Trace Rewriting: Computing normal forms in time $\mathcal{O}(n \log n)$

Michael Bertol* and Volker Diekert*

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

Abstract. We develop an $\mathcal{O}(n \log n)$ algorithm for computing normal forms in the case of finite weight-reducing trace rewriting systems with connected left-hand sides. The time complexity of previously known algorithms solving this problem has been square time in the worst-case.
Keywords: Mazurkiewicz traces, rewriting modulo a partial commutation.

Introduction

Free partially commutative monoids (trace monoids) have been introduced by Cartier and Foata [6] in order to solve some combinatorial problems of rearrangements. In computer science they were investigated by Mazurkiewicz [18] as an abstract and algebraic model of concurrent processes. Trace theory became a well-established concept being used in various branches of theoretical computer science, see e.g. [13].

In order to derive practical applications we need efficient methods for various basic problems. Here we are concerned with rewriting algorithms. This is a basic operation which involves pattern matching (finding a left-hand side as a factor), transformation by performing a rewriting step, and computing irreducible descendants (normal forms). In the special case of words we find the well understood theory of string rewriting systems, see [5] or [14]. However, as soon as we introduce partial commutations we meet new phenomena. For example, surprisingly confluence of finite, length-reducing trace rewriting systems becomes undecidable [19], and decidability of confluence remains a problem even in the case of one-rule systems, c.f. [8, 20, 21].

This paper is about the time complexity of computing normal forms. The translation of the known linear time algorithms for words to traces leads to square time algorithms, only; and up to now nothing better is known for the general case. It is a challenge to improve this square time bound. Our new result presents an $\mathcal{O}(n \log n)$ time reduction algorithm in the case where the left-hand sides are connected. For this purpose we introduce a new data structure which is called

* This research has been supported in part by the ESPRIT Basic Research Action No. 6317 ASMICS II and the French-German research program PROCOPE.

trace-tree. A trace-tree realizes a representation of a trace by a tuple of words (due to the Projection Lemma) and words are represented by labeled 2-3-trees. Our result has been conjectured in [3], but the techniques being developed there for certain one-rule systems could not be applied in the present situation. It should also be noticed that our new algorithm is really of order $\theta(n \log n)$. We do not get any better performance running the algorithm on words.

Other related work is the search for linear time algorithms. The best known result with this respect is due to Bertol [2]. He developed a linear time reduction algorithm (generalizing the string case) for finite, length-reducing systems, when the underlying dependence alphabet is a cograph.

1 Notations

1.1 Traces and dependence graphs

By Σ we denote a finite alphabet, $D \subseteq \Sigma \times \Sigma$ is a reflexive and symmetric relation, the *dependence relation*. The pair (Σ, D) is called *dependence alphabet*. The complement $I = \Sigma \times \Sigma \setminus D$ is called the *independence relation* and following Mazurkiewicz [18] the quotient monoid $\mathbb{M} = \mathbb{M}(\Sigma, D) = \Sigma^*/\{ab = ba \mid (a, b) \in I\}$ is called *trace monoid*. Accordingly an element $t \in \mathbb{M}$ is called a *trace*. We denote by $|t|$ the *length* of a trace t, i.e., the length of a representing word.

A dependence alphabet (Σ, D) will be viewed as an undirected graph where edges are between different dependent letters. A trace is called *connected*, if $\mathit{alph}(t)$ induces a connected subgraph in (Σ, D). 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 $1 \le i \le n$. Then the corresponding trace $[a_1 \cdots a_n] \in \mathbb{M}$ is given by a node-labeled acyclic graph $[V, E, \lambda]$ where the set of vertices is any n-point set, say $V = \{x_1, \ldots, x_n\}$, with the labeling $\lambda(x_i) = a_i$, and arcs are from x_i to x_j if both $i < j$ and $(a_i, a_j) \in D$. It is convenient to draw the Hasse diagram only. Thus all redundant edges are omitted. A trace is connected if and only if its Hasse diagram is a connected graph.

Example 1. Let the dependence alphabet (Σ, D) be given by the undirected graph

$$(\Sigma, D) \quad = \quad a - b - c$$

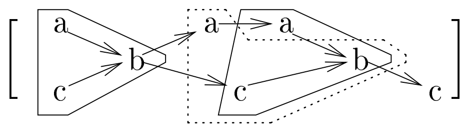
and let the trace $t = [acbacabc] = \{acbacabc, cabaacbc, \ldots\}$. Then the trace t is connected and the corresponding Hasse diagram looks like:

$$t \quad = \quad [acbacabc] \quad \cong \quad \left[\begin{tikzcd} a \arrow[dr] & & a \arrow[r] & a \arrow[dr] & & \\ & b \arrow[ur] \arrow[dr] & & & b \arrow[dr] & \\ c \arrow[ur] & & c \arrow[urr] & & & c \end{tikzcd}\right]$$

Let $[V, E, \lambda]$ be a dependence graph of some trace t and $U \subseteq V$ be any subset. Then the induced labeled subgraph by U is in fact a dependence graph. Thus, we can say that any subset $U \subseteq V$ yields a well-defined trace. The rewriting procedure is based on factorizations. A trace l is called a *factor* of t, if we can write $t = plq$ for some $p, q \in \mathbb{M}$. A factor l may occur at different places in a trace t, so there might be several factorizations, which have to be distinguished.

Example 2. In the graph of the trace in the example above we find two factors $l = [acb]$. Below the occurrences are shaped by polygons. The dotted polygon shapes a subgraph which is not a factor.

$$t \quad = \quad [acb][a][acb][c] \quad \cong$$



1.2 Trace rewriting systems

A *trace-rewriting system* is a (finite) set of rules $S \subseteq \mathbb{M}(\Sigma, D) \times \mathbb{M}(\Sigma, D)$. It defines a reduction relation $\underset{S}{\Longrightarrow}$ by $x \underset{S}{\Longrightarrow} y$ if $x = ulv, y = urv$ for some $u, v \in \mathbb{M} = \mathbb{M}(\Sigma, D)$ and $(l, r) \in S$.

By $\overset{*}{\underset{S}{\Longrightarrow}}$ ($\overset{*}{\underset{S}{\Longleftrightarrow}}$, resp.) we denote the reflexive, transitive (reflexive, symmetric, and transitive, resp.) closure of $\underset{S}{\Longrightarrow}$. The relation $\overset{*}{\underset{S}{\Longleftrightarrow}}$ is a congruence and its quotient monoid is denoted by $\mathbb{M}/S$.

A trace-rewriting system S is called *length-reducing*, if $|l| > |r|$, for all $(l, r) \in S$, *weight-reducing*, if there is a homomorphism $w : \mathbb{M} \longrightarrow \mathbb{N}$ such that $w(l) > w(r)$, for all $(l, r) \in S$. Note that every length-reducing system is weight-reducing by the weight $w(t) = |t|$ for all $t \in \mathbb{M}$. A trace rewriting system is called *Noetherian*, if there is no infinite derivation chain $x_0 \underset{S}{\Longrightarrow} x_1 \underset{S}{\Longrightarrow} x_2 \underset{S}{\Longrightarrow} \cdots$; it is called *confluent*, if $\overset{*}{\underset{S}{\Longleftrightarrow}} \subseteq \overset{*}{\underset{S}{\Longrightarrow}} \circ \overset{*}{\underset{S}{\Longleftarrow}}$, and it is *locally confluent*, if $\underset{S}{\Longleftarrow} \circ \underset{S}{\Longrightarrow} \subseteq \overset{*}{\underset{S}{\Longrightarrow}} \circ \overset{*}{\underset{S}{\Longleftarrow}}$. The set of *irreducible traces (normal forms)* is $\mathrm{Irr}(S) = \{t \in \mathbb{M} \mid \neg \exists s : t \underset{S}{\Longrightarrow} s\}$.

Remark. The following facts are known:

i) A Noetherian system is confluent if and only if it is locally confluent. Contrary to the free or free-commutative case, confluence of finite length-reducing trace rewriting systems is undecidable, see [19].
ii) If $S \subseteq \mathbb{M} \times \mathbb{M}$ is Noetherian and confluent, then an upper bound for the word problem of the quotient monoid $\mathbb{M}/S$ is given by the time complexity to compute normal forms.
iii) If $S \subseteq \mathbb{M} \times \mathbb{M}$ is weight-reducing and $\mathbb{M}$ is either free or commutative, then normal forms can be computed in linear time, [4]. This result has been generalized to the case where left-hand sides are so-called cones and blocks [11] and to the case where the dependence alphabet is a cograph [2].

iv) If $S \subseteq \mathbb{M} \times \mathbb{M}$ is length-reducing then the best known time complexity for computing normal forms is square-time by performing each reduction step in linear time. (Our new result applies only to the case where all left-hand sides are connected.)

In the following we consider a fixed weight-reducing trace rewriting system S with (connected) left-hand sides $\mathcal{L} = \{l \mid (l,r) \in S\}$. Thus, in all what follows the size of $\mathcal{L}$ and $|l|$, for $l \in \mathcal{L}$, is bounded by a constant and belongs to $\mathcal{O}(1)$.

2 Labeled 2-3-trees

We will need a certain data structure in order to represent words of C^* for $C \subseteq \Sigma$ such that C is a clique in the dependence alphabet (Σ, D), i.e., $C \times C \subseteq D$. We use a *2-3-tree* (for definitions see e.g. [1]) where the leaves are nodes with labels from C. The are ordered from left-to-right, hence the yield defines a unique word of C^*. The inner nodes v are labeled with subsets $A \subseteq C$. This subset is simply the alphabet of the yield which corresponds to the subtree with root v. Hence, if v has the children $v_1, \ldots, v_k$ labeled by $A_1, \ldots, A_k$ then v has the label $\bigcup\{A_i \mid 1 \leq i \leq k\}$. The labeling can be propagated bottom-up from the leaves to the root and the usual operations on 2-3-trees can support the correct update of the labeling. A basic feature of this data structure is that we can jump to the next position of some letter $a \in \Sigma$ in $\mathcal{O}(\log n)$ steps, where n is the number of leaves. In addition to the alphabetic information some leaves may be marked by left-hand sides of the underlying rewriting system. Thus for $l \in \mathcal{L}$ some leaves are marked with l. This marking is propagated to the root as the alphabet information. Hence an inner node is marked by some $l \in \mathcal{L}$ if and only if some child is marked by l. Our later interpretation will be that a leaf is marked by l if and only if the corresponding point is a reference point for a factor l of the trace. However for the moment this interpretation is not important. A 2-3-tree as described above is called a *labeled 2-3-tree* in the following. We may implement this data structure where nodes are labeled with subsets from $C \cup \mathcal{L}$ and leaves have markings of the form $\{a\} \cup K$, $a \in C$, $K \subseteq \mathcal{L}$. For convenience any label of a leaf will be identified with a pair of $\Sigma \times \mathcal{P}(\mathcal{L})$. Therefore a sequence of leaf-labels can be identified with a word of $(\Sigma \times \mathcal{P}(\mathcal{L}))^*$.

For 2-3-trees the following lemmata are well-known. The proofs of [1] are easily transformed to the present context.

Lemma 1. *Let T be a labeled 2-3-tree with n leaves and let $\leq$ be the ordering of the leaves. Then the following operations can be performed in $\mathcal{O}(\log n)$ time:*

i) Given leaves x, y in T. Decide whether or not $x \leq y$.
ii) Given $a \in \Sigma$ ($l \in \mathcal{L}$, resp.) such that a (l resp.) is contained in the label of the root. Compute the least (or some) leaf x which has a label a (l, resp.).
iii) Given a leaf x. Change the label of x and correspondingly of all ascendants up to the root.

iv) *Given a leaf x and a label $a \in \Sigma$ ($l \in \mathcal{L}$, resp.). Compute the least node y, such that $x \leq y$ and the label of y is a (or contains l), or report that there is no such node.*

The main advantage however is that labeled 2-3-trees support the operations *concat* and *split*.

Lemma 2. *Let T_1, T_2 be labeled 2-3-trees with at most n leaves. Let $\alpha_i \in (\Sigma \times \mathcal{P}(\mathcal{L}))^*$ the yield of T_i, $i = 1, 2$. Then we can compute in time $\mathcal{O}(\log n)$ a labeled 2-3-tree T with yield $\alpha_1\alpha_2$. This tree is denoted by $T_1 \cdot T_2$.*

In fact, the time to compute $T_1 \cdot T_2$ is only proportional to 1 plus the difference of the heights. This is crucial to obtain the split operation in time $\mathcal{O}(\log n)$. We derive the following lemma.

Lemma 3. *Let T be a labeled 2-3-tree and x be a leaf. Let $\alpha \in (\Sigma \times \mathcal{P}(\mathcal{L}))^*$ be the yield of the leaves y, with $y < x$, and $\beta \in (\Sigma \times \mathcal{P}(\mathcal{L}))^*$ the yield of the leaves z, with $x \leq z$. Then we can compute in time $\mathcal{O}(\log n)$ two labeled 2-3-trees T_1, T_2 with yield α and β respectively.*

3 Traces revisited

3.1 Jump-sequences

Above a description of a trace $t \in \mathbb{M}(\Sigma, D)$ is given by its dependence graph $t = [V, E, \lambda]$. However the number of edges is of order $\Omega(|t|^2)$, since the dependence graph of any trace of length n has at least $(1/|\Sigma|) \cdot \binom{n-1}{2}$ edges. Our aim is an $\mathcal{O}(n \log n)$ reduction algorithm, hence we may not use dependence graphs as a basic data structure. We better identify a trace t with its *Hasse diagram*. This is a labeled acyclic graph $t = [V, H, \lambda]$ such that $(x, y) \in H$ if and only if $(x, y) \in E$ and no longer path between x and y exists in the dependence graph of t. Thus, the Hasse diagram is the reduct of $[V, E, \lambda]$. Since the out-degree of any node is bounded by $|\Sigma|$, we obtain a graph with $\mathcal{O}(n)$ edges, where $n = |t|$. In fact, our final implementation will use projections on cliques. This is another representation of size $\mathcal{O}(n)$ which supports some more direct information about a trace.

Using the Hasse diagram we define the *distance* $d(x, y)$ of nodes $x, y \in V$ by the length of the shortest path between x and y in the underlying undirected graph $(V, H \cup H^{-1})$. (If there is no such path we set $d(x, y) = |V|$.) Clearly d is a metric on V. Let $U \subseteq V$ be any (connected) subset of nodes of $t \in \mathbb{M}(\Sigma, D)$. The diameter of U is defined by $d(U) = \max\{d(x, y) \mid x, y \in U\}$.

Since we deal only with factors $l \in \mathcal{L}$, we have, according to our assumptions, $|l| \in \mathcal{O}(1)$. A connected factor of constant size will correspond to a subset of constant diameter. If we can write $t = plq$ for some $p, q \in \mathbb{M}(\Sigma, D)$, then we can identify l with a subset $U \subseteq V$, $t = [V, H, \lambda]$ such that the induced subgraph of

U yields the trace l. Conversely, let $U \subseteq V$ be a subset such that the induced subgraph in $[V, H, \lambda]$ yields l. Then U defines a factor $l \in \mathcal{L}$ of t if and only if every direct path in $[V, H, \lambda]$ which starts and ends in U is completely contained in U, see [12, Lem. 1.2.5]. Although this observation is simple, it is fundamental for the following and has to be well understood.

Example 3. Let

$$(\Sigma, D) \quad = \quad \begin{tikzcd} a \arrow[r, dash] \arrow[d, dash] & b \arrow[d, dash] \\ d \arrow[r, dash] & c \end{tikzcd} \qquad t \quad = \quad \begin{tikzcd} & b \arrow[dr] & \\ a \arrow[ur] \arrow[dr] & & c \\ & d \arrow[ur] & \end{tikzcd}$$

Then the induced subgraph $b \to c$ is a factor, the subgraph $a \to b \to c$ is not a factor of t.

Definition 4. Let $t = [V, H, \lambda]$ be a trace.

i) For $x \in V$ we define the subset $\mathit{nextletter}(x) \subseteq V$ by the set of nodes y such that $(\lambda(x), \lambda(y)) \in D$ and there is a directed path in the Hasse diagram (or dependence graph) from x to y, and there is no such path from x to y visiting a node z with $x \neq z \neq y$ and $\lambda(z) = \lambda(y)$.

ii) A *jump-sequence* from x to y is a sequence $x = x_0, x_1, \ldots, x_k = y$, $x_i \in V$, $k \geq 1$ such that for all $1 \leq i \leq k$ we have $x_i \in \mathit{nextletter}(x_{i-1})$.

iii) The label of a *jump-sequence* $(x_0, \ldots, x_k)$ is the word $\lambda(x_0) \cdots \lambda(x_k) \in \Sigma^*$.

Example 4. Let (Σ, D) as in the example above and $t = a(bc)^n d$. Then ad is a quasi-factor and there is a jump-sequence with label $abcd$ since d is in the *nextletter*-relation already for the first c.

Remark. The reader may ask why we do not use a graph representation of a trace where arcs are drawn from x to y if and only if $y \in \mathit{nextletter}(x)$. This is a graph with $\mathcal{O}(n)$ edges which contains the Hasse diagram. However, a direct storage of this graph would lead to expensive concat- and split-operations. Consider the concatenation of a^n with a single b, $(a, b) \in D$, $a \neq b$. This causes n new edges. The situation is not better if we remove b from $a^n b$. If we delete the first b of $a^n b^2$, we have a change of n edges.

3.2 Projection on cliques

Let $\mathcal{C}$ be a covering of cliques of the dependence alphabet (Σ, D), i.e., $\Sigma = \bigcup\{C \mid C \in \mathcal{C}\}$ and $D = \bigcup\{C \times C \mid C \in \mathcal{C}\}$. For each $C \in \mathcal{C}$ we have the canonical homomorphism

$$\pi_C : \mathbb{M}(\Sigma, D) \longrightarrow C^*$$

which is defined by erasing all letters of a trace which do not belong to C, i.e., $\pi_C(a) = a$, for all $a \in C$, and $\pi_C(a) = 1$, for all $a \in \Sigma \setminus C$. If $t = [V, H, \lambda]$ and

$U \subseteq V$ then we may apply π_C to U and we define $\pi_C(U) \in C^*$ as follows. First we consider the labeled partial order $(U, \leq)$ which is induced by t. Then we erase all nodes with label outside C. We obtain a total order, which defines the string in C^*. More precisely, this string is a *scattered subword*, [16], of $\pi_C(t) \in C^*$. The well-known Projection Lemma (see [7], [9, Prop. 1.1] or [15, Lem. 2.5]) states that the homomorphism π is injective, where

$$\pi : \mathbb{M}(\Sigma, D) \longrightarrow \prod_{C \in \mathcal{C}} C^*, \quad t \longmapsto \left(\pi_C(t)\right)_{C \in \mathcal{C}}.$$

Definition 5. Let $l \in \mathcal{L}$ and $t = [V, H, \lambda] \in \mathbb{M}(\Sigma, D)$ be a trace. A subset $U \subseteq V$ is called a *quasi-factor for* l, if for all $C \in \mathcal{C}$ the scattered subword $\pi_C(U)$ is a factor of $\pi_C(t)$ and this factor is equal to $\pi_C(l)$. Hence for any $C \in \mathcal{C}$, we can write $\pi_C(t) = p \cdot \pi_C(U) \cdot q$ for some $p, q \in C^*$ and $\pi_C(U) = \pi_C(l)$.

Lemma 6. *There exists a constant $k_0 \leq |\Sigma|$ (depending on (Σ, D) only) such that the following holds: Let $t = [V, H, \lambda]$ be a trace and $U \subseteq V$ be a quasi-factor for $l \in \mathcal{L}$. Then U is* **not** *a factor of t if and only if there exists a jump-sequence $(x_0, \ldots, x_k)$ in V with $x_0, x_k \in U$, $k \leq k_0$ and $x_i \notin U$, for all $1 \leq i < k$.*

The proof is an induction being left to the reader. Note that U is a quasi-factor, otherwise the assertion would not hold. Consider e.g. the first and last point of the one-letter trace a^n.

A *position* of $l \in \mathcal{L}$ is a pair (a, i) such that $a \in alph(l)$ and $1 \leq i \leq |l|_a$. The position defines the i-th a of l. For each $l \in \mathcal{L}$ we fix exactly one position, this position is called the *reference point.*

If $U \subseteq V$ is a quasi-factor for l of some trace $t = [V, H, \lambda]$ then each position of l can be identified, by counting the letters only from U, with a unique point of U. In particular the reference point of l is defined as a fixed node of U.

Proposition 7. *Let k_0 be the value of Lemma 6. Then for each $l \in \mathcal{L}$ we may compute in preprocessing time (i.e., before we consider any input) a finite list of* forbidden words *$\{(x_1, y_1, u_1), \ldots, (x_m, y_m, u_m)\}$ where x_i, y_i are positions of l and $u_i \in \Sigma^*$ with $|u_i| \leq k_0$, $1 \leq i \leq m$, such that the following becomes true:*

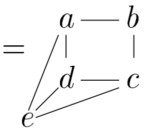
A quasi-factor U for l of a trace $t = [V, H, \lambda]$ is **not** *a factor if and only if there exists a forbidden word (x, y, u) from the list of l and a jump-sequence of t with label u which starts in $x \in U$, ends in $y \in U$, and no intermediate node belongs to U.*

Proof. This is an immediate consequence of Lemma 3 and Lemma 6.

Note that the list of forbidden words from Prop. 7 is of constant size. In many cases it will be very short, e.g., for free monoids the list is always empty since the notion of factor and quasi-factor then coincides. The interested reader may also consult [12] for the definition of *cones* and *blocks*. It turns out that for some covering by maximal cliques cones and blocks have empty lists of forbidden words. In the general case for designing a more efficient algorithm it might be useful to calculate the list of forbidden words dynamically. We omit a detailed analysis here.

Example 5. For

$$(\Sigma, D) \quad =$$



and $l = abc$ the set of forbidden words is simply

$$\{(a,1),(c,1),adc),((a,1),c,1),aec)\}.$$

Let $t = [V, H, \lambda]$ be a trace and $X \subseteq V$ a subset. By $\nabla_X(t)$ we denote the smallest suffix of t which contains X. In particular, $t = p \cdot \nabla_X(t)$, for some p, and p is the maximal prefix of t which does not contain any node of X. Note that if y is a minimal point of the word $\pi_C(\nabla_X(t))$, $C \in \mathcal{C}$, then there exists some $x \in X$ and a jump-sequence from x to y of length smaller than $|\Sigma|$.

We end this section with an example which illustrates some of the notions which has been introduced so far.

Example 6. Let

$$(\Sigma, D) \quad = \quad \begin{array}{ccccc} & & c_1 & & \\ a & b & & d & e \\ & & c_2 & & \end{array}$$

and S be the following four rules where all left-hand sides are connected:

$$\begin{aligned} b^2c_1d^2 &\underset{S}{\Longrightarrow} b^2d^2 \\ b^2c_2 &\underset{S}{\Longrightarrow} b^2c_1 \\ d^2c_2b^2 &\underset{S}{\Longrightarrow} bc_2d \\ ae &\underset{S}{\Longrightarrow} c_2 \end{aligned}$$

Start with a trace $t = a^kb^{2n}c_1d^{2n}e^k$, $k \geq 1$ and $n \geq 2$. Notice that the quasi-factor ae is no factor (consider the jump-sequence with label abc_1de) and that vertices labeled a, e of t have a distance at least $4n + 2$. The first reduction step yields the trace

$$t_1 \quad = \quad \begin{tikzcd}[row sep=tiny] a^k \arrow[r] \arrow[dr] & b^{2n} \\ d^{2n} \arrow[r] & e^k \end{tikzcd}$$

The next reduction step yields

$$t_2 \quad = \quad \begin{tikzcd}[row sep=tiny] a^{k-1} \arrow[dr] & & b^{2n} \\ & c_2 \arrow[ur] \arrow[dr] & \\ d^{2n} \arrow[ur] & & e^{k-1} \end{tikzcd}$$

After n more steps, we may obtain $t_3 = a^{k-1}b^n c_2 d^n e^{k-1}$, and, since $n \geq 2$, we rewrite to $t_4 = a^{k-1}b^n c_1 d^n e^{k-1}$. We finish our calculation here since t_4 is of a similar structure like t.

4 Trace-trees and rewriting

We are now able to describe the data structure for traces which is used in the rewriting procedure. We start with a trace $t = [V, H, \lambda]$. Then we store V as a labeled set, for each $C \in \mathcal{C}$ we build a labeled 2-3-tree $T_C(t)$, see Section 2, which has as leaves the subset $V_C = \{x \in V \mid \lambda(x) \in C\}$ such that $T_C(t)$ defines exactly the word $\pi_C(t)$. Note that the set V together with the family $\left(T_C(t)\right)_{C\in\mathcal{C}}$ contains the whole information about t. If $|t| = n$ then the size of this data structure is of order $\mathcal{O}(n)$. Now consider all $U \subseteq V$ such that U is a subset corresponding to a factor $l \in \mathcal{L}$. Each U has by the condition above a reference point. We mark this point by l. We propagate the marking to the root of $T_C(t)$ thereby defining our final object which is called *trace-tree*. Thus, a trace-tree is a tuple $(V, (T_C)_{C\in\mathcal{C}})$ where in particular T_C are labeled 2-3-trees with the correct markings. Note that we are able to check in the root of $T_C(t)$ whether or not the trace t has a factor $l \in \mathcal{L}$, i.e., whether or not t is irreducible.

The outline of the rewriting procedure is as follows: We start with $t = [a_1 \cdots a_n]$, $a_i \in \Sigma$ $(1 \leq i \leq n)$. Then we use n trace-concat operations (see below) to build up the trace-tree of t. If t is reducible then we find a reference point for some l, where $(l, r) \in \mathcal{S}$. The reference point defines a subset $U \subseteq V$ of constant diameter which corresponds to the factor l. We compute a factorization $t = plq$ such that $lq = \nabla_U(t)$. We split the trace-tree into the trees T_1, T_2 such that T_1 yields p and T_2 yields q. Then we use two trace-concat operations in order to compute $t' = prq$. This realizes one rewriting step. If finally the trace becomes irreducible, then we can read off the trace as a sequence in linear time.

Lemma 8. *Let t be given as a trace-tree (without a correct marking of left-hand sides), $n = |t|$, and $x \in V$ be a vertex. If $l \in \mathcal{L}$ is connected then we can decide in time $\mathcal{O}(\log n)$ whether x is a reference point of a quasi-factor U representing a factor l. Moreover, the factor U is unique in this case.*

Proof. Given x we can reach every point in the set V of the trace-tree at constant distance in time $\mathcal{O}(1)$. Since l is connected, the diameter of a possible U is constant, hence we can check in constant time whether there is a quasi-factor U for l with reference point x. (That means the yield word $w \in C^*$ of each tree $T_C(t)$ factorizes $p_C \pi_C(U) q_C$, for all $C \cap alph(l) \neq \emptyset$, the reference point x belongs to a leaf in a yield word $\pi_C(U)$, hence to a leaf which belongs to U.) Note that already this quasi-factor is unique. Now we have to check the forbidden words in the list of U. Since the trace-tree allows to compute the relation *nextletter* in time $\mathcal{O}(\log n)$, see Lemma 1, we can compute jump-sequences of constant length in time $\mathcal{O}(\log n)$.

Example 7. The assumption of connectedness is essential. Let $(a, b) \in I$ and $t = a^n b^n$, $l = ab$ with reference point $(a, 1)$. Then t contains n^2 factors U for l, but it has n reference points, only.

The connectedness of l is also crucial in the following lemma.

Lemma 9. *Let $t_i = [V_i, H_i, \lambda_i]$ be traces, $i = 1, 2$, and $t = t_1 \cdot t_2 = [V, H, \lambda]$. with $V = V_1 \dot{\cup} V_2$. Let $x \in V$ be a reference point of some factor U such that U corresponds to some connected trace $l \in \mathcal{L}$. If $U \cap V_i \neq \emptyset$ for $i = 1, 2$, then x is at constant distance either to some maximal point of t_1 or to some minimal point of t_2.*

Proof. Let $y, z \in U$ such that $y \in V_1$ and $z \in V_2$. Then the Hasse diagram contains an undirected path from y to z of constant length. This path leaves V_1 at a maximal point and enters V_2 at a minimal point at least one time.

We now can describe the trace-concat operation: Let $t_i = [V_i, H_i, \lambda_i]$ be traces, $i = 1, 2$, as above, $|t_1|, |t_2| \leq n$. Let T_1, T_2 be the corresponding trace-trees. By the results in Section 2 we can concatenate T_1 and T_2 and we obtain an object of the form

$$T \quad = \quad \left(V_1 \dot{\cup} V_2, \left(T_{1,C} \cdot T_{2,C}\right)_{C \in \mathcal{C}}\right)$$

in time $\mathcal{O}(\log n)$. However, this is not yet a trace-tree, in general. We have to introduce markings to reference points which correspond to factors U with $U \cap V_i \neq \emptyset$ for both, $i = 1, 2$. By Lemma 9 we find all reference points in time $\mathcal{O}(\log n)$. Finally, we update T such that it becomes a trace tree using Lemma 1 in time $\mathcal{O}(\log n)$. We may denote this trace tree by $T_1 \cdot T_2$. We have just shown:

Theorem 10. *The trace-concat operation can be performed in logarithmic time, if all left-hand sides are connected.*

We now describe a rewriting step in detail. Let $t = [V, H, \lambda]$, $|t| = n$, given as a trace-tree such that a root contains the marking of a left-hand side. Then we find in time $\mathcal{O}(\log n)$ a reference point x of a factor U. Let X be the set of minimal elements of U. Consider $\nabla_X(t) = \nabla_U(t)$ and the corresponding factorization $t =$

$p \cdot \nabla_X(t)$. A simple reflection yields that for each $C \in \mathcal{C}$ we find in time $\mathcal{O}(\log n)$ either $\pi_C(p) = 1$ or we find a leaf x_C in $T_C(t)$ such that $\pi_C(p)$ corresponds to the leaves y with $y < x_C$. This procedure uses jump-sequences and Lemma 1. Details are left to the reader. Then we split all trees $T_C(t)$ at these points x_C, $C \in \mathcal{C}$. Since U is a factor, these splits are consistent. Thus in time $\mathcal{O}(\log n)$ we have created two families $(T_{i,C}(t))_{C\in\mathcal{C}}$, $i = 1, 2$. The consistency implies that we can define V_i as the set of leaves in family i, $i = 1, 2$. We have $V = V_1 \dot\cup V_2$. Although the reference points to the factors are not correct yet, we see that $(V_1, (T_{1,C}(t))_{C\in\mathcal{C}})$ yields the trace p and $(V_2, (T_{2,C}(t))_{C\in\mathcal{C}})$ yields $\nabla_U(t)$. Since we know that l is a prefix of $\nabla_U(t)$ we may delete all nodes of U from $(V_2, (T_{2,C})_{C\in\mathcal{C}})$ in time $\mathcal{O}(\log n)$. We finish up this step with two structures T_1 and T_2 where, up to reference points, T_1 is a trace-tree of p and T_2 is a trace-tree of q and $t = plq$. The remaining problem is that T_1 and T_2 may have reference points where the corresponding factors have now disappeared. But by Lemma 9 we find all these possible points in constant time. We check whether they are still reference points of factors, and if not, then we update the labeled 2-3-trees. Thus, after time $\mathcal{O}(\log n)$ we have trace-trees representing p and q. The right-hand side r is a trace-tree of constant size. Two more trace-concat operations conclude the rewriting step.

Therefore we can state our main result.

Theorem 11. *Let $t = [V, H, \lambda]$ be given as a trace-tree T and $|t| = n$. If all left-hand sides of S are connected, then a rewriting step $t = plq \underset{S}{\Longrightarrow} prq = t'$ can be computed in time $\mathcal{O}(\log n)$ by constructing a trace-tree T' of t'.*

Corollary 12. *Let $S \subseteq \mathbb{M} \times \mathbb{M}$ be a finite Noetherian system where all left-hand sides are connected, and let $d_S(n)$ be the maximal number of rewriting steps which are possible on traces of length n. Then we can compute normal forms in time $\mathcal{O}(d_S \cdot \log(d_S))$.*

Corollary 13. *Let M be a monoid which is presentable by some finite confluent, and weight-reducing trace rewriting system where all left-hand sides are connected. Then the word problem of M is decidable in time $\mathcal{O}(n \log n)$.*

References

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, M.A., 1974.
2. M. Bertol. Efficient rewriting in cograph trace monoids. In *Proceedings of the Fundamentals of Computation Theory (FCT '95), Dresden (Germany) 1995*, Lecture Notes in Computer Science, Berlin-Heidelberg-New York, 1995. Springer.
3. M. Bertol and V. Diekert. On efficient reduction-algorithms for some trace rewriting systems. In H. Common and J.-P. Jouannaud, editors, *Proceedings of the Term Rewriting.*, number 909 in Lecture Notes in Computer Science, pages 114–126, Berlin-Heidelberg-New York, 1993. Springer.

4. R. Book. Confluent and other types of Thue systems. *J. Assoc. Comput. Mach.*, 29:171–182, 1982.
5. R.V. Book and F. Otto. *String Rewriting Systems.* Texts and Monographs in Computer Science. Springer Verlag, New York, 1993.
6. P. Cartier and D. Foata. *Problèmes combinatoires de commutation et réarrangements.* Number 85 in Lecture Notes in Mathematics. Springer, Berlin-Heidelberg-New York, 1969.
7. M. Clerbout. *Commutations Partielles et Familles de Langages.* Thèse, Université des Sciences et Technologies de Lille (France), 1984.
8. M. Clerbout, M. Latteux, Y. Roos, and P. Wacrenier. A result on confluence for one-rule trace-rewriting systems. Technical Report IT-95-275, Laboratoire d'Informatique Fondamentale de Lille, Universite des Sciences et Technologies de Lille, Apr. 1995.
9. R. Cori and D. Perrin. Automates et commutations partielles. *R.A.I.R.O. — Informatique Théorique et Applications*, 19:21–32, 1985.
10. V. Diekert. On the Knuth-Bendix completion for concurrent processes. In Th. Ottmann, editor, *Proceedings of the 14th International Colloquium on Automata, Languages and Programming (ICALP'87), Karlsruhe (FRG) 1987*, number 267 in Lecture Notes in Computer Science, pages 42–53, Berlin-Heidelberg-New York, 1987. Springer. Appeared also in a revised version in Theoretical Computer Science 66:117-136, 1989.
11. V. Diekert. Word problems over traces which are solvable in linear time. In B. Monien et al., editors, *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS'89), Paderborn (FRG) 1989*, number 349 in Lecture Notes in Computer Science, pages 168–180, Berlin-Heidelberg-New York, 1989. Springer. Revised and extended version in Theoretical Computer Science 74 (1990) 3-18.
12. V. Diekert. *Combinatorics on Traces.* Number 454 in Lecture Notes in Computer Science. Springer, Berlin-Heidelberg-New York, 1990.
13. V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.
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. M. Lothaire. *Combinatorics on Words.* Addison-Wesley, Reading, M.A., 1983.
17. E. W. Mayr and A. R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Math.*, 46:305–329, 1982.
18. A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.
19. 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.
20. F. Otto. On confluence versus strong confluence for one-rule trace-rewriting systems. *Mathematical Systems Theory*, 28:363–384, 1995.
21. C. Wrathall and V. Diekert. On confluence of one-rule trace-rewriting systems. *Mathematical Systems Theory*, 28:341–361, 1995.

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