COMPUTING
SCIENCE

Lexicographical Generations of Combined Traces

Lukasz Mikulski, Marcin Piatkowski and Sebastian Smyczynski

TECHNICAL REPORT SERIES

No. CS-TR-1365 January 2013

TECHNICAL REPORT SERIES

No. CS-TR-1365 January, 2013

Lexicographical Generations of Combined Traces

L. Mikulski, M. Piatkowski and S. Smyczynski

Abstract

Combined traces are intrinsic mathematical model for studying concurrent systems behaviours. They can be used to describe and investigate processes of elementary net systems with inhibitor arcs and allow to describe weak causality and simultaneity of actions. We provide several algorithms for manipulating combined traces using their language theoretic representations. In particular, we propose two methods of enumeration related to combined traces, supported by a collection of auxiliary procedures. First, for a specified combined trace we iterate the set of all its representatives (namely step sequences). Next, we use the lexicographical order on step sequences to list all combined traces of a fixed size. We discuss the time complexity of all presented algorithms.

© 2013 Newcastle University.
Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

MIKULSKI, L., PIATKOWSKI, M., SMYCZYNSKI, S.

Lexicographical Generations of Combined Traces
[By] L. Mikulski, M. Piatkowski, S. Smyczynski

Newcastle upon Tyne: Newcastle University: Computing Science, 2013.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1365)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1365

Abstract

Combined traces are intrinsic mathematical model for studying concurrent systems behaviours. They can be used to describe and investigate processes of elementary net systems with inhibitor arcs and allow to describe weak causality and simultaneity of actions. We provide several algorithms for manipulating combined traces using their language theoretic representations. In particular, we propose two methods of enumeration related to combined traces, supported by a collection of auxiliary procedures. First, for a specified combined trace we iterate the set of all its representatives (namely step sequences). Next, we use the lexicographical order on step sequences to list all combined traces of a fixed size. We discuss the time complexity of all presented algorithms.

About the authors

Lukasz Mikulski obtained his PhD in 2012 from the University of Warsaw, Poland. He now works as a Lecturer at the Faculty of Mathematics and Computer Science of the Nicolaus Copernicus University in Torun, Poland and is a Visiting Researcher in the School of Computing Science, Newcastle University, U.K.

Marcin Piatkowski obtained his PhD in 2011 from the University of Warsaw, Poland. He now works as a Lecturer at the Faculty of Mathematics and Computer Science of the Nicolaus Copernicus University in Torun, Poland.

Sebastian Smyczynski obtained his MSc in 2002 from the Nicolaus Copernicus University of Torun, Poland. He is the founder and currently the CEO of Simplito LLC in Torun, Poland.

Suggested keywords

COMBINED TRACES
CONCURRENT SYSTEMS
GENERATIONS
CAUSALITY
INDEPENDENCE
PARTIAL ORDER
LEXICOGRAPHICAL ORDER

Lexicographical Generations of Combined Traces

Łukasz Mikulski[1,2]* and Marcin Piątkowski[1] and Sebastian Smyczyński[1]

[1] Faculty of Mathematics and Computer Science
Nicolaus Copernicus University
Toruń, Chopina 12/18, Poland

[2] Newcastle University
School of Computing Science
Newcastle Upon Tyne, NE1 7RU, U.K

{lukasz.mikulski, marcin.piatkowski, sebastian.smyczynski}@mat.umk.pl

Abstract. Combined traces are intrinsic mathematical model for studying concurrent systems behaviors. They can be used to describe and investigate processes of elementary net systems with inhibitor arcs and allow to describe weak causality and simultaneity of actions. We provide several algorithms for manipulating combined traces using their language theoretic representations. In particular, we propose two methods of enumeration related to combined traces, supported by a collection of auxiliary procedures. First, for a specified combined trace we iterate the set of all its representatives (namely step sequences). Next, we use the lexicographical order on step sequences to list all combined traces of a fixed size. We discuss the time complexity of all presented algorithms.

Introduction

The quickly increasing number of devices equipped with multicore computing units gives the strong motivation for the reinvestigation of formal models of concurrent systems. In particular, the widely used graph-based model of Petri nets (see [9, 10]) and its formal language semantics described by traces (see [6]), are surely worth to consider. Moreover, despite of its importance, an algorithmic part of the trace theory seems to be undervalued.

In this paper we provide several algorithms for manipulating the natural extension of traces, which allows to describe and study not only causality and concurrency, but also weak causality and simultaneity. To adress this issue we recall the notion of *combined traces* (*comtraces* in short), see [3].

Consider for example the elementary net system with inhibitor arcs presented on Figure 1. The depicted system consists of two parts: upper (actions a, b, c, d) and lower (actions e, f, x). Some of possible processes modeled by this net lead to the execution of the actions e and f. One of such processes starts with the execution (step sequence in case of comtraces) of the whole upper part.

* This research was supported by a fellowship funded by the "Enhancing Educational Potential of Nicolaus Copernicus University in the Disciplines of Mathematical and Natural Sciences" Project POKL.04.01.01-00-081/10.

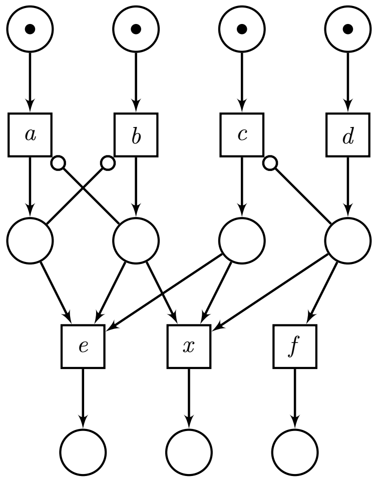


Fig. 1. Elementary net system

Due to existence of inhibitor arcs, the actions a and b have to be executed together. Such phenomena is called *indivisibility* of the step (ab) (see [8]). On the other hand, the actions c and d could be executed together as one step (cd) or serialised, but in one way only. Namely, action c has to be executed before or together with (i.e. "not later than") action d. Such behaviour (being neither full causality nor unordering) of actions is called *weak causality*. As a second part, we can execute the actions e and f together or in any order.

In some applications of concurrent systems, like hardware modelling, we are interested in conflict-free executions only. By conflict we mean a situation, where two actions are simultaneously enabled, however executing one of them disables the other. The main disadvantage of the above mentioned execution of discussed process is conflict that arises between the actions e and f, and the action x. To avoid such conflict, we can mix the execution of the upper and lower parts, remaining inside the same process. Namely, we can execute it in three steps $(abc)(de)(f)$ instead of doing it in two steps $(abcd)(ef)$.

One can see that actions c and f cannot be executed simultaneously, which means that they are dependent. Another solution for avoiding the conflict situation is to change the order of actions c and f. Executing f before c is possible and both leads to another, non equivalent process. One of possible realisations of this process is $(abd)(f)(c)(e)$. In no realisation of this second process, the problem with conflict situation appears.

To address this issue one can simply enumerate all possible executions of the considered process. In this paper we describe an algorithm that generates (in the fixed order) all step sequences contained in a given comtrace. To complete the algorithmic framework, we provide the method of generating all comtraces of the fixed size. It can be utilised to search for another processes of a given system that realises the same goal.

The paper is organised as follows. We start with the presentation of some basic notions and terminology. Next, we describe the concepts of normal forms and a projection representation of combined traces. The part concerning an algorithmic framework starts with the description of several auxiliary procedures. Further, there are presented algorithms for enumeration all step sequences contained in a given comtrace and for generation of all comtraces of the fixed size.

1 Preliminaries

Throughout the paper we use the standard notions of the formal language theory. In particular, by an *alphabet* we mean a nonempty finite set Σ, the elements of which are called *(atomic) actions*. Finite sequences of actions over Σ are called *words*. The set of all finite words over Σ, including the empty word ε, is denoted by Σ^*.

Let $\cdot$ denote the words concatenation operator (usually omitted). Since the concatenation operator is associative, the triple $(\Sigma^*, \cdot, \varepsilon)$ is a monoid.

Let $w = a_1 \dots a_n$ be a word. We use the standard notions of a prefix and a suffix of the word w. Moreover, for any $k \leq n$ the *k-suffix* of w, denoted by suff_k, is a word $a_k \dots a_n$. The *k-prefix* of w, denoted by pref_k, is a word $a_1 \dots a_k$.

We assume that the alphabet Σ is given together with a total order $\leq$, called the lexicographical order. We extend it in the natural way to the case of words over Σ. Moreover, we define the order $\widehat{\leq}$ on subsets of Σ as follows. If the sets A and B are of equal size we compare minimal elements of $A \setminus B$ and $B \setminus A$. Otherwise, the larger set is greater.

$$A \widehat{\leq} B \iff \begin{cases} |A| < |B| \\ \text{or} \\ |A| = |B| \;\text{ and }\; (A = B \text{ or } \min(A \setminus B) \leq \min(B \setminus A)) \end{cases} \tag{1}$$

This way $(2^\Sigma, \widehat{\leq})$ is a totally ordered set.
The projection onto binary subalphabet $\{a, b\}$ is the function $\Pi_{a,b} : \Sigma^* \to \Sigma^*$ defined as follows:

$$\Pi_{a,b}(cw) = \begin{cases} c\Pi_{a,b}(w) & \text{for } \; c \in \{a, b\} \\ \Pi_{a,b}(w) & \text{for } \; c \notin \{a, b\} \end{cases}$$

and $\Pi_{a,b}(\varepsilon) = \varepsilon$. In the same way we define projection onto a unary subalphabet $\{a\}$, denoted by $\Pi_{a,a} : \Sigma^* \to \Sigma^*$.

The algebra of binary relations over set X (i.e., a subsets of $X \times X$) is equipped with a concatenation operation $\circ$, where $R_1 \circ R_2 = \{(x, y) | \exists_{z \in X}\; xR_1z \wedge zR_2y\}$. The neutral element for $\circ$ is identity relation $I_X = \{(x, x) \mid x \in X\}$, the index X is omitted if is clear from context. The n-th power of relation R is defined as $R^n = R^{n-1} \circ R$ for all $n \geq 1$, where $R^0 = I$. The transitive closure of R is $R^+ = R^1 \cup R^2 \cup \dots$, while its reflexive transitive closure is $R^* = R^0 \cup R^+$. Moreover, for relation $R \subset X \times X$ we define its reverse $R^{-1} = \{(x, y) | (y, x) \in R\}$, and its symmetric closure $R^{sym} = R \cup R^{-1}$. The restriction of the relation R to the set $Y \subseteq X$ is the relation $R|_Y = \{(x, y) \mid x, y \in Y \wedge xRy\}$.

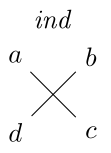
The relation $R \subseteq X \times X$ is called symmetric if $R = R^{-1}$, reflexive if $I \subseteq R$, irreflexive if $I \cap R = \varnothing$, transitive if $R^2 \subseteq R$, and acyclic if R^+ is irreflexive.

1.1 Traces

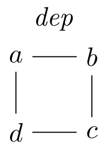
The *concurrent alphabet* is a pair $\Psi = (\Sigma, ind)$, where Σ is an alphabet and $ind \subseteq \Sigma \times \Sigma$ is an arbitrary irreflexive and symmetric relation, called *independence relation*. With independence we associate, as another relation, a *dependence relation* $dep = \Sigma \times \Sigma \setminus ind$. Having the concurrent alphabet, we define a relation that identifies similar words. We say that word $\sigma \in \Sigma^*$ is in relation $\equiv_\Psi$ with word $\tau \in \Sigma^*$ if there exists a finite sequence of commutations of subsequent, independent actions that leads from σ to τ. Relation $\equiv_\Psi \subseteq \Sigma^* \times \Sigma^*$ is a congruence relation (whenever it is not confusing, relation symbol Ψ will be omitted).

After dividing set Σ^* by the relation $\equiv_\Psi$ we get a quotient monoid. The elements of $\Sigma^*/_{\equiv_\Psi}$ are called *traces* (see [1, 6, 7]). This way, every word σ is related to a trace $\alpha = [\sigma]$, containing this word.

Example 1. To the alphabet $\Sigma = \{a, b, c, d\}$ we add an independence relation



or, equivalent,
a dependence relation



In this case words $abb\boldsymbol{aac}d$ and $abb\boldsymbol{caa}d$ are equivalent.

1.2 Combined traces

A *comtrace alphabet* is a triple $\Theta = (\Sigma, sim, ser)$, where Σ is an alphabet and $ser \subseteq sim \subseteq \Sigma \times \Sigma$ are two relations, respectively called *serialisability* and *simultaneity*; it is assumed that sim is irreflexive and symmetric. Intuitively, if $(a, b) \in sim$ then a and b may occur simultaneously, whereas $(a, b) \in ser$ means that their simultaneous execution is equivalent with sequential execution a before b. Any nonempty set of simultaneously executable actions $A \subseteq \Sigma$ such that $(a, b) \in sim$, for all distinct $a, b \in A$, is called a *step*. The set $\mathbb{S}$ of all steps over Θ is called a *step alphabet*. Finite sequences in $\mathbb{S}^*$, including the empty one denoted by λ, are called *step sequences*. To avoid the collision with standard notions used in formal languages theory, we write (abc) instead of $\{a, b, c\}$ to denote a step containing actions a, b and c.

We lift a number of notions and notations introduced for words to the level of step sequences. In what follows, $\Theta = (\Sigma, sim, ser)$ is a *fixed* comtrace alphabet.

Let $\circ$ denote the step sequence concatenation operator (usually omitted). Since the concatenation operator is associative, the triple $(\mathbb{S}^*, \circ, \lambda)$ is a monoid.

As in the case of traces, we define the relation that identifies similar step sequences. Following the intuitive description of serialisability relation, for $(a, b) \in ser$, we identify a single step (ab) with a step sequence $(a)(b)$. Formally, the *comtrace congruence* over Θ, denoted by $\equiv_\Theta$, is the reflexive, symmetric and transitive closure of the relation $\sim_\Theta \subseteq \mathbb{S}^* \times \mathbb{S}^*$, where $w \sim_\Theta v$ if there exist $u, z \in \mathbb{S}^*$ and $A, B, C \in \mathbb{S}$ satisfying $w = uAz$, $v = uBCz$, $A = B \cup C$ and $B \times C \subseteq ser$. Note that $B \cap C = \varnothing$ as ser is irreflexive.

Equivalence classes of the relation $\equiv_\Theta$ are called *combined traces*, in short *comtraces* (see [2, 3]). The combined trace containing a given step sequence w is denoted by $[w]$. The set of all comtraces is denoted by $\mathbb{S}^*/_\equiv$, and the triple $(\mathbb{S}^*/_\equiv, \circ, [\lambda])$ is a (comtrace) monoid, where $\tau \circ \tau' = [w \circ w']$, for any step sequences $w \in \tau$ and $w' \in \tau'$. Comtrace concatenation is well-defined as $[w \circ w'] = [v \circ v']$, for all $w, v \in \tau$ and $w', v' \in \tau'$. We say that a comtrace τ is a prefix of a comtrace τ' if there is a comtrace τ'' such that $\tau \circ \tau'' = \tau'$.

In a technical discussion, we will use the following set of relations covering all possible relationships between individual actions:

Dependence $dep = (\Sigma \times \Sigma) \setminus sim$, and **independence** $ind = ser \cap ser^{-1}$. Both relations have their counterparts in trace theory, and so we denote them in the same way. If two actions are dependent then their two occurrences must happen in the same order (and never simultaneously) in all the step sequences forming a given comtrace. Two actions are independent if they can be executed in any order as well as simultaneously (as $ser \subseteq sim$).

Semi-independence $sin = sim \setminus ser$. We say that a pair of actions that are neither dependent nor independent is semi-independent. Note that, since serialisability is not necessary symmetric, the semi-independence relation may be not symmetric. We further split it into two parts, symmetric ssm and antisymmetric wdp.

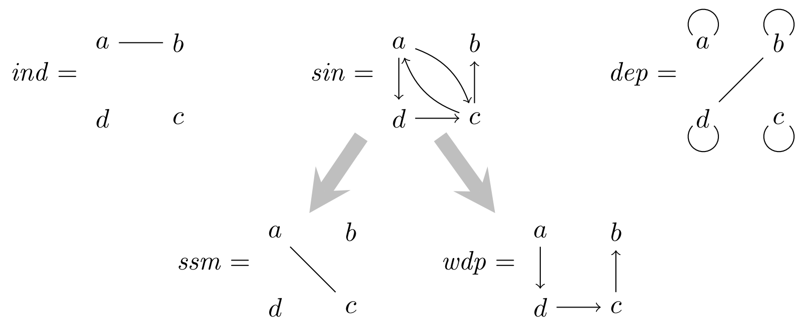
Strong simultaneity $ssm = sim \setminus (ser \cup ser^{-1})$. If two actions are strongly simultaneous then their two occurrences must happen in the same order or simultaneously in all the step sequences forming a given comtrace.

Weak dependence $wdp = ser^{-1} \setminus ser$. Two actions are weakly dependent if they can be serialised only in one way; hence this relation is antisymmetric.

Example 2. Consider a comtrace alphabet Θ with four actions $\Sigma = \{a, b, c, d\}$ together with a simultaneity and serialisability relations, ser and sim given by:

$$sim = \begin{tikzcd} a \arrow[r, dash] \arrow[d, dash] \arrow[dr, dash] & b \arrow[d, dash] \\ d \arrow[r, dash] & c \end{tikzcd} \qquad ser = \begin{tikzcd} a \arrow[r, bend left] & b \arrow[l, bend left] \arrow[d] \\ d \arrow[u] & c \arrow[l] \end{tikzcd}$$

Then the five derived relations on actions are as follows:



In this case we have $(d)(ab)(cd)(abc)(acd) \equiv (ad)(bc)(d)(b)(ac)(acd)$. $\square$

2 Lexicographical normal form

We follow the notations from [8]. Let $(\Sigma, \leq)$ be a standard, linear order on actions, and the order $(2^{\Sigma}, \widehat{\leq})$ as defined in (1) with a natural extension of $\widehat{\leq}$ to step sequences.

We define the *lexicographical normal form* of a comtrace τ as the least, with respect to $\widehat{\leq}$, step sequence contained in τ, and denote it by $minlex(\tau)$. Another canonical representation of a comtrace is its *Foata normal form*, where actions are grouped to achieve their maximal concurrent execution. A step sequence $Foata(\tau) = A_1 \ldots A_n \in \mathbb{S}^*$ is in Foata normal form if, for each $i \leq n$, whenever $Av \equiv_{\Theta} A_i \ldots A_k$ for some $A \in \mathbb{S}$ and $v \in \mathbb{S}^*$, then $A \subseteq A_i$, see [4] for more details.
Let us consider a step $X \in \mathbb{S}$ and an equivalence relation $\equiv_X \subseteq X \times X$. For two actions $a, b \in X$ we say that $a \equiv_X b$ if and only if $(a, b) \in (sin|_X)^*$ and $(b, a) \in (sin|_X)^*$. We say that a step $A \in \mathbb{S}$ is *indivisible* if and only if $\equiv_A = A \times A$. The set of all indivisible steps is denoted by $\widehat{\mathbb{S}}$. By $indiv(\tau)$ we denote the set of all step sequences contained in the comtrace τ that are built using indivisible steps only. Intuitively, we can treat the indivisible step sequences belonging to $indiv(\tau)$ as classical sequences over the alphabet $\widehat{\mathbb{S}}$. Hence we define two complementary relations over this alphabet, the independence relation $\widehat{ind}$ and the dependence relation $\widehat{dep}$. We say that two indivisible steps A and B are (*indivisibly*) *independent* if $A \times B \subseteq ind = ser \cap ser^{-1}$, otherwise A and B are (*indivisibly*) *dependent.*

It is worth noting that the notion of indivisibility was introduced, in the case of Mazurkiewicz traces, in [12]. This notion allows there some dependent actions to occur simultaneously. In the presented model of combined traces such relationship is covered by the strong simultaneity - the main reason for the presence of indivisible steps.

The normal forms play a crucial role in the generation procedures presented in this paper. We recall some facts related with those forms.

Proposition 3 ([8]). *Let τ be a comtrace. The Foata normal form of τ is the greatest, with respect to order $\widehat{\leq}$, step sequence contained in τ.*

Proof. Let $u = A_1 \dots A_n, v = B_1 \dots B_m$, $u \neq v$, $u \equiv_\Theta v$, and u be in Foata canonical form. Moreover, let $i = min\{j | j \leq n \wedge A_j \neq B_j\}$. Note that such a number i exists, since $u \neq v$ and $u \equiv_\Theta v$ so one sequence cannot be a prefix of another. We have $A_1 \dots A_{i-1} = B_1 \dots B_{i-1}$, so directly form the definition of Foata canonical form $B_i \neq A_i \wedge B_i \subseteq A_i$. Since $B_i \widehat{\leq} A_i$, we have $B_i \dots B_m \widehat{\leq} A_i \dots A_n$, and $v \widehat{\leq} u$. □

Proposition 4 ([8]). *Let τ be a comtrace. All steps contained in lexicographical normal form of τ are indivisible ($minlex(\tau) \in indiv(\tau)$).*

Proof. Suppose, to the contrary, that $minlex(\tau) = uAv$ contains a non-indivisible step A. For two disjoint steps B and C we have a step sequence $uBCv \in \tau$ which is different from the step sequence $minlex(\tau)$. Since $B \subseteq A$ and $A \neq B$ we have $uBCv \,\widehat{\leq}\, uAv$ so we found a step sequence contained in τ that is lexicographically smaller than $minlex(\tau)$, which contradicts our assumption. Hence all steps contained in $minlex(\tau)$ are indivisible. □

Proposition 5. *Let $w = A_1 \dots A_m$ be a step sequence in lexicographical normal form. For every $n > m$ there exists a step sequence in lexicographical normal form $A_1 \dots A_m B_{m+1} \dots B_n$ such that all B_i are singletons (steps consisting only one action).*

Proof. Let a be an arbitrary element of A_m. Note that (a) is dependent as a step not only with (a) but also with A_m. This means that any B_i may be set to (a). □

Theorem 6 ([8]). *Let τ be a comtrace. The set $indiv(\tau)$ is a trace over the concurrent alphabet $(\widehat{\mathbb{S}}, \widehat{ind})$.*

Proof. To prove the statement of the theorem it is sufficient to show two facts. Firstly, we need to prove that relation $\widehat{ind}$ is symmetric and irreflexive. Secondly, we need to argue that by the transposing of two subsequent and independent actions (in fact indivisible steps) we can reach any of other elements of the set $indiv(\tau)$ and cannot go beyond this set.

We start from the first statement. By the definition of ser the relation $ind = ser \cap ser^{-1}$ is symmetric and irreflexive. Since two indivisible steps A and B are in relation $\widehat{ind}$ if all pairs of actions $(a, b) \in A \times B$ are independent, we conclude that the relation $\widehat{ind}$ is also symmetric and irreflexive.

Let $w = uABv$ be a step sequence from $indiv(\tau)$ and $(A, B) \in \widehat{ind}$. By the definition of the $\widehat{ind}$ relation we have $AB \sim_\Theta C$ and $BA \sim_\Theta C$, where $C = A \cup B$. Therefore $uABv \equiv_\Theta uBAv$ and the set $indiv(\tau)$ is equal to its own trace closure.

Let us suppose that there are two comtrace equivalent step sequences u and v belonging to $indiv(\tau)$ that are not trace equivalent. Hence they differ in at least one projection to binary dependent subalphabet so there are two occurrences of indivisible steps A and B that appear in the two different orders and are dependent ($(A,B) \in \widehat{dep}$). For definiteness, let A precedes B in the step sequence u, while B precedes A in the step sequence v. From the definition of comtrace equivalence there exits a sequence of equivalent step sequences $(w_i)_{i=1\dots m}$ such that $u = w_1$, $w_i \sim_\Theta w_{i+1}$, and $w_m = v$. In this sequence there have to exist an element w_i where the considered occurrences of indivisible steps was the last time in the same order as in u ($w_i = w'_i X_i Y_i w''_i$ and $w_{i+1} = w'_{i+1} Z_i w''_{i+1}$ and $A \subseteq X_i$ and $B \subseteq Y_i$). Hence $A \times B \subseteq ser$. Moreover, there exists an element w_j where the considered occurrences first time after w_i are in the same order as in v ($w_j = w'_j Z_j w''_j$ and $w_{j+1} = w'_{j+1} X_j Y_j w''_{j+1}$ and $A \subseteq X_j$ and $B \subseteq Y_j$). Hence also $B \times A \subseteq ser$. Therefore $(A,B) \in \widehat{ind}$, which is impossible and completes the proof. □

3 Projection representation

In the trace theory concerning projections onto cliques of a dependence relation graph (see also [11]) appeared to be a very useful tool. A special kind of such clique cover uses the binary and unary cliques only (see also [7]). Trying to extend this theory to the case of comtraces, we have to introduce new symbols, not included in Σ, connected with the preorder of weak causality (indivisible steps in fact). We use the simplified approach (concentrating on binary and unary cliques only). It allows us to add only one new symbol $\perp$ used in the case of strong simultaneous actions which occur together.

Let $a, b \in \Sigma$ and $(a,b) \notin ind$ (possibly $a = b$). For each such pair we define the projection function $\Pi_{a,b} : \mathbb{S}^* \to (\Sigma \cup \{\perp\})^*$ as follows.
For a step $A \in \mathbb{S}$ we have

$$\Pi_{a,b}(A) = \Pi_{b,a}(A) = \begin{cases} \varepsilon & \text{if } \{a,b\} \cap A = \varnothing \\ a & \text{if } \{a,b\} \cap A = \{a\} \\ b & \text{if } \{a,b\} \cap A = \{b\} \\ ab & \text{if } \{a,b\} \subseteq A \wedge (b,a) \in wdp \\ ba & \text{if } \{a,b\} \subseteq A \wedge (a,b) \in wdp \\ \perp & \text{otherwise (if } \{a,b\} \subseteq A \wedge (a,b) \in ssm) \end{cases} \tag{2}$$

For a step sequence $w = A_1 A_2 \dots A_n$ we have

$$\Pi_{a,b}(w) = \Pi_{a,b}(A_1) \cdot \Pi_{a,b}(A_2) \cdot \dots \cdot \Pi_{a,b}(A_n). \tag{3}$$

Projection representation of a step sequence w is a function

$$\begin{aligned} &\Pi_w : \Sigma \times \Sigma \setminus ind \longrightarrow (\Sigma \cup \{\perp\})^*, \\ &\Pi_w(a,b) = \Pi_{a,b}(w). \end{aligned}$$

Any function of these domain and image is called the *projection set*. Moreover, for two projection sets Π_1 and Π_2 we define their concatenation $\Pi_1 \bullet \Pi_2$ componentwise (i.e., $(\Pi_1 \bullet \Pi_2)(a,b) = \Pi_1(a,b) \cdot \Pi_2(a,b)$).

Theorem 7 **([8]).** *Let w, u be step sequences over a comtrace alphabet Θ.*

$$w \equiv_{\mathbb{S}} u \;\Leftrightarrow\; \forall_{(a,b)\in\Sigma\times\Sigma\setminus ind}\; \Pi_{a,b}(w) = \Pi_{a,b}(u).$$

Proof.
$\Rightarrow$:
We first prove that

$$w \equiv_{\Theta} u \;\Rightarrow\; \forall_{(a,b)\notin ind}\; \Pi_{a,b}(w) = \Pi_{a,b}(u).$$

According to the definition of comtrace equivalence, it is sufficient to prove the stated statement it the case of equivalent step sequences $w = A$ and $u = BC$. Let $a, b \in A$. We consider all but one possible relationships of these actions (the excepted case is naturally independence).

Case 1: $(a, b) \in dep$.
Since actions a and b occur simultaneously in the step A it is impossible.

Case 2: $(a, b) \in ssm$.
Since actions a and b are strongly simultaneous, they both have to occur in step B or C. It means that

$$\Pi_{a,b}(A) = \perp \epsilon = \epsilon \perp = \Pi_{a,b}(B)\Pi_{a,b}(C) = \Pi_{a,b}(BC).$$

Case 3: $(a, b) \in wdp \wedge a, b \in B$.
Since $B \times C \subseteq ser$, it is impossible that $b \in C$ and $a \in B$. If they both belong to one step, we have

$$\Pi_{a,b}(A) = ba \cdot \epsilon = \epsilon \cdot ba = \Pi_{a,b}(B)\Pi_{a,b}(C) = \Pi_{a,b}(BC),$$

while belonging to the different steps (namely $b \in B$ and $a \in C$) gives

$$\Pi_{a,b}(A) = ba = \Pi_{a,b}(B)\Pi_{a,b}(C) = \Pi_{a,b}(BC),$$

which completes the first part of the proof.

$\Leftarrow$:
Now, let us assume that we have two step sequences $u, v \in \mathbb{S}^*$ and

$$\forall_{(a,b)\notin ind}\; \Pi_{a,b}(w) = \Pi_{a,b}(u).$$

Without losing any generality we can assume that $u = Au'$ is in the lexicographical canonical form and v consists of indivisible steps only. We claim that there exist such $v', v'' \in \mathbb{S}^*$ that $v = v'Av''$. Moreover, we claim that no action occurring in A may occur in v' and $A \times alph(v') \subseteq ind$.

Directly from the definition of the projection representation we see that all projections to the subalphabets containing actions from the indivisible step A starts with the actions contained in A. More precisely, if both $a, b \in A$ then $\Pi_{a,b}(u)$ starts with ab, ba or $\perp$, depending on the relation between a and b. If $a \in A$ but $b \notin A$ however, $\Pi_{a,b}(u)$ starts with a single action a.

Let v' be the longest prefix of v such that $alph(v') \cap A = \varnothing$ and $v = v'Bv''$. Obviously, all projections to the subalphabets containing actions from the step A are equal for v and Bv''. Moreover, from the definition of the indivisible step, between every two actions a, b contained in A there is a sequence of pairwise different actions $a = a_1, \dots, a_n = b$ contained in A such that for every $i < n$ we have $(a_{i+1}, a_i) \in sin$. It means that for every such a pair of consecutive actions we have $\Pi_{a_i,a_{i+1}}(B) = a_i a_{i+1}$ if $(a_{i+1}, a_i) \in wdp$ or $\Pi_{a_i,a_{i+1}}(B) = \perp$ if $(a_{i+1}, a_i) \in ssm$. Nevertheless, if a_{i+1} is in B then also a_i have to be in B. Otherwise $\Pi_{a_i,a_{i+1}}(B)$ would start with a_{i+1}. This proves that, since $A \cap B \neq \varnothing$, $A \subseteq B$. Using the same arguments, we see that since B is indivisible, no other action may occur in B and $A = B$.

It remains to show that $A \times alph(v') \subseteq ind$. Let $a \in A$ and $c \in alph(v')$. Naturally, $c \notin A$ from the definition of sequence v'. In the step sequence v the action c appears before action a so, if the are not independent, $\Pi_{a,c}(v) = \Pi_{a,c}(u)$ starts with c. But $a \in A$ and $c \notin A$ so $\Pi_{a,c}(u)$ starts with a. This contradicts our assumption that a and c may be not independent and proves that $v \equiv_\Theta Av'v''$. Repeating above reasoning, we achieve that u is the lexicographical canonical form of v which ends the second part of the proof. $\square$

Due to Theorem 7, all representatives of fixed comtrace have the same projection representation. Projection representation of a comtrace τ is a function Π_τ, such that $\Pi_\tau = \Pi_w$ for an arbitrary step sequence $w \in \tau$.

Example 8. Consider a step sequence $w = (d)(ab)(cd)(abc)(acd)$ over comtrace alphabet introduced in the example 2. The projection representation Π_w is fully determined by its value on the following pairs:

$$\begin{aligned} \Pi_w(a,c) &= ac \perp\perp, \\ \Pi_w(a,d) &= dadada, \\ \Pi_w(b,c) &= bcbcc, \\ \Pi_w(b,d) &= dbdbd, \\ \Pi_w(c,d) &= dcdccd. \end{aligned}$$

Moreover, a step sequence $v = (ad)(bc)(d)(b)(ac)(acd)$ has the same project representation, i.e. is a representative of the same comtrace.

3.1 Reconstruction

We recall the nondeterministic procedure of reconstruction a step sequence from its projection representation (see [8]). At each stage of the algorithm, we take one element from the set of possible steps. The necessity of choice causes the nondeterminism of the procedure. Using a proper selection strategy we can obtain every step sequence contained in a given comtrace, including those minimal and maximal ones (with respect to the relation $\widehat{\leq}$). Such a strategy determines the whole procedure. In fact, the reconstruction algorithm can be applied to any projection set returning the largest possible step sequence of its prefix.

At first we introduce the notion of impossibility based on the conditionally possible actions. Let Π be a projection set. We say that an action $a \in \Sigma$ is *conditionally possible* for projection function Π, if it is in front of all projections $\Pi(a,b)$, or in cases when it is not in front of $\Pi(a,b)$ we have $(a,b) \in wdp \land pref_2(\Pi(a,b)) = ba$ or $(a,b) \in ssm \land pref_1(\Pi(a,b)) = \perp$.

Formally, for all $b \in \Sigma$ the following implications have to be satisfied:

- $(a,b) \in dep \Rightarrow pref_1(\Pi(a,b)) = a$
- $(b,a) \in wdp \Rightarrow pref_1(\Pi(a,b)) = a$
- $(a,b) \in wdp \Rightarrow pref_1(\Pi(a,b)) = a \lor pref_2(\Pi(a,b)) = ba$
- $(a,b) \in ssm \Rightarrow pref_1(\Pi(a,b)) = a \lor pref_1(\Pi(a,b)) = \perp$

We denote the set of all conditionally possible actions as cpa and define the relation $sin_\Pi \subseteq \Sigma \times \Sigma$, which describes the conditions that must be satisfied. Let $(a,b) \in wdp \land pref_2(\Pi(a,b)) = ba$ or $(a,b) \in ssm \land pref_1(\Pi(a,b)) = \perp$. In such situation, we say that the existence of the action b in the constructed step is the *necessary condition* for the presence of the action a in this step, which is denoted by $(a,b) \in sin_\Pi$.

We exclude conditionally possible actions with impossible to satisfy conditions to form a set of *(truly) possible* actions. Any action $a \in \Sigma$ that is not conditionally possible in Π is *impossible* in Π. Moreover, any conditionally possible action a under the impossible condition ($\exists_b\ (a,b) \in sin_\Pi \land b \notin cpa$) is also impossible. Formally, the set of actions, impossible for projection function Π, is the least set imp that satisfies the following conditions:

- $\Sigma \setminus cpa \subseteq imp$
- $b \in imp \land (a,b) \in sin_\Pi \Rightarrow a \in imp$

By $M(\Pi)$ we denote the maximal step that is possible for projection set Π (i.e., the set of all truly possible actions for Π).

Proposition 9. *Let Π be a projection set over $\Theta = (\Sigma, sim, ser)$ and cpa a set of all conditionally possible actions in Π with necessary conditions relation sin_Π. Then $sin_\Pi \subseteq sin$.*

Proof. By definition $(a,b) \in sin_\Pi$ implies that $(a,b) \in wdp$ or $(a,b) \in ssm$. Recalling the definition of relations sin, ssm and wdp we easily deduce that $sin = ssm \cup wdp$, hence $sin_\Pi \subseteq sin$. □

Proposition 10. *Let Π be a projection set over $\Theta = (\Sigma, sim, ser)$ and $M(\Pi)$ a set of all truly possible actions in Π with necessary conditions relation sin_Π. Then $sin_\Pi|_{M(\Pi)} = sin|_{M(\Pi)}$.*

Proof. By Proposition 9 we have $sin_\Pi|_{M(\Pi)} \subseteq sin|_{M(\Pi)}$.

Note that $M(\Pi) \subseteq cpa$. Let $a, b \in M(\Pi)$ and $(a,b) \in sin$. Then $(a,b) \in wdp$ or $(a,b) \in ssm$.

Case 1: $(a,b) \in wdp$
We have $a \in cpa$ so $\big(pref_1(\Pi(a,b)) = a \lor pref_2(\Pi(a,b)) = ba\big)$ and $b \in cpa$ so $pref_1(\Pi(a,b)) = b$, hence $pref_2(\Pi(a,b)) = ba$. Therefore $(a,b) \in sin_\Pi$.

Case 2: $(a,b) \in ssm$
We have $a \in cpa$ so $\big(pref_1(\Pi(a,b)) = a \vee pref_1(\Pi(a,b)) = \perp\big)$ and $b \in cpa$ so $\big(pref_1(\Pi(a,b)) = b \vee pref_1(\Pi(a,b)) = \perp\big)$, hence $pref_1(\Pi(a,b)) = \perp$. Therefore $(a,b) \in sin_{\Pi}$.

Summing up we achieve that $sin|_{M(\Pi)} \subseteq sin_{\Pi}|_{M(\Pi)}$, which concludes the proof. □

We choose any subset $X \subseteq M(\Pi)$ as a seed of the first step for Π. We compute its closure (with respect to the relation $sin|_{M(\Pi)}$) and denote the result by $X^{\uparrow}$. We can cut it from the left side of Π achieving Π'. If Π is a projection representation of comtrace τ, ss a result we get $\tau = X^{\uparrow} \circ \sigma$, where $\Pi(\sigma) = \Pi'$. For technical reasons, we emphasize *self-closed seeds* (i.e., subsets $X \subseteq M(\Pi)$ satisfying the condition $X = X^{\uparrow}$). Properly cutting self-closed seeds we can compute any representative of a comtrace τ. In particular, we can use a maximal or minimal strategy. In the maximal strategy, we always take the whole set $M(\Pi)$, and as a result we obtain the Foata normal form of τ. On the other hand, in the minimal strategy we take the least, with respect to the order $\widehat{\leq}$, self-closed seed X, and obtain the lexicographical normal form.

4 Auxiliary procedures

In algorithms presented in this paper we use some technical auxiliary procedures. We assume that the comtrace alphabet Θ is fixed and given as a global variable with all sufficient relations (especially *sin*). In what follows, τ is a comtrace, $\Pi = \Pi_{\tau}$ a projection represenataion of τ, and $A \in \mathbb{S}$ a single step. Morover, in the complexity discussion, by k we denote the size of the alphabet Σ, by p the size of the indivisible steps alphabet $\widehat{\mathbb{S}}$, by m the number of steps in considered step sequence w and by n the number of atomic action occurrences in w.

We start with some algebraic operations on the projection representation Π. The procedure RIGHT-ADD(Π, A) returns the projection set Π' such that $\Pi'(a,b) = \Pi(a,b) \cdot \Pi_{a,b}(A)$. The procedure LEFT-ADD$(\Pi, A)$ is defined similarly and returns $\Pi'(a,b) = \Pi_{a,b}(A) \cdot \Pi(a,b)$. We also use the inverse procedures RIGHT-CUT and LEFT-CUT. If for Π_{τ} there exists a comtrace τ' such that for all (a,b) we have $\Pi_{\tau}(a,b) = \Pi_{\tau'}(a,b) \cdot \Pi_{a,b}(A)$, then the procedure RIGHT-CUT returns $\Pi_{\tau'}$. Otherwise its result is not defined. The procedure LEFT-CUT is defined similarly. Note that the operation LEFT-CUT is well defined for every self-closed subset of the set of truly possible actions of Π.

Observe that for a step A we have $|A| \leq k$. For each action a we can precompute the list of pointers to the projections related to a. Since every action a occurs in at most k projections, the size of each list is limited by k. Moreover, the single operation of adding or removing an individual action a, either on the left or the right side of $\Pi(a,b)$, can be implemented in constant time. Therefore, each of the above mentioned procedures can be implemented in the time complexity of $O(k^2)$.

Example 11. Let us recall the projection representation Π_w of the step sequence $w = (d)(ab)(cd)(abc)(acd)$ from Example 8. The projection representations obtained by left cutting a step (ad) and right adding a step (b) looks as follows:

	Π_w	LEFT-CUT$(\Pi_w, (ad))$	RIGHT-ADD$(\Pi_w, (b))$
(a, c)	$ac \perp\perp$	$\boldsymbol{c} \perp\perp$	$ac \perp\perp$
(a, d)	$dadada$	$\boldsymbol{dada}$	$dadada$
(b, c)	$bcbcc$	$bcbcc$	$\boldsymbol{bcbccb}$
(b, d)	$dbdbd$	$\boldsymbol{bdbd}$	$\boldsymbol{dbdbdb}$
(c, d)	$dcdccd$	$\boldsymbol{cdccd}$	$dcdccd$

□

The next important procedure is FIND-M(Π), which returns the set $M(\Pi)$ of all truly possible actions for Π. The procedure starts with computing the set cpa for Π by checking the necessary conditions described in equation (2). Next, it traverses the graph of the relation sin_Π excluding from the precomputed set all impossible actions. Following the description from section 3.1 the computation of the set M requires a few operations on the graph consisting of k vertices. We utilise once more the idea of the precomputed lists of pointers to the projections related to the action a. Therefore, the whole procedure can be implemented in time O(k^2).

Example 12. Let us recall the comtrace alphabet introduced in Example 2 and the step sequence $w = (d)(ab)(cd)(abc)(acd)$. Moreover, let us consider the step sequence $w' = (d)(abc)(acd)$, which is a (comtrace) suffix of w. Then, the relations sin_Π, and sets cpa and M are depicted below:

$$sin = \begin{tikzcd} a \arrow[d] \arrow[dr, bend left] & b \\ d \arrow[r] & c \arrow[u] \arrow[ul, bend left] \end{tikzcd} \qquad dep = \begin{tikzcd} a \arrow[loop above, dash] & b \arrow[dl, dash] \arrow[loop above, dash] \\ d \arrow[loop below, dash] & c \arrow[loop below, dash] \end{tikzcd}$$

	Π_w
(a, c)	$ac \perp\perp$
(a, d)	$dadada$
(b, c)	$bcbcc$
(b, d)	$dbdbd$
(c, d)	$dcdccd$

$$cpa = \{a, d\} \qquad sin_\Pi = \begin{tikzcd} a \arrow[d] & b \\ d & c \end{tikzcd} \qquad M = \{a, d\}$$

	$\Pi_{w'}$
(a, c)	$\perp\perp$
(a, d)	$dada$
(b, c)	bcc
(b, d)	dbd
(c, d)	$dccd$

$$cpa = \{a, d\} \qquad sin_\Pi = \begin{tikzcd} a \arrow[d] \arrow[dr] & b \\ d & c \end{tikzcd} \qquad M = \{d\}$$

Note that for both Π_w and $\Pi_{w'}$ sets cpa are equal. However, associated with them relations sin_Π and sets M differ. □

The procedure NEXT(X, A) computes the lexicographical successor (with respect to the order $\widehat{\leq}$) of the set A, i.e. the minimal set $B \subseteq X$, where $B \neq A$, satisfying the condition $A \widehat{\leq} B$. Similarly, the procedure PREV(X, A) computes the lexicographical predecessor of the set A. In boundary cases both procedures return NULL value. Using the standard technique of representing the sets as a binary vector, one can implement this procedure in $\mathrm{O}(k)$ time. However, taking into acount the whole generation procedure the time of a single execution of NEXT is amortised constant.

The procedure ISINDIV(A) tests indivisibility of the step A. The returned result is *true*, if for all $(a, b) \in A$ we have $(a, b) \in (sin|_A)^*$ and $(b, a) \in (sin|_A)^*$. Otherwise its result is *false*. The considered condition can be reduced to checking if the subgraph of the graph of the relation *sin* induced by actions contained in A is strongly connected. Such an operation can be done in time $\mathrm{O}(k^2)$.

Example 13. Recall the comtrace alphabet from Example 2 (and especially the relation *sin*). In this case all indivisible steps in lexicographical order are as follows:

$$\begin{tikzcd} a \arrow[d] \arrow[dr, bend left] & b \\ d \arrow[r] & c \arrow[u] \arrow[ul, bend left] \end{tikzcd} \longrightarrow \left\{ a,\ b,\ c,\ d,\ \begin{tikzcd} a \arrow[dr, bend left] & \\ & c \arrow[ul, bend left] \end{tikzcd},\ \begin{tikzcd} a \arrow[d] \arrow[dr, bend left] & \\ d \arrow[r] & c \arrow[ul, bend left] \end{tikzcd} \right\}$$

□

For a lexicographically minimal representative of a comtrace $w = A_1 A_2 \ldots A_m$ and an indivisible step A, the procedure ISMINLEX(w, A) checks if $w \circ A$ is the lexicographically minimal representative of $[w \circ A]$. It is enough to test if there exists a suffix $A_j \circ A_{j+1} \circ \ldots \circ A_m$ of w, such that all steps $A_j, \ldots, A_m$ are independent with A and $A \widehat{\leq} A_j$. Recall that two steps B and C are called independent if $B \times C \subseteq ind$. In such case the result of the procedure is *false*, otherwise *true*. The procedure ISMINLEX compares the indivisible step A with at most m steps. Both test of commutation and lexicographical comparison can be done in time $\mathrm{O}(|A_i| \cdot |A|)$. Since $\sum_{i=1}^{m} |A_i| = n$ and $\forall_i |A_i| \leq k$, the procedure has the time complexity of $\mathrm{O}(nk)$.

We introduce also the procedure IS-SELF-CLOSED(M, A) checking if a subset $A \subseteq M$ of actions is self-closed with respect to the relation $sin|_M$. Intuitively this procedure verifies if for a given step M the set of actions A forms a proper step such that $M \equiv A \circ (M \setminus A)$. It is enough to test, if for each action $a \in A$ and each $b \in M$ such that $(a, b) \in sin|_M$, we have $b \in A$. In such a case, the result of the procedure is *true*, otherwise *false*. Straightforward implementation has the time complexity $\mathrm{O}(k^2)$.

The last two procedures are PREV-S-CLS(X, A) and NEXT-INDIV(X, A). For a given subset $A \subseteq X$, using the order $\widehat{\leq}$, they return the previous self-closed or the next indivisible subset of X, respectively. They can be easily implemented by combining the previously described procedures. However, such an implementation would require the testing of a number of subsets of M potentially close to the number of all its subsets, and therefore is inefficient. One can improve the running time of PREV-S-CLS by changing the order on steps and using the alphabet of indivisible steps (local change only). Such a reordering on steps cause the loss of the lexicographicality of the generation, but allows to obtain the complexity of $O(k^2)$. In a case of the procedure NEXT-INDIV, the order change does not seem to be such a good solution as in the case of PREV-S-CLS. Nevertheless, one can enhance its time complexity by the precomputation of the whole order in the preprocessing phase. Having all elements of $\widehat{\mathbb{S}}$ stored in an array, the procedure would run in the constant time.

5 Generation of all representatives of a given comtrace

In this section we present an algorithm for generation of all representatives of a given comtrace τ. During the generation procedure we keep the common prefix of the two consecutive representatives, changing only the *working suffix* they differ.

To enumerate all representatives of a given comtrace τ it is sufficient to find its Foata normal form and iterate the procedure of finding previous representative until one reaches the lexicographical normal form of τ. The presentation of the algorithms starts with the one used to compute the Foata normal form of a comtrace τ. The input for this procedure is Π_τ – the projection representation of the considered comtrace.

The code of the procedure that computes the Foata normal form of a comtrace is presented in Algorithm 1. The recursive nature of the procedure is the reason of the statement in line 1. It is the stop condition for the recursion. We compute the set M of all truly possible actions for Π_τ (i.e. the largest possible step). We cut it from Π_τ and compute the Foata normal form of the reduced projection set. Observe that the number of the recursive calls of the procedure FOATA does not exceed the number of steps contained in the considered comtrace. Hence, the running time of the procedures FIND-M and LEFT-CUT implies the time complexity of $O(mk^2)$.

Algorithm 1: Foata normal form (FOATA)

```
Input: Projection: $\Pi_\tau$
Output: Step sequence: $A_1 A_2 \dots A_m$ – the greatest in the lexicographical
        order element of $\tau$
1 if $\Pi_\tau \neq \varnothing$ then
2     M:=FIND-M($\Pi_\tau$);
3     return $M \circ$FOATA(LEFT-CUT($\Pi_\tau, M$));
4 else
5     return NULL;
```

Example 14. Let us recall the comtrace alphabet from Example 2 and the projection representation Π_w of the step sequence $w = (d)(ab)(cd)(abc)(acd)$. The subsequent stages of computing its Foata normal form are depicted below.

$$\text{FOATA}\begin{pmatrix} \Pi \\ (a,c)\ ac\perp\perp \\ (a,d)\ dadada \\ (b,c)\ bcbcc \\ (b,d)\ dbdbd \\ (c,d)\ dcdccd \end{pmatrix} = (ad)\circ\text{FOATA}\begin{pmatrix} \Pi \\ (a,c)\ c\perp\perp \\ (a,d)\ dada \\ (b,c)\ bcbcc \\ (b,d)\ bdbd \\ (c,d)\ cdccd \end{pmatrix} =$$

$$(ad)(bc)\circ\text{FOATA}\begin{pmatrix} \Pi \\ (a,c)\ \perp\perp \\ (a,d)\ dada \\ (b,c)\ bcc \\ (b,d)\ dbd \\ (c,d)\ dccd \end{pmatrix} = (ad)(bc)(d)\circ\text{FOATA}\begin{pmatrix} \Pi \\ (a,c)\ \perp\perp \\ (a,d)\ ada \\ (b,c)\ bcc \\ (b,d)\ bd \\ (c,d)\ ccd \end{pmatrix} =$$

$$(ad)(bc)(d)(abc)\circ\text{FOATA}\begin{pmatrix} \Pi \\ (a,c)\ \perp \\ (a,d)\ da \\ (b,c)\ c \\ (b,d)\ d \\ (c,d)\ cd \end{pmatrix} = (ad)(bc)(d)(abc)(acd)$$

□

The algorithm 2 takes as an input w_1 – a representative of a comtrace τ and its projection representation Π_τ. As a result it returns w_2 – the previous (with respect to $\widehat{\leq}$) representative of τ or NULL if w_1 is already in the lexicographical normal form.

The main variables used in this algorithm are i – the position where we cut passed step sequence, and two projection sets $\Pi_{pref} = \Pi(A_1 \ldots A_i)$ and $\Pi_{suff} = \Pi(A_{i+1} \ldots A_m)$. We compute the largest position i, such that Π_{suff} is not in the lexicograhical normal form. If there is no such position, we conclude that the whole sequence is in the lexicographical normal form. Hence, there is no previous representative and we return NULL value. During the search we move the cut position from m to 0, moving A_i from Π_{pref} to Π_{suff}. We invoke the procedure FIND-M to compute the maximal step M that can be cut from the projection set Π_{suff}. We use it to find out if there is any valid step $A'_i \subset M$ smaller than A_i. After the positive check we replace A_i by largest possible A'_i (getting the prefix $A_1 \ldots A_{i-1}A'_i$ of the previous representative), and compute the Foata normal form of the reduced Π_{suff}. If there is no such step A'_i we continue searching.

Algorithm 2: Previous representative

```
Input: Step sequence: $A_1A_2\dots A_m$, Projection: $\Pi_\tau$
Output: Step sequence: $B_1B_2\dots B_{m'}$ – previous in lexicographical order
        equivalent to $A_1A_2\dots A_m$

$i := m$;
$\Pi_{pref} := \Pi_\tau$; $\Pi_{suff} := \varnothing$;;
while $i > 0$ do
    RIGHT-CUT($\Pi_{pref}, A_i$); LEFT-ADD($\Pi_{suff}, A_i$);
    M:=FIND-M($\Pi_{suff}$);
    $A'_i$ := PREV-S-CLS($M, A_i$);
    if $A'_k \neq NULL$ then
        return $A_1 \dots A_{i-1}A'_i\circ$FOATA(LEFT-CUT($\Pi_{suff}, A'_i$));
    else
        $i := i-1$;
return NULL;
```

The most time consuming part of the above algorithm is the procedure PREV-S-CLS, see its description in the previous section. We assume here its implementation with time complexity of $O(k^2)$. The while-loop (lines 3-10) performs at most m iterations, wherein the condition in line 7 is true only once and causes breaking the loop. Taking into account the running time of the auxiliary procedures, the complexity of the whole algorithm is $O(mk^2)$.

It is worth noting that the algorithm searches the tree of all possible choices without building or storing the tree (neither directly nor using recursion). The price of such behavior is the recomputation of the maximal step M. One can reduce this cost storing with each step A_i the previously computed maximal step M (which is valid for a suffix $A_i \dots A_m$). Those maximal steps may be then computed only once during the procedure FOATA. Multiplying the used memory twice we avoid the effect of multiple recomputation, and still do not store the whole tree of possible choices.

6 Generation of all non-equivalent comtraces of fixed size

In this section we utilize the fact that the lexicographical normal forms of comtraces over fixed comtrace alphabet $\Theta = (\Sigma, sim, ser)$ are in one to one correspondence with the lexicographical normal forms of traces over a concurrent alphabet $(\widehat{\mathbb{S}}, \widehat{ind})$, where $\widehat{\mathbb{S}} = \{\widehat{a_1}, \widehat{a_2}, \dots, \widehat{a_p}\}$. Moreover, from the language theoretical point of view, the normal forms of those two corresponding objects are given by the same word over the alphabet $\widehat{\mathbb{S}}$. In this section the step sequence that is a lexicographical normal of a comtrace is called *canonical*.

In the presented algorithm we identify and modify only the working suffix. We begin enumeration with the lexicographically minimal step sequence

$$w = A_1A_2\dots A_m = \widehat{a_1}\widehat{a_1}\dots\widehat{a_1} = (a_1)(a_1)\dots(a_1).$$

Then, we consecutively modify the current step sequence to its successor in lexicographic order, skipping all noncanonical ones. The procedure of computing the next canonical step sequence is given in Algorithm 3.

```
Algorithm 3: Next Comtrace
   Input: Canonical step sequence of length $m$: $A_1A_2\dots A_m$
   Output: Canonical step sequence of length $m$: $B_1B_2\dots B_m$ – next in the
           lexicographical order or NULL if $A_1A_2\dots A_m = \widehat{a_p}\widehat{a_p}\dots\widehat{a_p}$

 1 $i := m$;
 2 while $A_i \neq \widehat{a_p}$ do
 3     $i := i - 1$;
 4     if $i = 0$ then
 5         return NULL;
 6 repeat
 7     $A_i :=$ NEXT-INDIV($\Sigma, A_i$);
 8 until ISMINLEX($A_1\dots A_{i-1}, A_i$);
 9 $A_{i+1} := \min(D_{\min}(\{a\})|a \in A_i)$;
10 for $j := i + 2$ to $n$ do
11     $A_j := D_{\min}(A_{j-1})$; // Generate suffix
12 return $B_1B_2\dots B_m = A_1A_2\dots A_m$ ;
```

The presented algorithm consists of three stages:

1. Find the last index i such that $A_i \neq \widehat{a_p}$. Such index is the starting position of the working suffix (lines 2-5).
2. Compute the minimal indivisible step A greater than A_i such that the step sequence $A_1A_2\dots A_{i-1}A$ is canonical (lines 6-8).
3. Generate the rest of the working suffix to obtain the minimal canonical step sequence having $A_1A_2\dots A_{i-1}A$ as a prefix (lines 9-11).

For the efficient generation of the working suffix in the stage three we use a precomputed table $D_{\min}$ defined as follows:

$$\forall_{a\in\Sigma}\ D_{\min}(\{a\}) = \Big\{ \min\{b \in \Sigma \mid (a,b) \notin ind\} \Big\}.$$

Due to Proposition 5, we can assume that all steps of computed working suffix are singletons. The table $D_{\min}$ allows to generate the first element by choosing the minimal singleton step not independent with the step A. The rest of the generation procedure goes by the direct usage of the values from the table $D_{\min}$.

The while-loop (lines 2-5) and the for-loop (lines 10-11) can be implemented in O(m)≤O(n). The number of iterations of the repeat-loop (lines 6-8) is proportional to the size of the step alphabet $|\widehat{\mathbb{S}}| = p$. Therefore, the whole algorithm can be implemented with the time complexity of O(pnk).

Summary

In this paper we discuss some algorithmic issues of combined traces. In particular, we present two algorithms for enumeration, utilizing the notion of the normal forms of a comtrace.

The first algorithm is a procedure that enumerates all step sequences contained in a given comtrace. The main notion used by this algorithm is the self-closed step allowed in the procedure of reconstruction of a step sequence from the projection representation of the considered comtrace.

The second algorithm enumerates all combined traces of a given size over a fixed comtrace alphabet. The main idea of this algorithm is the alphabet change. We use all indivisible steps over the base alphabet of actions.

Both algorithms contain the core subprocedures, which have crucial influence to the efficiency of whole method. The most time consuming part of the first algorithm is the procedure of finding the previous (in lexicographical order) self-closed seed of the maximal allowed step. We proposed the solution based on the local reordering on $\widehat{\mathbb{S}}$. Although it improves the time complexity, we lose the lexicographical order on the generated sequence of comtrace representatives. As a question for future work we consider the problem of preserving the complexity whithout changing the order.

In the second case, the critical subprocedure is the generation of the lexicographically next element over the alphabet $\widehat{\mathbb{S}}$ (i.e. an indivisible step over Θ). Even the precomputation of the array containing all indivisible steps does not reduce the time complexity. The most aspiring future challenge is to find a structural solution similar to the one found for the first problem.

References

1. Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.
2. Ryszard Janicki, Jetty Klein, and Maciej Koutny. Quotient monoids and concurrent behaviours. In Carlos Martín-Vide, editor, *Scientific Applications of Language Methods [5]*, chapter 6, pages 313–386. Imperial College Press, London, 2011.
3. Ryszard Janicki and Maciej Koutny. Semantics of inhibitor nets. *Information and Computation*, 123(1):1–16, 1995.
4. Ryszard Janicki and Dai Tri Man Le. Modelling concurrency with comtraces and generalized comtraces. *Information and Computation*, 209(11):1355–1389, 2011.
5. Carlos Martín-Vide, editor. *Scientific Applications of Language Methods.* Imperial College Press, 2011.
6. Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. Daimi report pb-78, Aarhus University, 1977.
7. Łukasz Mikulski. Projection representation of Mazurkiewicz traces. *Fundamenta Informaticae*, 85:399–408, 2008.
8. Lukasz Mikulski. Algebraic structure of combined traces. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 456–470. Springer, 2012.

9. Carl A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, Bonn, Germany, 1962.
10. Carl A. Petri. Communication with automata. Technical report, Princeton, 1966.
11. Mike W. Shields. Concurrent machines. *The Computer Journal*, 28(5):449–465, 1985.
12. Walter Vogler. A generalization of traces. *ITA*, 25:147–156, 1991.