Note that the assumption $\pi_A(l) \neq 1$ for all $A \in \mathcal{C}$ is too strong. The lemma holds already, if l is connected and for every $a \in \Sigma$, there is some $b \in \mathrm{alph}(l)$ such that $(a, b) \in D$.

The straightforward pattern matching algorithm yields the following lemma.

Lemma 13. *Let l be connected and $\pi_A(l) \neq 1$ for all $A \in \mathcal{C}$. Let $t \in \mathbb{M}(\Sigma, D)$ be a trace, $|t| = n$, and $(u_i l v_i)_{1\leq i\leq m}$ the family of all possible factorizations $t = u_i l v_i$, ordered by the length of u_i, i.e., $|u_{i-1}| < |u_i|$ for all $1 < i \leq m$. Then we have $m \in \mathcal{O}(n)$ and there is a linear time algorithm which computes on input t the sequence of locations $((|u_{i,A}|)_{A\in\mathcal{C}})_{1\leq i\leq m}$.*

Proof. The lemma is a direct consequence of lemma 12, which yields $m \in \mathcal{O}(n)$ and a pattern matching algorithm. This algorithm is explained below.

In the following we denote by a pattern a tuple of positions $(p_A)_{A\in\mathcal{C}}$ satisfying the following conditions.

- For every $A \in \mathcal{C}$, p_A is a position in $\pi_A(t)$ denoting the beginning of a local factor $\pi_A(l)$, i.e., we have $\pi_A(t) = u_A\pi_A(l)v_A$ with $p_A = |u_A| + 1$.
- Moreover, we require, that a pattern marks synchronized factors $\pi_A(l)$, only. This means, we require that $|u_A|_c = |u_B|_c$ for all $c \in A \cap B \cap alph(l)$.

Since we can instantaneously identify the occurrences of a letter c, i.e., a vertex in the dependence graph, by using counters, the second condition will be checked by looking at the factors $\pi_A(l)$, only. Note that a pattern does not necessarily locate a factor l of t. A pattern $(p_A)_{A\in\mathcal{C}}$ is called a global factor (or factor for short) if there is a factorization $t = ulv$ such that p_A yields to the factorization of the projections $\pi_A(t) = \pi_A(u)\pi_A(l)\pi_A(v)$.

We can now compute the sequence of patterns that are pointing to factors $((p_A^{(i)})_{A\in\mathcal{C}})_{1\leq i\leq m}$ by scanning t from left to right. If we read a letter $a \in \Sigma$ we add it to all projections on A^* with $A \in \mathcal{C}$ where $a \in A$, and update the corresponding counters. We check in constant time if this completes some local factor $\pi_A(l)$. If so, we validate by using the counters whether this yields a pattern. Finally, to check whether or not this pattern points to the location of a factor we have to look at the letters $c \in \Sigma \setminus \mathrm{alph}(l)$, only. We need to know whether $|u_A|_c = |u_B|_c$ for all $c \in A \cap B$, $A, B \in \mathcal{C}$. During the reading of the input, we provided each letter $a \in \mathrm{alph}(l)$ with the information, how many letters $c \in A \setminus \mathrm{alph}(l)$ have been read before a for each $A \in C$ with $a \in A$. These numbers must be equal for letters of $\pi_A(l)$ and $\pi_B(l)$ if $c \in (A \cap B) \setminus \mathrm{alph}(l)$. To summarize, the sequence of factors is computable in linear time.

In the following we will encounter the problem how to maintain this information about the letters $c \in \Sigma \setminus \mathrm{alph}(l)$ in a reduction step.

Definition 14. We say that two patterns $p = (p_A)_{A\in\mathcal{C}}$ and $q = (q_A)_{A\in\mathcal{C}}$ have an overlap, if there is some $A \in \mathcal{C}$ such that the positions p_A, q_A differ by a length less than $|\pi_A(l)|$.

This means, if p_A and q_A yield factorizations $u_A\pi_A(l)v_A$ and $u'_A\pi_A(l)v'_A$, then the two factors $\pi_A(l)$ have a non-trivial overlap in the string $\pi_A(l)$.

Our reduction algorithm starts with a linear time preprocessing, which computes on input t the internal representation $(\pi_A(t))_{A\in\mathcal{C}}$. Moreover, we compute the chain of all factors l in t by creating an ordered list of patterns $(p_i)_{1\leq i\leq m}$ pointing to factors. Each p_i is a global factor, corresponding to a factorization $t = u_ilv_i$ such that $|u_{i-1}| < |u_i|$ for all $1 < i \leq m$. Clearly, t is irreducible if and only if $m = 0$, i.e., we terminate, if the chain of patterns is empty. Hence, we assume $m \geq 1$ and we perform a reduction step. We take any p_i, say the first p_1, and we replace each string $\pi_A(u_i)\pi_A(l)\pi_A(v_i)$ by the string $\pi_A(u_i)\pi_A(r)\pi_A(v_i)$. Note that this can be done in time $\mathcal{O}(|l|)$ by local modifications in our data structure. This yields a new trace $t' = u_i r_i v_i$. We have to reconstruct the list of factors. We may have destroyed some factors and some new factors may have been created. However, the destroyed factors are just those having an overlap with p_i. Thus there is a constant $k \leq \sum_{A \in \mathcal{C}}(|\pi_A(l)| - 1)$ such that all new factors are between p_{i-k-1} and p_{i+k+1} and they have an overlap with the position where the right-hand side r is inserted. The sequence $(p_1, \ldots, p_{i-k-1})$ remains at the beginning and $(p_{i+k+1}, \ldots, p_m)$ remains at the end of the list of factors.

Again in constant time, i.e., in time in order of $\mathcal{O}(|l|)$ we can create an ordered list of new patterns $(q_1, \ldots, q_s)$, $s \leq \sum_{A \in \mathcal{C}}(|\pi_A(l)| + |\pi_A(r)| - 1)$, which are candidates for new factors. Let q be an element of this list yielding the factorizations $u_A \pi_A(l) v_A$ for $A \in \mathcal{C}$. Then q represents a factorization, that means q points to the beginning of a factor, if and only if for all $A, B \in \mathcal{C}$, $c \in A \cap B$ we have $|u_A|_c = |u_B|_c$. This is given for $c \in \mathrm{alph}(l)$ since by building the list $(q_i)_{1 \leq i \leq r}$ the patterns are synchronized. The question is whether or not a pattern is a factor.

Definition 15. A letter $c \in \Sigma$ is called visible (by l), if for all $A, B \in \mathcal{C}$ such that $c \in A \cap B$ there are $a \in A \cap \mathrm{alph}(l)$ and $b \in B \cap \mathrm{alph}(l)$, $C \in \mathcal{C}$ with $\{a, b, c\} \subseteq C$.

It is clear that all letters of $\mathrm{alph}(l)$ are visible, simply take $a = b = c \in \mathrm{alph}(l)$. The next lemma shows that visible letters do not prevent an element of the generated list from being a factor.

Lemma 16. *Let p be a pattern of a trace t yielding the factorizations $u_A \pi_A(l) v_A$ for $A \in \mathcal{C}$. Let $A, B \in \mathcal{C}$ and $c \in A \cap B$. If c is visible, then we have $|u_A|_c = |u_B|_c$.*

Proof. As noted above, this is true for $c \in \mathrm{alph}(l)$. For $c \notin \mathrm{alph}(l)$ assume $|u_A|_c < |u_B|_c$, $a \in A \cap \mathrm{alph}(l)$, $b \in B \cap \mathrm{alph}(l)$, $C \in \mathcal{C}$ with $\{a, b, c\} \subseteq C$. Since $|u_A|_c < |u_B|_c$ there are vertices x, y, z with labels a, b, c and arcs from x to z and z to y in the dependence graph of t. Looking at the projection $\pi_C(t) = u_C \pi_C(l) v_C$, the factor contains a scattered subword acb. This is impossible, since $c \notin \mathrm{alph}(l)$.

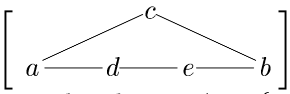
The following theorem includes the case of one-rule semi-Thue systems or vector replacement systems. Thus, for one-rule systems the results known before are generalized.

Theorem 17. *Let $\mathcal{R} = \{l \to r\}$ be a length-reducing one-rule trace rewriting system such that l is connected and for each $A \in \mathcal{C}$ the projection is not empty, i.e., $\pi_A(l) \neq 1$. Assume that all letters $c \in \mathrm{alph}(r)$ are visible by l. Then there is a linear time algorithm which computes an irreducible descendant $t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t} \in \mathrm{Irr}(\mathcal{R})$ for any input trace $t \in \mathbb{M}(\Sigma, D)$.*

Note that the visibility condition is in particular fulfilled, if $\mathrm{alph}(r) \subseteq \mathrm{alph}(l)$ or if $\mathbb{M}(\Sigma, D)$ is a free monoid. We can also generalize to the case where $\mathbb{M}(\Sigma, D)$ is a direct product of free monoids, e.g., to the case $\mathbb{N}^{\Sigma}$.

Proof. Our reduction algorithm computes during the preprocessing of the input for each $A \in \mathcal{C}$ and $a \in A$ the c-length for each letter $c \in A$ at the time when a letter a is read. This number is stored with this letter. It is used to compute the ordered list of patterns representing factors $(p_i)_{1 \le i \le m}$. Later this information is needed for invisible letters. Assume we apply the rule to the i-th factor, i.e., we replace $t = u_i l v_i$ by $t' = u_i r v_i$ where $p_i = (p_A)_{A \in \mathcal{C}}$ corresponds to $(\pi_A(u_i)\pi_A(l)\pi_A(v_i))_{A \in \mathcal{C}}$. Then we have $\pi_A(t') = \pi_A(u_i)\pi_A(r)\pi_A(v_i)$. Since all letters of $\mathrm{alph}(l) \cup \mathrm{alph}(r)$ are visible, the c-length of an invisible letter w.r.t. a position in $\pi_A(u_i)$ or $\pi_A(v_i)$ remains unchanged. It is zero if $\pi_A(u_i) = 1$ and otherwise we can simply copy the c-length of the last letter of $\pi_A(u_i)$. This is possible in constant time by our data structure. The result follows.

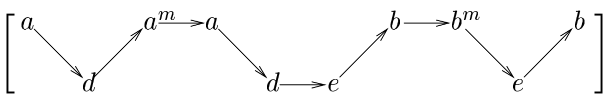
For the rest of this section we assume that $\mathrm{alph}(r)$ contains an invisible letter c. In order to clarify the difficulty we consider an example.

Example 5. Let $(\Sigma, D) =$  be the dependence alphabet and $l = [adeb]$. We have the cliques $A = \{a, c\}$, $B = \{b, e\}$, $C = \{b, c\}$, $D = \{a, d\}$ and $E = \{d, e\}$. The letter c is invisible by l, since there is no clique containing $\{a, b, c\}$. Consider two rewriting systems

$$\mathcal{R}_1 = \{l \to 1\} \text{ and } \mathcal{R}_2 = \{l \to c\}.$$

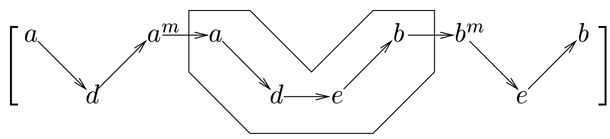
Let us study the different behavior for the reduction algorithm and the input trace $t = [ada^m adebb^m eb]$. We get the projections

$$t = \begin{pmatrix} aa^m a \\ ebb^m eb \\ bb^m b \\ ada^m ad \\ ddee \end{pmatrix} \cong$$



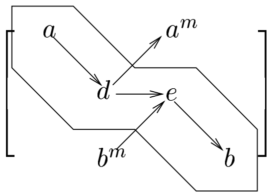
There is exactly one factor l indicated by a box in each component.

$$t = \begin{pmatrix} aa^m \boxed{a} \\ \boxed{eb}\, b^m eb \\ \boxed{b}\, b^m b \\ ada^m \boxed{ad} \\ d\, \boxed{de}\, e \end{pmatrix} \cong$$



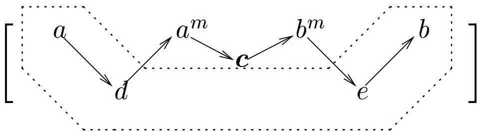
Let $t \Rightarrow_{\mathcal{R}_1} t_1$ and $t \Rightarrow_{\mathcal{R}_2} t_2$. We have a new pattern in t_1 as well as in t_2 has a candidate for a factor.

$$t_1 = \begin{pmatrix} \boxed{a}\,a^m \\ b^m\,\boxed{eb} \\ b^m\,\boxed{b} \\ \boxed{ad}\,a^m \\ \boxed{de} \end{pmatrix} \cong$$



In t_1 the marked part is a factor. This can be checked in constant time, since we can use the old c-length.

$$t_2 = \begin{pmatrix} \boxed{a}\,a^m\boldsymbol{c} \\ b^m\,\boxed{eb}\,b^m \\ \boldsymbol{c}b^m\,\boxed{b} \\ \boxed{ad}\,a^m \\ \boxed{de} \end{pmatrix} \cong$$



However the marked substrings in t_2 is not a factor. The c-length of the a is 0, whereas the c-length of all b is now 1. Updating the c-value of all $(m+1)$ letters b is certainly not the most efficient way to continue.

In order to archieve an $\mathcal{O}(n \cdot \log(n))$-algorithm we use a tree structure to represent each word $\pi_A(t)$ for $A \in \mathcal{C}$. In the following 2-3-trees would work, but we prefer AVL-trees as underlying data structure. Recall that an AVL-tree is a binary search tree. Each subtree represents a substring. With respect to the root of a subtree the left child represents a prefix and the right child the suffix. The idea is to provide each inner node with the information about the Parikh-image of the represented subtree (substring). To be more precise, let $A \in \mathcal{C}$ be a clique and $w \in A^*$. Let B be an AVL-tree representing w and x an inner node of B. Then x represents a factor w_x of $w = w_1 w_x w_2$. For each letter $c \in A$ we store the c-length $|w_x|_c$ in the node x. It is easy to see that insertion and deletion of letters is possible in logarithmic time such that the information about the Parikh-images can be maintained.

Let us return to our original problem. We have factorizations $u_A \pi_A(l) v_A$, $u_B \pi_B(l) v_B$ and for some invisible letter $c \in A \cap B$ we have to test whether $|u_A|_c = |u_B|_c$. Suppose $a \in A \cap \mathrm{alph}(l)$, $b \in B \cap \mathrm{alph}(l)$ such that a belongs to the factor $\pi_A(l)$ and b to $\pi_B(l)$ respectively. Since c is invisible, we have $c \notin \mathrm{alph}(l)$. Let x_a be the inner node of the AVL-tree for $u_A \pi_A(l) v_A$ corresponding to a. Then the left child contains the information about the c-length$|u_A|_c$. This information is available in constant time. Likewise we obtain the information about $|u_B|_c$. Therefore we can check in constant time whether a position $(p_A)_{A \in \mathcal{C}}$ is a factor. This procedure allows to state our final result.

Theorem 18. *Let $\mathcal{R} = \{l \to r\}$ be a length-reducing one-rule trace rewriting system such that l is connected and for each $A \in \mathcal{C}$ the projection is not empty,* *i.e.,* $\pi_A(l) \neq 1$. *Then there is an algorithm which computes an irreducible descendant* $t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t} \in \mathrm{Irr}(\mathcal{R})$ *for any input trace* $t \in \mathbb{M}(\Sigma, D)$ *of length* $|t| = n$ *in time* $\mathcal{O}(n \log(n))$.

6 Conclusion and future work

The main result of this paper is the $\mathcal{O}(n \log(n))$ reduction algorithm for length-reducing one-rule systems. In a forthcoming work we will study multiple rule systems. We claim that there is a general $\mathcal{O}(n \log(n))$ algorithm that manages multi-rule systems, if all left-hand sides are connected, and believe that this can extended without the hypothesis of connectness. Whether or not a linear time algorithm is possible, remains unsolved. This work seems to be more an indication that linear time is impossible, in general. However this last remark is highly tentative and might be disproved by a more clever approach.

References

1. I.J. Aalbersberg and G. Rozenberg. Theory of traces. *Theoretical Computer Science*, 60:1–82, 1988.
2. R.V. Book and F. Otto. *String Rewriting Systems.* Texts and Monographs in Computer Science. Springer Verlag, New York, 1993.
3. P. Cartier and D. Foata. *Problèmes combinatoires de commutation et réarrangements.* Number 85 in Lecture Notes in Mathematics. Springer, Berlin-Heidelberg-New York, 1969.
4. M. Clerbout. *Commutations partielles et familles de langages.* Thése, Université de Lille (France), 1984.
5. M. Clerbout and M. Latteux. Partial commutations and faithful rational transductions. *Theoretical Computer Science*, 35:241–254, 1985.
6. R. Cori and D. Perrin. Automates et commutations partielles. *R.A.I.R.O. — Informatique Théorique et Applications*, 19:21–32, 1985.
7. Max Dauchet. Termination of rewriting is undecidable in the one-rule case. In M. Chytil et al., editors, *Proceedings of the 13th Symposium on Mathematical Foundations of Computer Science (MFCS'88), Carlsbad (CSSR) 1988*, number 324 in Lecture Notes in Computer Science, pages 262–288, Berlin-Heidelberg-New York, 1988. Springer.
8. V. Diekert. Combinatorial rewriting on traces. In C. Choffrut et al., editors, *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS'90), Rouen (France) 1990*, number 415 in Lecture Notes in Computer Science, pages 138–151, Berlin-Heidelberg-New York, 1990. Springer.
9. V. Diekert. *Combinatorics on Traces.* Number 454 in Lecture Notes in Computer Science. Springer, Berlin-Heidelberg-New York, 1990.
10. V. Diekert. Word problems over traces which are solvable in linear time. *Theoret. Comput. Sci.*, 74:3–18, 1990.
11. V. Diekert. Rewriting, semi-commutations, and möbius functions. In Z. Ésik, editor, *Proc. of the 9th Fundamentals of Computation Theory (FCT 93), Szeged (Hungary) 1993*, number 710 in Lecture Notes in Computer Science, pages 1–15, Berlin-Heidelberg-New York, 1993. Springer. Invited Lecture.

12. V. Diekert and G. Rozenberg, editors. *Trace Book (preliminary title)*. 1993. To appear.
13. W. Fischer. Über erkennbare und rationale Mengen in freien partiell kommutativen Monoiden. Report FBI-HH-B-121/86, Fachbereich Informatik der Universität Hamburg, Hamburg, 1986. (Diplomarbeit 1985).
14. M. Jantzen. *Confluent String Rewriting*. EATCS Monographs on Theoretical Computer Science 14. Springer, Berlin-Heidelberg-New York, 1988.
15. R. Keller. Parallel program schemata and maximal parallelism I. Fundamental results. *Journal of the Association of Computing Machinery*, 20:514–537, 1973.
16. A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.
17. A. Mazurkiewicz. Trace theory. In W. Brauer et al., editors, *Petri Nets, Applications and Relationship to other Models of Concurrency*, number 255 in Lecture Notes in Computer Science, pages 279–324, Berlin-Heidelberg-New York, 1987. Springer.
18. P. Narendran and F. Otto. Preperfectness is undecidable for Thue systems containing only length-reducing rules and a single commutation rule. *Information Processing Letters*, 29:125–130, 1988.

This article was processed using the LaTeX macro package with LLNCS style