Efficient rewriting in cograph trace monoids

Michael Bertol*
Universität Stuttgart, Institut für Informatik
Breitwiesenstr. 20-22, D-70565 Stuttgart
e-mail: bertol@informatik.uni-stuttgart.de

Abstract. We consider the basic problem of finding irreducible forms w.r.t. a finite noetherian rewriting system over a free partially commutative monoid where the underlying dependence alphabet is a cograph. A linear time algorithm is developed which determines irreducible normal forms w.r.t. finite, length-reducing trace rewriting systems over cograph monoids. This generalizes well-known results for free monoids and commutative monoids and is a significant improvement to the previously known square time algorithm.

1 Introduction

Free partially commutative monoids were introduced in combinatorics by Cartier and Foata [4]. In computer science these monoids are known as trace monoids, cf. Mazurkiewicz [7]. For background material we refer to [1, 8] or [5].

The theory of rewriting over trace monoids combines combinatorial aspects from string rewriting (modulo some commutations) and graph rewriting. The restriction to traces leads to feasible algorithms, but some interesting complexity questions are still open. One of the challenging open problems is to improve the known square time bound for the non-uniform complexity of length-reducing systems. It is well-known that we can compute irreducible descendants over free monoids in linear time [3, Thm. 4.1], if the rewriting system is length-reducing. Obviously, the linear time bound holds for free commutative monoids, too. Our aim is to generalize this bound to a larger class of free partially commutative monoids.

We give a linear time algorithm for determining irreducible forms w.r.t. length-reducing rewriting systems over a natural subclass of finitely generated free partially commutative monoids, the cograph monoids. Cograph monoids are trace monoids where the underlying dependence alphabet is a cograph, i.e. the smallest class of graphs containing singletons and which is closed under finite disjoint union and complementation. From an algebraic viewpoint cograph monoids are the smallest class of monoids containing the finitely generated free and free commutative monoids and which are closed under finite free and finite direct product.

* Partially supported by the ESPRIT Basic Research Action No. 6317 ASMICS II.

These monoids provide a natural model for asynchronous computation. Especially the cograph structure models the execution of “well structured” concurrency, e.g. Dijkstra’s construct “`par_begin...par_end`” and the usual sequential composition “`;`” in a programming language. Since there are many related problems, e.g. (parallel) programming languages that are based on rewriting, it is of interest to know efficient methods for computing irreducible forms.

In the first part of the paper we recall some background about traces, and we establish some facts about cograph monoids and decompositions. Then we develop a linear time algorithm which solves the considered problem.

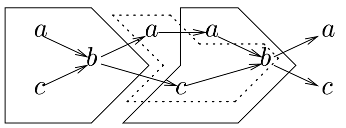
2 Notations

By Σ we denote a finite *alphabet*; a *dependence alphabet* is a pair (Σ, D), where $D \subseteq \Sigma \times \Sigma$ 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 the *free partially commutative monoid* generated by the dependence alphabet (Σ, D). An element $t \in \mathbb{M}$ is called a trace. We denote by $|t|$ the *length* of a trace t, by $|t|_a$ its *a-length*, for $a \in \Sigma$, 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 $\mathrm{alph}(u) \times \mathrm{alph}(v) \subseteq I$. We denote by $[k]$ the set $\{1, \ldots, k\}$.

A trace may be viewed as a partially ordered multiset (pomset). Let $w = a_1 \ldots a_n \in \Sigma^*$ be a word, $a_i \in \Sigma$, for all $i \in [n]$. Then the pomset of the corresponding trace $t = [w] \in \mathbb{M}$ is given by $(S_t, \sqsubset_t^*)$, where $S_t = \{a_1, \ldots, a_n\}$ is the multiset of letters occurring in w, and $\sqsubset_t^*$ is the partial order induced by $a_i \sqsubset_t a_j$ if both $i < j$ and $(a_i, a_j) \in D$. Interpreted as a directed (acyclic) graph, $(S_t, \sqsubset_t^*)$ is called the dependence graph of t. This provides a nice geometric interpretation.

A *subtrace* s of a trace 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., $t = usv$ (this factorization is not unique, in general). Conversely, if $t = usv$ then the traces u, s and v can be identified as subtraces of t.

Example 1. Let the dependence alphabet (Σ, D) be $(\{a,b,c\}, [a - b - c])$ and the trace $t = [acbacabca] = \{acbaacbac, acbaacbca, \ldots, cabcaabca\}$. In the graph of the trace[2] in the example we find two subgraphs that are subtraces of t representing the trace $l = [acb]$. The occurrences are shaped by polygons. The dotted polygon is a subgraph which is not a factor

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

[2] We are drawing (without loss of information for fixed dependence alphabets) the Hasse diagram, only. This is the reduced graph of the partial order.

3 Cograph monoids and decompositions

We restrict our investigations to a subclass of finitely generated free partially commutative monoids, the cograph monoids.

For two disjoint dependence alphabets $(\Sigma, D_\Sigma), (\Delta, D_\Delta)$, i.e., $\Sigma \cap \Delta = \emptyset$, we define the *direct sum* by

$$(\Sigma, D_\Sigma) \oplus (\Delta, D_\Delta) \ := \ (\Sigma \cup \Delta, D_\Sigma \cup D_\Delta),$$

and the *complex product* by

$$(\Sigma, D_\Sigma) * (\Delta, D_\Delta) \ := \ (\Sigma \cup \Delta, D_\Sigma \cup D_\Delta \cup (\Sigma \times \Delta) \cup (\Delta \times \Sigma)).$$

Definition 1. Let $\mathcal{D}$ be the smallest subclass of dependence alphabets containing all one-letter dependence alphabets $(\{a\}, \{(a,a)\})$ that is closed under taking finite direct sums and complex products. The trace monoids generated by dependence alphabets from $\mathcal{D}$ form the class of *cograph (trace) monoids*, denoted by $\mathcal{M}$.

There is a well-known graph theoretical characterization of cographs, see [10], [11] or [12]. They are precisely undirected (loop-less) graphs $(\Sigma, D - \{(a,a) \mid a \in \Sigma\})$ containing no induced subgraph isomorphic to $L_3 = [a - b - c - d]$.

A *description* for a dependence alphabet (Σ, D) is a well-formed linear term built from the constants $a \in \Sigma$ (representing the one-letter dependence alphabet $(\{a\}, \{(a,a)\})$) and the binary operations $\oplus$ and $*$. We identify descriptions modulo commutativity and associativity of $\oplus$ ($*$, resp.).

For a dependence alphabet (Σ, D) we call the set of connected components its *decomposition set* if the graph (Σ, D) is not connected. If (Σ, D) is connected we call the set of complements of the connected components of the graph $(\Sigma, \Sigma \times \Sigma - D)$ the *decomposition set* of (Σ, D). Note that the decomposition set of $(\Sigma, D) \in \mathcal{D}$ contains disjoint dependence alphabets.

Fact A. Let $T = T_1 \oplus \ldots \oplus T_n$ (or $T = T_i * \ldots * T_n$, resp.), for some $n \in \mathbb{N}$, a description of a dependence alphabet $(\Sigma, D) \in \mathcal{D}$ such that each T_i has no non-trivial $\oplus$-decomposition ($*$, resp.) then the set of dependence alphabets corresponding to the descriptions T_i, $i \in [n]$, is the decomposition set.

Fact B. Let $\mathbb{M}_\Sigma = \mathbb{M}(\Sigma, D_\Sigma)$ and $\mathbb{M}_\Delta = \mathbb{M}(\Delta, D_\Delta)$ be the free partially commutative monoids generated by the disjoint dependence alphabets (Σ, D_Σ) and (Δ, D_Δ). Then

$$\begin{aligned} \mathbb{M}((\Sigma, D_\Sigma) \oplus (\Delta, D_\Delta)) &\cong \mathbb{M}_\Sigma \times \mathbb{M}_\Delta \\ \mathbb{M}((\Sigma, D_\Sigma) * (\Delta, D_\Delta)) &\cong \mathbb{M}_\Sigma * \mathbb{M}_\Delta. \end{aligned}$$

Here $\mathbb{M}_\Sigma * \mathbb{M}_\Delta$ denotes the free product and $\mathbb{M}_\Sigma \times \mathbb{M}_\Delta$ denotes the direct product of monoids $\mathbb{M}_\Sigma$ and $\mathbb{M}_\Delta$.

Let $\mathcal{T}$ be the set of well-formed terms consisting of elements $t \in \mathbb{M}(\Sigma, D)$ and function symbols $*$ and $\times$ (of arbitrary arity). We define the following mapping inductively

$$h_{(\Sigma,D)} : \mathbb{M}(\Sigma, D) \longrightarrow \mathcal{T}$$

If (Σ, D) is not decomposable, i.e. the decomposition set contains only the element (Σ, D) then

$$h_{(\Sigma,D)} : x \longmapsto x$$

is the embedding. Otherwise, (Σ, D) is decomposable with decomposition set $\{(\Sigma_i, D_i) \mid i \in [k]\}$ $(k > 1)$, say.

(i) If the dependence alphabet (Σ, D) is the direct sum $\oplus_{i\in[k]}(\Sigma_i, D_i)$, we define

$$h_{(\Sigma,D)} : \mathbb{M}(\Sigma, D) \cong \prod_{i\in[k]} \mathbb{M}(\Sigma_i, D_i) \longrightarrow \mathcal{T}$$

$$(t_1, \ldots, t_k) \longmapsto \times(h_{(\Sigma_1,D_1)}(t_1), \ldots, h_{(\Sigma_k,D_k)}(t_h))$$

(ii) If (Σ, D) is the complex product $*_{i\in[k]}(\Sigma_i, D_i)$, we define

$$h_{(\Sigma,D)} : \mathbb{M}(\Sigma, D) \cong \mathop{*}_{i\in[k]} \mathbb{M}(\Sigma_i, D_i) \longrightarrow \mathcal{T}$$

$$t_1 \cdot \ldots \cdot t_n \longmapsto *(h_{(\Sigma_{i_1},D_{i_1})}(t_1), \ldots, h_{(\Sigma_{i_n},D_{i_n})}(t_n)),$$

where $t_1 \cdot \ldots \cdot t_n$ denotes the usual free product, i.e., $\{i_1, \ldots, i_n\} \subseteq [k]$ and $i_j \neq i_{j+1}$, for all $j \in [n-1]$, and $t_j \in \mathbb{M}(\Sigma_{i_j}, D_{i_j})$, $j \in [n]$.

Below we write h for $h_{(\Sigma,D)}$.

Remark 2. For cograph dependence alphabets $(\Sigma, D) \in \mathcal{D}$ the images $h(t)$, $t \in \mathbb{M}(\Sigma, D)$, are well-formed terms consisting of words a^n, $n \geq 0$, and the function symbols $*$ and $\times$.

Definition 3. For a (cograph) monoid $\mathbb{M}(\Sigma, D)$ the *trace term* associated with $t \in \mathbb{M}(\Sigma, D)$ is the image $h(t) \in \mathcal{T}$ by the mapping h.

We call non-decomposable dependence alphabets (Σ, D) *nuclear* (i.e. the decomposition set contains one element).

Let $T(\Sigma, D)$ be the h-image $h_{(\Sigma,D)}(\mathbb{M}(\Sigma, D)) \subset \mathcal{T}$. Then the subset $T(\Sigma, D)$ of well-formed terms forms a monoid, where 1 is the empty term and the concatenation $s \circ t$ of two trace terms s and t is defined in the following way.
if $s = \times(s_1, \ldots, s_k)$ and $t = \times(t_1, \ldots, t_k)$ then $s \circ t = \times(s_1 \circ t_1, \ldots, s_n \circ t_n)$
if $s = *(s_1, \ldots, s_m)$ and $t = *(t_1, \ldots, t_n)$ then

$$s \circ t = *(s_1, \ldots s_{m-1}, s_m \circ t_1, t_2, \ldots, t_n), \text{ when } s_m, t_1 \in T(\Sigma_i, D_i), \text{ resp.}$$

$$s \circ t = *(s_1, \ldots, s_m, t_1, \ldots, t_n), \text{ when } s_m \in T(\Sigma_i, D_i) \neq T(\Sigma_j, D_j) \ni t_1$$

Otherwise $s \circ t = st$ (when $s, t \in \mathbb{M}(\Sigma, D)$, and (Σ, D) is nuclear).

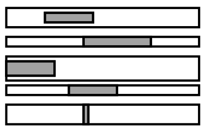
Proposition 4. *The mapping* $h : \mathbb{M}(\Sigma, D) \longrightarrow T(\Sigma, D)$ *is injective and a monoid morphism.*

The structure of these objects allows to locate factors of a trace. The following lemma provides a characterization of the factor property.

Lemma 5. *Let* $\{(\Sigma_i, D_i) \mid i \in [k]\}$ *be the decomposition set of the dependence alphabet* (Σ, D)*,* $\mathbb{M}_i = \mathbb{M}(\Sigma_i, D_i)$*, for all* $i \in [k]$*, and let* $t, l \in \mathbb{M}(\Sigma, D)$*.*

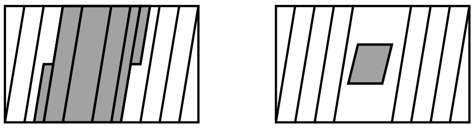
(i) If $t, l \in \prod_{i\in[k]} \mathbb{M}_i$ *(direct product) then* $t = ulv$*, for some* u, v*, iff*
- $\forall i \in [k] : \pi_i(t) = \pi_i(u)\pi_i(l)\pi_i(v)$,

where π_i *is the projection monoid morphism* $\mathbb{M}(\Sigma, D) \longrightarrow \mathbb{M}_i$*.*



(ii) If $t, l \in *_{i\in[k]}\mathbb{M}_i$ (free product) with $t = t_1 \cdot \ldots \cdot t_n$ and $l = l_1 \cdot \ldots \cdot l_m$ then $t = ulv$, for some u, v, iff either

- *$\exists\, j \in \{0\} \cup [n] \;:\; t_{j+2} = l_2 \;\wedge \ldots \wedge\; t_{j+m-1} = l_{m-1}$, and $\exists\, x, y \;:\; xl_1 = t_{j+1} \;\wedge\; l_m y = t_{j+m}$, or*
- *$\exists i \in [k] \;:\; l \in \mathbb{M}_i$ with $t_j = xly$ for some $x, y \in \mathbb{M}_i$ and $j \in [n]$.*



The grey rectangles depict the factor l, the diagonal lines free products, the horizontal lines direct products, and the envelopping square is the shape of the dependence graph of the trace t.

FACT C. Deciding whether two traces t and l are equal, l is a prefix of t, and l is a suffix of t is possible in $\mathcal{O}(\min(|t|, |l|))$ time.

4 Trace rewriting systems

In this section we introduce trace rewriting systems.

Definition 6. A trace-rewriting system is a finite set of rewriting rules $\mathcal{R} \subseteq \mathbb{M}(\Sigma, D) \times \mathbb{M}(\Sigma, D)$.

A trace-rewriting system $\mathcal{R}$ defines a *derivation relation* $\Longrightarrow_{\mathcal{R}}$ by $x \Longrightarrow_{\mathcal{R}} y$ if $x = ulv, y = urv$ for some $u, v \in \mathbb{M}$ and $l \to r \in \mathcal{R}$. By $\overset{*}{\Longrightarrow}_{\mathcal{R}}$ we denote the reflexive transitive closure of $\Longrightarrow_{\mathcal{R}}$. A trace-rewriting system $\mathcal{R}$ is called *length-reducing*, if $|l| > |r|$ for all $l \to r \in \mathcal{R}$. The set of *irreducible traces* is given by $\mathrm{Irr}(\mathcal{R}) = \{t \in \mathbb{M} \mid \neg\exists s : t \Longrightarrow_{\mathcal{R}} s\}$.

The following proposition is well-known, see e.g. [6, p.138, Thm. 5.7.1]. 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 rewriting step is then to define $v := v'rv''$. By iterating these two steps we obtain a correct algorithm, but its worst-case behavior is square time. No better time bound is known, in general.

Proposition 7. *Let $\mathcal{R} \subseteq \mathbb{M} \times \mathbb{M}$ be a finite and length-reducing trace rewriting system. Then irreducible forms w.r.t. $\mathcal{R}$ can be computed in square time.*

5 The rewriting algorithm for cographs

In this section we show our main result which is an improvement of Proposition 7. In the following let $\mathcal{R} \subseteq \mathbb{M} \times \mathbb{M}$ be a finite and length-reducing trace rewriting system such that each right hand side is irreducible. By abuse of language we identify the various representations of a trace.

Let us denote the set of prefixes (suffixes resp.) of a trace $t \in \mathbb{M}$ by $\mathit{Pref}(t) := \{\alpha \in \mathbb{M} \mid \alpha\beta = t\}$ ($\mathit{Suff}(t) := \{\beta \in \mathbb{M} \mid \alpha\beta = t\}$, resp.). For a set $S \subseteq \mathbb{M}$ we define $\mathit{Pref}(S) := \bigcup_{s \in S} \mathit{Pref}(s)$ and $\mathit{Suff}(S) := \bigcup_{s \in S} \mathit{Suff}(s)$.

Let $L_{\mathcal{R}}$ denote the set of left hand sides $\{l \mid (l,r) \in \mathcal{R}\}$ of a rewriting system $\mathcal{R} \subseteq \mathbb{M} \times \mathbb{M}$. We denote by $Dec(l) = Dec(h(l))$ the *decomposition of the trace* $l \in L_{\mathcal{R}}$, with $Dec(\times(l_1,\ldots,l_k)) := \bigcup_{i\in[k]} Dec(l_i)$, and $Dec(*(l_1,\ldots,l_n)) := \{l_1 \cdot \ldots \cdot l_n\}$, $n > 1$; $Dec(*(l_1)) := Dec(l_1)$ (is considered as a direct product). Otherwise, for nuclear monoids, $Dec(l) = \{l\}$. Further we define

$$Dec(\mathcal{R}) := \bigcup_{l \in L_{\mathcal{R}}} Dec(l), \quad Pref(\mathcal{R}) := Pref(L_{\mathcal{R}}), \quad Suff(\mathcal{R}) := Suff(L_{\mathcal{R}}).$$

We describe now a method for determining irreducible forms by performing iteratively derivation steps $t = ulv \Longrightarrow_{\mathcal{R}} urv = t'$. Each trace is represented by a tree corresponding to its trace term. Each subterm $h(s)$ corresponding to the trace s (represented by its subtree) is additionally labeled by the sets

$$P(s) \ = \ Suff(s) \cap Pref(\mathcal{R}) \quad \text{ and } \quad S(s) \ = \ Pref(s) \cap Suff(\mathcal{R}).$$

Further, each subterm s is labeled by $L(s)$ where,

(i) if $s = *(s_1,\ldots,s_n)$ then $L(s) = (L_d(s))_{d\in \mathrm{Dec}(\mathcal{R})}$, with $L_d(s)$ is an ordered set of integers $\{i_1,\ldots,i_{n_d}\} \subseteq [n]$ such that $i_j \in L_d(s)$ iff either

$$udv = ud_1 \cdot \ldots \cdot d_k v = s_{i_j} \cdot \ldots \cdot s_{i_j+k-1}, \ k > 1,$$

for some $u, v \in \mathbb{M}$, $j \in [n_d]$, or $i_j \in L_d(s)$ if $d \in L(s_{i_j})$.

(ii) If $s = \times(s_1,\ldots,s_k)$ then the term is labeled by the set of traces $L(s) = \bigcup_{i\in[k]}\{d \in Dec(\mathcal{R}) \mid L_d(s_i) \neq \emptyset\}$.

(iii) If $s = a^n$, $n > 0$ then $L_d(s) = \emptyset$, for $d \notin a^* \cap Dec(\mathcal{R})$, resp. $L_d(s) = \{j \mid 0 < j \leq n - |d|\}$, for $d \in a^* \cap Dec(\mathcal{R})$.

Applying Fact C iteratively for determining the labels $P(s)$ and $S(s)$ of each subterm s of t we can use this information to determine the L-labeling bottom-up by propagating the information on the path to the root of the tree representing the term. Hence this data structure can be constructed in linear time.

We introduce the following functions and give an implementation of two of them. (Each of the functions maintains the labeling correctly.)

(i) SPLIT$(i, *(t_1,\ldots,t_n))$ — split the term $*(t_1,\ldots,t_n)$ into the two terms $*(t_1,\ldots,t_{i-1})$ and $*(t_i,\ldots,t_n)$;

(ii) PROJECT$(i, \times(t_1,\ldots,t_k))$ — output the i-th projection t_i of the trace $\times(t_1,\ldots,t_k)$;

(iii) COMPOSE$(t_1,\ldots,t_k)$ — construct the trace term $\times(t_1,\ldots,t_k)$;

(iv) CONCAT(t_1,t_2) — concatenate two trace terms t_1 and t_2 such that the result represents $t_1 \circ t_2$.

Because of space limitations we only show two of these methods where some trivial cases are omitted.

FUNCTION SPLIT$(i, *(t_1,\ldots,t_i,\ldots,t_n))$
  if $(1 < i < n)$ **then** $\langle t_1', t_2' \rangle \leftarrow \langle *(t_1,\ldots,t_{i-1}), *(t_i,\cdot\ldots\cdot,t_n)\rangle$
    $P(t_1') \leftarrow P(t_{i-1}) \cup \{l_1 \cdot \ldots \cdot l_j \in Pref(\mathcal{R}) \mid t_1 \cdot \ldots \cdot t_{i-1} = \alpha l_1 \cdot \ldots \cdot l_j, j > 1\}$
    $P(t_2') \leftarrow P(t_n) \cup \{l_1 \cdot \ldots \cdot l_j \in Pref(\mathcal{R}) \mid t_i \cdot \ldots \cdot t_n = \alpha l_1 \cdot \ldots \cdot l_j, j > 1\}$
    $S(t_1') \leftarrow S(t_1) \cup \{l_1 \cdot \ldots \cdot l_j \in Suff(\mathcal{R}) \mid t_1 \cdot \ldots \cdot t_{i-1} = l_1 \cdot \ldots \cdot l_j\beta, j > 1\}$
    $S(t_2') \leftarrow S(t_i) \cup \{l_1 \cdot \ldots \cdot l_j \in Suff(\mathcal{R}) \mid t_i \cdot \ldots \cdot t_n = l_1 \cdot \ldots \cdot l_j\beta, j > 1\}$

forall $d \in Dec(\mathcal{R})$ **do**
$L_d(t_1') \leftarrow (L_d(t) \cap [i-1]) - \{i-j \mid d_1 = d_1' \cdot \ldots \cdot d_j' \in P(t_{i-1}),$
$d_2 \in S(t_i), d_1 d_2 \in Dec(\mathcal{R})\}$
$L_d(t_2') \leftarrow L_d(t) \cap \{i, \ldots, n\}$
return $(\langle t_1', t_2' \rangle$ with labeling S, P, L, resp.)
else if $i = 1$ **then return**$(\langle 1, *(t_1, \cdot \ldots \cdot, t_n) \rangle)$
else if $i = n$ **then return**$(\langle *(t_1, \cdot \ldots \cdot, t_n), 1 \rangle)$
otherwise error

The function CONCAT uses the previous functions. Note that each operation maps consistent data structure to consistent structure.

FUNCTION CONCAT(t_1, t_2)
if $t_1 = \times(t_1', \ldots, t_k')$ and $t_2 = \times(t_1'', \ldots, t_k'')$
then forall $(1 \le i \le k)$ **do**
$\langle t_i^1, t_i^2 \rangle \leftarrow \langle$ PROJECT(i, t_1), PROJECT$(i, t_2) \rangle$
$t_i''' \leftarrow$CONCAT(t_i^1, t_i^2)
return (COMPOSE$(t_1''', \ldots, t_k''')$)
if $t_1 = *(t_1', \ldots, t_{n_1}')$ and $t_2 = *(t_1'', \ldots, t_{n_2}'')$ **then**
if $t_{n_1}' \in \mathbb{M}_i \neq \mathbb{M}_j \ni t_1''$, where $\mathbb{M} = *_{i \in [k]} \mathbb{M}_i$
then $s \leftarrow *(t_1', \ldots, t_{n_1}', t_1'', \ldots, t_{n_2}'')$
else $\langle s_1', s_2' \rangle \leftarrow$SPLIT$(n_1, t_1)$ $\langle s_1'', s_2'' \rangle \leftarrow$SPLIT$(2, t_2)$
$s \leftarrow$CONCAT$(s_1'$,CONCAT(CONCAT$(s_2', s_1''), s_2''))$
let $s = *(s_1, \ldots, s_n)$
$P(s) \leftarrow P(s_n) \cup \{l_1 \cdot \ldots \cdot l_j \in Pref(\mathcal{R}) \mid s_1 \cdot \ldots \cdot s_n = \alpha l_1 \cdot \ldots \cdot l_j, j > 1\}$
$S(s) \leftarrow S(s_1) \cup \{l_1 \cdot \ldots \cdot l_j \in Suff(\mathcal{R}) \mid s_1 \cdot \ldots \cdot s_n = l_1 \cdot \ldots \cdot l_j \beta, j > 1\}$
forall $d \in Dec(\mathcal{R})$ **do**
if there exists $i \in [k]$ with $L_d(s_i) \neq \emptyset$ **then** $L(s) \leftarrow L(s) \cup \{d\}$
$L(s) \leftarrow L(s) \cup \{d_1 d_2 \in Dec(\mathcal{R}) \mid d_1 \in P(t_1), d_2 \in S(t_2)\}$
return s with labeling P, S, L
else ... trivial (base) cases ...

Further each of the four operations has constant time complexity, because the relabeling is (partially) done by inheriting the labeling information and the remaining sets can be calculated locally, hence in constant time (use Fact C). Recall the monoid description and therefore the depth $d(h(t))$ of the term corresponding to the trace t is bounded by the cardinality $|\Sigma|$. Here the depth $d(w) = 1$ for a term $w \in a^*$, and $d(*(t_1, \ldots, t_n)) = d(\times(t_1, \ldots, t_n)) = 1 + \max_{i \in [n]} d(t_i)$.

Suppose we are given three traces $u, v, t \in \mathbb{M}$ with $t = uv$. Then the *right quotient* tv^{-1} is u, and the *left quotient* $u^{-1}t$ is v, respectively.

With the operations presented above we are able to describe a method for calculating a right quotient and left quotient respectively, that is running in linear time and maintains the described data structure. Since we are interested in rewriting of left hand sides it is sufficient considering only quotients tp^{-1} for $p \in Pref(\mathcal{R})$, and $s^{-1}t$ for $s \in Suff(\mathcal{R})$. Again, these procedures run in constant time.

FUNCTION rightQUOTIENT(t, l)
if $l \notin P(t)$ **then error**

else if $t = \times(t_1, \ldots, t_k)$ and $l = \times(l_1, \ldots, l_k)$
then for all $(1 \leq i \leq k)$ **do**
$s_i \leftarrow$rightQUOTIENT(PROJECT(i,t),PROJECT(i,l))
return (COMPOSE$(s_1, \ldots, s_k)$)
else if $t = *(t_1, \ldots, t_n)$ and $l = *(l_1, \ldots, l_m)$
then if $m = 1$ **then**
$\langle s_1, s_2 \rangle \leftarrow$SPLIT$(n,t)$
return CONCAT(s_1,rightQUOTIENT(s_2, l_1))
else $\langle s, t' \rangle \leftarrow$ SPLIT$(n - m + 1, t)$
$\langle l', l'' \rangle \leftarrow$ SPLIT$(2, l)$
return (rightQUOTIENT(s, l'))
else ... trivial (base) cases ...

Note that all functions return on consistent data consistent results, i.e. the interpretation of the resulting term(s) as trace reflects the semantic of the used transformation and the labeling corresponds to the definition of S, P and L.

In the following we may assume that the rewriting system is reduced. In particular, we may assume that all right-hand sides are irreducible. Now we are able to describe an algorithm solving the considered non-uniform normal form problem.

In the method described below, we distinguish between three cases: the base case, when the underlying monoid is nuclear, and the two cases when the monoid is a direct or a free product. In the free product case and the base case we proceed like in Book's algorithm, overlapping factors that occur by concatenation are eliminated. In the direct product case there might arise combinations of factors, i.e., in each projection t_i might occur a factor $d_i \in Dec(\mathcal{R})$, such that $(d_1, \ldots, d_k)$ is a left hand side. Therefore we introduced the labeling L. To reference these factors we use another procedure called PART. PART(t,d) returns for a decomposition $d \in Dec(\mathcal{R})$ a factorization $t = uv$ such that for $d = d_1 \cdot \ldots \cdot d_m$: $u = \tilde{u} d_1$ and $v = d_2 \cdots d_n \tilde{v}$, with $|u|$ minimal.

PROCEDURE PART(t,d)
if $t = *(t_1, \ldots, t_n)$ and $L_d(t) \neq \emptyset$ **then**
let $j \leftarrow \min L_d(t)$
$\langle t', t'' \rangle \leftarrow$SPLIT$(t, j+1)$
if $d = *(d_1, \ldots d_m)$ $(m > 1)$ **then return**$\langle t', t'' \rangle$
if $d = *(d_1)$ **then** $\langle u, v \rangle \leftarrow$SPLIT$(t', j)$)
$\langle x, y \rangle \leftarrow$PART$(v, d)$
return($\langle$CONCAT(u,x),CONCAT$(y,t'')\rangle$)
else if $t = \times(t_1, \ldots, t_k)$ and $d \in L(t)$ **then**
let $j \in [k]$ with $L_d(t_j) \neq \emptyset$
forall $(1 \leq i \leq k)$ **do** $t_i \leftarrow$PROJECT(t,i)
$\langle u, v \rangle \leftarrow$PART$(t_j, d)$
return($\langle$COMPOSE$(1, \ldots, u, \ldots, 1)$,COMPOSE$(t_1, \ldots, v, \ldots, t_k)\rangle$
else if $t, d \in a^*$ and $|t| \geq |d|$ **then return**($\langle a, a^{|t|-1}\rangle$)
otherwise error

Using the labeling L extensively this procedure works in constant time and returns consistent data structures (by the same arguments proving the correctness and complexity of e.g. SPLIT).

```
PROCEDURE CONCATandREDUCE(s, t)
  if s = *(s_1, ..., s_m) and t = *(t_1, ..., t_n) then
    if exists l = l_1 l_2 → r ∈ R with l_1 ∈ S(s_m) and l_2 ∈ P(t_1) then
      s' ←rightQUOTIENT(s, l_1); t' ←leftQUOTIENT(t, l_2)
      return(CONCATandREDUCE(CONCATandREDUCE(s', r), t'))
    else return(CONCAT(s, t))
  else if s = ×(s_1, ..., s_k) and t = ×(t_1, ..., t_k) then
    v ←CONCAT(s, t)
    for all (1 ≤ i ≤ k) do u_i ← 1; v_i ←PROJECT(i, v)
    while there exists l = (l_1, ..., l_k) ∈ L_R such that l_j ∈ L(s) ∪ L(t),
          for all j ∈ [k], and l → r ∈ R,
          let l be chosen such that x is irreducible in v = xly do
      forall (1 ≤ i ≤ k) do ⟨v'_i, v''_i⟩ ←PART(v_i, l_i)
                            let l_i = l_i^1 l_i^2 with l_i^1 ∈ P(v'_i) and l_i^2 ∈ S(v''_i)
                            u_i ←CONCAT(u_i,rightQUOTIENT(v'_i, l_i^1))
                            v_i ←leftQUOTIENT(l_i^2, v''_i)
      u ←CONCATandREDUCE(COMPOSE(u_1, ..., u_k), r)
      v ←COMPOSE(v_1, ..., v_k)
    end while
    return CONCAT(u, v)
  else if st = a^n and there exists a^m → r ∈ R and m ≤ n then
    return(CONCATandREDUCE(r, a^(n-m)))
  otherwise return CONCAT(s, t)
```

Suppose procedure CONCATandREDUCE(s,t) is called on a consistent data structure, i.e. s and t are representations of trace terms with correct labelings, then the procedure has the following invariants.

(i) When leaving the procedure CONCATandREDUCE the data structure is consistent.
(ii) CONCATandREDUCE(s,t) determines a descendant $\hat{t}$ of the trace st, i.e. $st \stackrel{*}{\Longrightarrow}_{\mathcal{R}} \hat{t}$. Moreover, if s and t are irreducible then $\hat{t}$ is irreducible, too.

Theorem 8. *Let $\mathcal{R} \in \mathbb{M}^2$ be a length-reducing rewriting system over some co-graph monoid $\mathbb{M}$. Further let $t \in \mathbb{M}$ be a trace. Determining an irreducible form of t is solvable in linear time.*

Proof. Without loss of generality $\mathcal{R}$ is normalized, i.e., each right hand side is irreducible and no single letter. Let $t = [a_1 \dots a_n] \in \mathbb{M}(\Sigma, D)$ be the input trace. We call the routine by "**for** $i = 1$ **to** n **do** $v \leftarrow$CONCATandREDUCE(v, a_i)" with $v = 1$ initially, and return the last value of v. Let $\Psi : \mathbb{M}(\Sigma, D) \longrightarrow \mathbb{N}$ be the linear potential function defined by

$$\Psi(t) \quad = \quad c \cdot |h(t)| + \sum_{t' \text{ is subterm of } h(t)} |P(t')| + |S(t')| + |L(t')|,$$

where $|h(t)|$ is the number of vertices. There exists a constant $c > 0$ (depending on (Σ, D) and $\mathcal{R}$), such that $\Psi(v[a_i \dots a_n])$ is strictly decreasing while performing reduction steps (i.e. doing constant many of the primitive operations for performing one derivation step). Since $\Psi(t) \geq 0$ and $\psi(t) \in \mathcal{O}(|t|)$ we get the result. $\square$

6 Conclusion and further work

We have established a method for solving the non-uniform normal form problem for finite length-reducing trace rewriting systems over cograph monoids working in linear time. We conjecture that it is possible to solve this problem for arbitrary free partially commutative monoids in $\mathcal{O}(n \log n)$ where n denotes the input size, but we doubt that there exists a linear time algorithm, in general.

Acknowledgment: Discussions with Dan Teodosiu motivated me to investigate these structures. I would like to thank Volker Diekert, Anca Muscholl and Zoltán Ésik as well as the anonymous referees for comments and suggestions.

References

1. I.J. Aalbersberg and G. Rozenberg. Theory of traces. *Theoretical Computer Science*, 60:1–82, 1988.
2. A.V. Aho *Algorithms for finding patterns in strings* Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, Elsevier 1990
3. R. Book. *Confluent and other types of Thue systems. J. Assoc. Comput. Mach.*, 29:171–182, 1982.
4. P. Cartier and D. Foata. *Problèmes combinatoires de commutation et réarrangements. Lecture Notes in Mathematics 85.* Springer, 1969.
5. V. Diekert and G. Rozenberg. *The Book of Traces. World Scientific*, 1995.
6. V. Diekert. *Combinatorics on Traces.* Number 454 in Lecture Notes in Computer Science. Springer, 1990.
7. A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.
8. 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, Springer, 1987.
9. Vaughan Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Processing*, 15(1):33–71, 1986.
10. Jan Grabowski. On partial languages. *Fundamenta Informatica*, IV(2):427–498, 1981.
11. J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series-parallel digraphs. *SIAM Journal of Computing*, 11(2):298–313, 1981.
12. E.S. Wolk. A note on "The comparabitily graph of a tree". *Amer. Math. Soc.*, 16:17–22, 1965.