COMPUTING SCIENCE

Hasse Diagrams of Combined Traces

Lukasz Mikulski and Maciej Koutny

TECHNICAL REPORT SERIES

No. CS-TR-1301 November 2011

TECHNICAL REPORT SERIES

No. CS-TR-1301 November, 2011

Hasse Diagrams of Combined Traces

L. Mikulski, M. Koutny

Abstract

One of the standard ways to represent concurrent behaviours is to use concepts originating from language theory, such as traces and comtraces. Traces can express notions such as concurrency and causality, whereas comtraces can also capture weak causality and simultaneity. This paper is concerned with the development of efficient data structures and algorithms for manipulating comtraces. We introduce Hasse diagrams for comtraces which are a generalisation of Hasse diagrams defined for partial orders and traces, and develop an efficient algorithm for deriving them from language theoretic representations of comtraces. We also explain how the new representation of comtraces can be used to implement efficiently some basic operations on comtraces.

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

Bibliographical details

MIKULSKI, L., KOUTNY, M.

Hasse Diagrams of Combined Traces
[By] L. Mikulski, M. Koutny
Newcastle upon Tyne: Newcastle University: Computing Science, 2011.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1301)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1301

Abstract

One of the standard ways to represent concurrent behaviours is to use concepts originating from language theory, such as traces and comtraces. Traces can express notions such as concurrency and causality, whereas comtraces can also capture weak causality and simultaneity. This paper is concerned with the development of efficient data structures and algorithms for manipulating comtraces. We introduce Hasse diagrams for comtraces which are a generalisation of Hasse diagrams defined for partial orders and traces, and develop an efficient algorithm for deriving them from language theoretic representations of comtraces. We also explain how the new representation of comtraces can be used to implement efficiently some basic operations on comtraces.

About the authors

Lukasz Mikulski is a Junior Lecturer at the Faculty of Mathematics and Computer Science of the Nicolaus Copernicus University, Torun, Poland.

Maciej Koutny obtained his MSc (1982) and PhD (1984) from the Warsaw University of Technology. In 1985 he joined the then Computing Laboratory of the University of Newcastle upon Tyne to work as a Research Associate. In 1986 he became a Lecturer in Computing Science at Newcastle, and in 1994 was promoted to an established Readership at Newcastle. In 2000 he became a Professor of Computing Science.

Suggested keywords

COMTRACE
TRACE
CONCURRENCY
CAUSALITY
HASSE DIAGRAM
STRATIFIED ORDER STRUCTURE
PARTIAL ORDER
INDEPENDENCE, PETRI NET
INHIBITOR ARC
COMPLEXITY

Hasse Diagrams of Combined Traces

Łukasz Mikulski[1] and Maciej Koutny[2]

[1] Faculty of Mathematics and Computer Science
Nicolaus Copernicus University
Toruń, Chopina 12/18, Poland
frodo@mat.umk.pl
[2] School of Computing Science, Newcastle University
Newcastle upon Tyne, NE1 7RU, United Kingdom
maciej.koutny@ncl.ac.uk

Abstract. One of the standard ways to represent concurrent behaviours is to use concepts originating from language theory, such as traces and comtraces. Traces can express notions such as concurrency and causality, whereas comtraces can also capture weak causality and simultaneity. This paper is concerned with the development of efficient data structures and algorithms for manipulating comtraces. We introduce Hasse diagrams for comtraces which are a generalisation of Hasse diagrams defined for partial orders and traces, and develop an efficient algorithm for deriving them from language theoretic representations of comtraces. We also explain how the new representation of comtraces can be used to implement efficiently some basic operations on comtraces.
Keywords: comtrace, trace, concurrency, causality, Hasse diagram, stratified order structure, partial order, independence, Petri net, inhibitor arc, complexity

1 Introduction

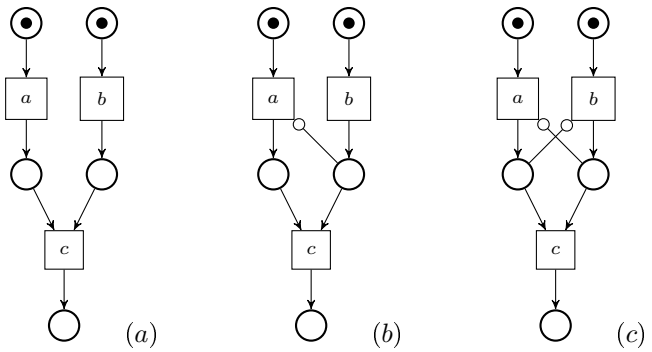
The dynamic behaviours of concurrent systems, e.g., represented by Petri nets, are usually modelled as ongoing evolutions involving actions that take place at the interface with the environment. The simplest representations of such evolutions are sequences (or words) of executed actions, leading to a formal language semantics of Petri nets. However, words alone cannot express concurrency and causality between executed actions which are features of paramount importance if one wants to understand or efficiently analyse concurrent behaviours. To address this issue, one may consider adding an additional information about the relevant properties of behaviours, for example, in the form of causal dependencies between actions. This approach underpins the trace model of concurrent behaviour [1, 9].

Consider, for example, the elementary net system [13] in Example 1(a). It is rather clear that two of its transitions, a and b, are independent (or concurrent) and both cause the execution of the third transition, c. Therefore, the words abc and bac are essentially the same views of a concurrent behaviour in which one first independently executes a and b, and then their execution causes c. This

interpretation is captured by the trace $[abc] = \{abc, bac\}$ which is an equivalence class of the relation capturing the intrinsic similarity of sequential executions of concurrent systems represented by elementary net systems. Moreover, such a trace can be seen as a (causal) partial order with three elements — one for each executed transition — in which a and b are unordered and both precede c.

Traces are not sufficient, however, when one needs to deal with elementary net systems with inhibitor arcs, such as that in Example $1(b)$. In this case, we allow sets of transitions (or steps) to be executed simultaneously. As a result, the net can execute step sequences $\{a\}\{b\}\{c\}$ and $\{a,b\}\{c\}$, but not $\{b\}\{a\}\{c\}$. Another example is shown in Example $1(c)$ where $\{a,b\}\{c\}$ is a possible execution, but neither $\{a\}\{b\}\{c\}$ nor $\{b\}\{a\}\{c\}$ is. To deal with elementary net systems with inhibitor arcs one may extend trace with additional information about intrinsic relationships between executed actions in the form of *weak causality* (where a weakly precedes b if it can be executed earlier or simultaneously with b). Hence, the situation we described for Example $1(b,c)$ can be captured by two equivalence classes of step sequences, called *comtraces*, $[\{a\}\{b\}\{c\}] = \{\{a\}\{b\}\{c\}, \{a,b\}\{c\}\}$ and $[\{\{a,b\}\{c\}] = \{\{a,b\}\{c\}\}$. The resulting model of comtraces [5, 7, 6] enjoys properties similar to that developed for traces. In particular, comtraces can be represented by so-structures which add weak causality to the standard causal partial orders.

Example 1. Three elementary net systems (with inhibitor arcs in case of (b) and (c)) of similar shape.



For system (a) traces are sufficient to capture concurrent behaviour, but for (b) and (c) we need comtraces. □

In this paper, we are concerned with the development of efficient data structures and algorithms for manipulating comtraces. We introduce Hasse diagrams for comtraces which are a generalisation of Hasse diagrams [15] defined for partial orders and traces, and develop an efficient algorithm for deriving them from single step sequence representatives of comtraces. We also explain how the proposed representation of comtraces can be used to implement efficiently some basic operations on comtraces.

The paper is organised as follows. In Section 2, we provide basic notation and terminology. Section 3 recalls a number of notions and notations concerning traces, and sketches an algorithm for deriving Hasse diagram for the partial order induced by a trace. In Section 4 we discuss comtraces and their graph representations (so-structures). This is followed up in Section 5 by an efficient construction of Hasse diagrams from step sequence representatives of comtraces. Section 6 describes some applications of the newly proposed Hasse diagram of an so-structure. Finally, in Section 7, we summarise the contribution of this paper and briefly discuss directions for further research. Proofs of all the results can be found in [11].

2 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 over Σ are called *words*. The sets of all finite words, including the empty word λ, is denoted by Σ^*.

Let $w = a_1 \dots a_n$ and $v = b_1 \dots b_m$ be two words. Then

$$w \circ v = wv = a_1 \dots a_n b_1 \dots b_m$$

is the concatenation of w and v. The alphabet $alph(w)$ of w is the set of all the actions occurring within w, and $\#_a(w)$ is the number of occurrences of an action a within w. The set $occ(w)$ of *action occurrences* of w comprises all pairs (a, i) such that $a \in alph(w)$ and $1 \leq i \leq \#_a(w)$. The *head* (first action occurrences) and *tail* (last action occurrences) of a word w are two sets defined by:

$$\begin{aligned} head(w) &= \{(a, 1) \mid a \in alph(w)\} \\ tail(w) &= \{(a, \#_a(w)) \mid a \in alph(w)\} \ . \end{aligned}$$

Let $\alpha = (a, i)$ be an action occurrence in $occ(w)$. The position $pos_w(\alpha)$ of α within w is the smallest integer j such that $\#_a(a_1 \dots a_j) = i$, and $\ell(\alpha) = a$ is the default *label* of α. We can apply ℓ to sequences and sets of action occurrences in the usual way, i.e,

$$\ell(\alpha_1 \dots \alpha_n) = \ell(\alpha_1) \dots \ell(\alpha_n) \ \text{ and } \ \ell(\{\alpha_1, \dots, \alpha_n\}) = \{\ell(\alpha_1), \dots, \ell(\alpha_n)\} \ .$$

A *poset* is a pair $po = (X, \prec)$, where X is a finite set and $\prec$ is a transitive and irreflexive binary relation on X. In a diagrammatical representation, X is the set of vertices while $\prec$ the set of arcs of po. A pair (X, R), where R is a binary relation on X, is a *po-diagram* if $(X, R^+) = po$. Among all the po-diagrams, we can distinguish the smallest one (i.e., one with the smallest relation R), denoted by $H(po) = (X, \prec^{\mathsf{cov}})$ and called the *Hasse diagram* of po. Note that $\prec^{\mathsf{cov}}$ can be obtained from $\prec$ by simply removing all the arcs implied by the transitivity of $\prec$; in other words, $\prec^{\mathsf{cov}} = \prec \setminus \prec \circ \prec$. Moreover, if (X, R) is a po-diagram, then

$$\prec^{\mathsf{cov}} = R \setminus \bigcup_{i \geq 2} R^i \ .$$

A *linearisation* of a poset $po = (X, \prec)$ is any sequence $u = x_1 \dots x_n$ of distinct elements of po such that $X = \{x_1, \dots, x_n\}$ and, for all $1 \leq i < j \leq n$, $x_j \not\prec x_i$.

3 Traces

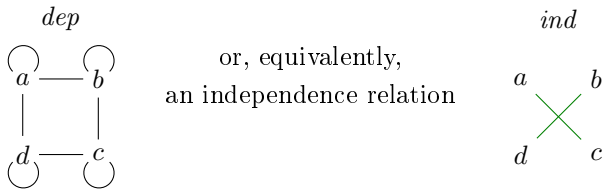
A *concurrent alphabet* is a pair $\Psi = (\Sigma, dep)$, where Σ is an alphabet and $dep \subseteq \Sigma \times \Sigma$ is a reflexive and symmetric *dependence* relation. The corresponding *independence* relation is given by $ind = (\Sigma \times \Sigma) \setminus dep$.

A concurrent alphabet Ψ defines an equivalence relation $\equiv_\Psi$ identifying words which differ only by the ordering of independent actions. Two words, $w, v \in \Sigma^*$, satisfy $w \equiv_\Psi v$ if there exists a finite sequence of commutations of adjacent independent actions transforming w into v. More precisely, $\equiv_\Psi$ is a binary relation over Σ^* which is the reflexive and transitive closure of the relation $\sim_\Psi$ such that $w \sim_\Psi v$ if there are $u, z \in \Sigma^*$ and $(a, b) \in ind$ satisfying $w = uabz$ and $v = ubaz$.

Equivalence classes of $\equiv_\Psi$ are called *(Mazurkiewicz) traces* (see [4, 9, 10]), and the trace containing a given word w is denoted by $[w]$. The set of all traces over Ψ is denoted by $\Sigma^*/_{\equiv_\Psi}$, and the pair $(\Sigma^*/_{\equiv_\Psi}, \circ)$ is a (trace) monoid, where $\tau \circ \tau' = [w \circ w']$, for any words $w \in \tau$ and $w' \in \tau'$, is the concatenation operation for traces. Note that trace concatenation is well-defined as $[w \circ w'] = [v \circ v']$, for all $w, v \in \tau$ and $w', v' \in \tau'$. Similarly, for every trace $\tau = [w]$ and every action $a \in \Sigma$, we can define:

$$\begin{array}{lll} alph(\tau) = alph(w) & occ(\tau) = occ(w) & \#_a(\tau) = \#_a(w) \\ head(\tau) = head(w) & tail(\tau) = tail(w)\ . & \end{array}$$

Example 2. Consider a concurrent alphabet Ψ with four actions $\Sigma = \{a, b, c, d\}$ together with a dependence relation dep given by:



Note that in the example diagrams we will use a_i to denote an action occurrence (a, i), etc.
Then $w = abbaacd \equiv_\Psi abbcaad$. The set of action occurrences of w is:

$$\begin{aligned} occ(w) &= \{(a,1),(a,2),(a,3),(b,1),(b,2),(c,1),(d,1)\} \\ &= \{a_1, a_2, a_3, b_1, b_2, c_1, d_1\}\ . \end{aligned}$$

Its head is $\{a_1, b_1, c_1, d_1\}$ and tail $\{a_3, b_2, c_1, d_1\}$. □

A word $w \in \Sigma^*$ is in *(Foata) canonical form* (see [3]) w.r.t. the dependence relation dep and a lexicographic order lex on Σ, if $w = w_1 \dots w_n$ $(n \geq 0)$, where each w_i is a nonempty word such that:

- w_i is fully commutative and minimal w.r.t. *lex* among all sequences $u \in \Sigma^*$ satisfying $occ(w_i) = occ(u)$; and
- for each $i > 2$ and a occurring in w_i, there is b occurring in w_{i-1} such that $(a, b) \in dep$.

Each trace contains exactly one sequence in canonical form, called its *canonical representation*.

One can represent a trace τ as a poset of action occurrences. More precisely, $po(\tau) = (occ(w), \prec_w^+)$ is the poset *induced* by τ, where w is any word belonging to τ, and $\prec_w$ is a binary relation on $occ(w)$ such that $\alpha \prec_w \beta$ if $pos_w(\alpha) < pos_w(\beta)$ and $(\ell(\alpha), \ell(\beta)) \in dep$. The soundness of this definition stems from the following:

- $\tau = \{\ell(u) \mid u \text{ is a linearisation of } po(\tau)\}$; and
- $(occ(w), \prec_w^+) = (occ(v), \prec_v^+)$, for all $w, v \in \tau$.

Note that $\prec_w = \prec_v$, for all words $w, v \in \tau$.

The *Hasse diagram* of τ, denoted by $H(\tau)$ (or $H(w)$, for any $w \in \tau$), is then simply the Hasse diagram of $po(\tau)$. Note that each arc of $H(w)$ belongs to $\prec_w$, but not necessarily vice versa.

3.1 Direct construction of Hasse diagrams for traces

We now sketch the idea behind an efficient way of generating Hasse diagrams for traces. The procedure takes an arbitrary word belonging to a trace together with the dependence relation.

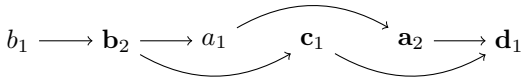
The on-line algorithm of [12] scans the input word from left to right, and is based on an observation that when adding a new vertex to an already constructed Hasse diagram $H(v) = (occ(v), \prec_v^{\mathsf{h}})$ in order to generate $H(va) = (occ(va), \prec_{va}^{\mathsf{h}})$, it suffices to consider only $tail(v)$, i.e., the latest occurrences of actions in v. To use the relevant relationships involving such occurrences, for every action a appearing in the input word, the algorithm maintains:

- ORD_a which is the list of all actions dependent with a which have been seen so far, in the reversed order of appearance of their latest occurrences; and
- VS_a which is the set of all actions (called *sources*) whose latest occurrences are connected to the latest occurrence of a (if the latter exists), i.e., $b \in VS_a$ if there is a path from $(b, \#_b(v))$ to $(a, \#_a(v))$ in $H(v)$.

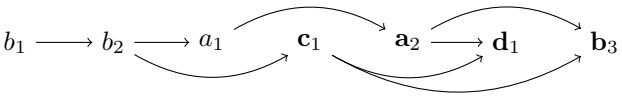
When constructing $H(va)$ from an already built $H(v)$, the algorithm adds new arcs from $tail(v)$ to $(a, \#_a(v) + 1)$, and dynamically computes a new set of sources VS_a^{new} of a (initially, $VS = \varnothing$). The algorithm scans the list ORD_a and, for each b which is not a source of any of the sources of a (i.e., $b \notin \bigcup_{c \in VS} VS_c$), generates a new arc from $(b, \#_b(v))$ to $(a, \#_a(v) + 1)$ and sets VS to $VS \cup VS_b$. Moreover, after each step the data structure that describes the tail is updated; in particular, each VS_b with $b \neq a$ is set to $VS_b \setminus \{a\}$ (see Example 3). The correctness of whole procedure is guaranteed by respecting the generation order of the latest occurrences of actions stored in ORD_a.

The algorithm has time complexity of $O(nk^2)$, where n is the number of action occurrences in the input word, and k is the number of different actions appearing in the input word. Typically, k (bounded by $|\Sigma|$) is much smaller than n and in any case it is fixed for a given concurrent system (e.g., a Petri net). For practical applications the above algorithm can therefore be considered as linear in the size of its input, and so optimal as any algorithm generating the Hasse diagram of a trace must look at least once at each of its action occurrences.

Example 3. Consider two words, $v = bbacad$ and $vb = bbacadb$, over the concurrent alphabet from Example 2. Below we show the diagrams of the corresponding Hasse diagrams, indicating in bold the action occurrences belonging to $tail(v)$ and $tail(vb)$, respectively, together with auxiliary tail data structures.



	ORD	*VS*
a	(d, a, b)	$\{a, b\}$
b	(a, c, b)	$\{b\}$
c	(d, c, b)	$\{c, b\}$
d	(d, a, c)	$\{a, b, c, d\}$



	ORD	*VS*
a	(b, d, a)	$\{a\}$
b	(b, a, c)	$\{a, b, c\}$
c	(b, d, c)	$\{c\}$
d	(d, a, c)	$\{a, c, d\}$

□

4 Comtraces

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 in such a case a may also occur before b (with both executions being equivalent). The set of all (potential) steps over Θ, or *step alphabet*, is then defined as the set $\mathbb{S}$ comprising all nonempty sets of actions $A \subseteq \Sigma$ such that $(a, b) \in sim$, for all distinct $a, b \in A$. Finite sequences in $\mathbb{S}^*$, including the empty one denoted by λ, are called *step sequences*.

We will now 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 $w = A_1 \dots A_n$ and $v = B_1 \dots B_m$ be two step sequences. Then $w \circ v = wv = A_1 \dots A_n B_1 \dots B_m$ is the concatenation of w and v. The alphabet $alph(w)$ of w comprises all action occurring within w, and $\#_a(w)$ is the number of occurrences of an action a within w. The set $occ(w)$ of action occurrences of w comprises all pairs (a, i) with $a \in alph(w)$ and $1 \leq i \leq \#_a(w)$. The position $pos_w(\alpha)$ of an action occurrence $\alpha = (a, i) \in occ(w)$ is given as the smallest integer j such that $\#_a(A_1 \dots A_j) = i$. In such a case, we also denote $\alpha \in occ_j(w)$. Hence the sets $occ_1(w), \dots, occ_n(w)$ form a partition of $occ(w)$. We can also apply the default labelling ℓ to (sequences of) sets of action occurrences in the usual way. The head and tail of w are given by:

$$\begin{aligned} head(w) &= \{(a, 1) \mid a \in alph(w)\} \\ tail(w) &= \{(a, \#_a(w)) \mid a \in alph(w)\} \ . \end{aligned}$$

The *comtrace congruence* over Θ, denoted by $\equiv_\Theta$, is the reflexive, symmetric and transitive closure of the relation $\sim_\Theta \subseteq \mathbb{S}^* \times \mathbb{S}^*$ such that $w \sim_\Theta v$ if there are $u, z \in \mathbb{S}^*$ 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 *comtraces* (see [6]), and the comtrace containing a given step sequence w is denoted by $[w]$. The set of all comtraces is denoted by $\mathbb{S}^*/_\equiv$, and the pair $(\mathbb{S}^*/_\equiv, \circ)$ 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 $w', v' \in \tau'$. A comtrace τ is a prefix of a comtrace τ' if there is a comtrace τ'' such that $\tau \circ \tau'' = \tau'$. As in the case of traces, for every comtrace τ and every $a \in \Sigma$, we can define $alph(\tau) = alph(w)$, $\#_a(\tau) = \#_a(w)$, $occ(\tau) = occ(w)$, $head(\tau) = head(w)$ and $tail(\tau) = tail(w)$, where w is any step sequence belonging to τ.

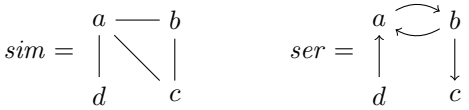
Next, we give the canonical form of a comtrace which essentially captures a greedy, maximally concurrent, execution of the actions occurring in the comtrace conforming to the simultaneity and serialisability relations. A step sequence $w = A_1 \dots A_n \in \mathbb{S}^*$ is in (Foata) *canonical form* if, for each $i \leq n$, whenever $Av \equiv_\Theta A_i \dots A_k$ for some $A \in \mathbb{S}$ and $v \in \mathbb{S}^*$, then $|A| \leq |A_i|$. One can see that each comtrace comprises a unique step sequence in normal form. Note that an alternative (equivalent) definition of normal form requires that, for every $i < k$, there is no $\varnothing \neq A \subseteq A_{i+1}$ such that $A_i \times A \subseteq ser$ and $A \times (A_{i+1} \backslash A) \subseteq ser$.

In technical discussion, we will use four relations covering all possible relationships between individual actions:

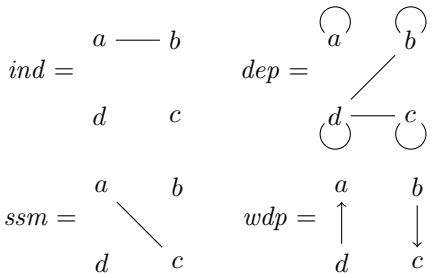
- Dependence $dep = (\Sigma \times \Sigma) \setminus sim$, and independence $ind = ser \cap ser^{-1}$.
  Both relations have their counterparts in trace theory, and so we denote them in the same way. If two actions are dependent then their two occurrences must happen in the same order (and never simultaneously)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

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

Example 4. 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:



Then the four derived relations on actions are as follows:



Then $[w] = [u]$, where:

$$w = \{d\}\{a, b\}\{c\}\{d\}\{a, b, c\}\{c\}$$
$$u = \{a, d\}\{b, c\}\{d\}\{a, b, c\}\{c\}\,.$$

Moreover, u is a step sequence in normal form. □

4.1 Stratified order structures

Comtraces can be represented by so-structures, in a similar way as traces can be represented by posets.

A *stratified order structure* (or so-structure) is a tuple $sos = (X, \prec, \sqsubset)$ comprising two binary relations, $\prec$ (*causality*) and $\sqsubset$ (*weak causality*), on a finite set X such that, for all $x, y, z \in X$:

$$S1: \ x \not\sqsubset x \qquad S3: \ x \sqsubset y \sqsubset z \ \wedge \ x \neq z \implies x \sqsubset z$$
$$S2: \ x \prec y \implies x \sqsubset y \qquad S4: \ x \sqsubset y \prec z \ \vee \ x \prec y \sqsubset z \implies x \prec z\,.$$

Intuitively, $\prec$ represents the 'earlier than' relationship in X, and $\sqsubset$ the 'not later than' relationship. Note that $\prec$ is a partial order, and $x \prec y$ implies $y \not\sqsubset x$. In a diagrammatical representation, $\sqsubset$ is represented by dashed arcs.

The *so-closure* of a triple $\varrho = (X, \prec^{\mathsf{pre}}, \sqsubset^{\mathsf{pre}})$, where X is a finite set and $\prec^{\mathsf{pre}}, \sqsubset^{\mathsf{pre}}$ are binary relations on X is defined as $\varrho^{\diamondsuit} = (X, \gamma \circ \prec^{\mathsf{pre}} \circ \gamma, \gamma \setminus id_X)$, where $\gamma = (\prec^{\mathsf{pre}} \cup \sqsubset^{\mathsf{pre}})^*$. If $\varrho^{\diamondsuit} = sos$, where sos is an so-structure, then ρ is called an *sos-diagram*.

A *stratification* of an so-structure $sos = (X, \prec, \sqsubset)$ is any sequence $u = X_1 \ldots X_n$ of nonempty disjoint subsets of X such that $X = X_1 \cup \ldots \cup X_n$,

- $(X_j \times X_i) \cap \prec = \varnothing$, for all $1 \leq i \leq j \leq n$; and
- $(X_j \times X_i) \cap \sqsubset = \varnothing$, for all $1 \leq i < j \leq n$.

The so-structure *induced* by a comtrace τ is defined as

$$sos(\tau) = (occ(w), \prec_w, \sqsubset_w)^{\diamondsuit} ,$$

where w is any step sequence $w \in \tau$, and:

$$\begin{aligned} \prec_w &= \{(\alpha, \beta) \in occ(w) \times occ(w) \mid pos_w(\alpha) < pos_w(\beta) \wedge (\ell(\alpha), \ell(\beta)) \notin ser\} \\ \sqsubset_w &= \{(\alpha, \beta) \in occ(w) \times occ(w) \mid pos_w(\alpha) \leq pos_w(\beta) \wedge (\ell(\beta), \ell(\alpha)) \notin ser\} . \end{aligned}$$

Note that $\prec_w = \prec_v$ and $\sqsubset_w = \sqsubset_v$, for all $w, v \in \tau$.

The soundness of the last definition stems from the following:

- $\tau = \{\ell(v) \mid v \text{ is a stratification of } sos(\tau)\}$, and
- $(occ(w), \prec_w, \sqsubset_w)^{\diamondsuit} = (occ(v), \prec_v, \sqsubset_v)^{\diamondsuit}$, for all step sequences $w, v \in \tau$.

4.2 Folded so-structures

Weak causality is a pre-order rather than a partial order relation, and it can be advantageous to work with a quotient so-structure derived from $sos(\tau) = (occ(\tau), \prec, \sqsubset)$ induced by a comtrace τ. First, for each action occurrence $\alpha \in occ(\tau)$, we denote by $\langle \alpha \rangle$ the equivalence class of the $\sqsubset$-cycle relation comprising α, i.e., α together with the set of all $\beta \in occ(\tau)$ satisfying $\alpha \sqsubset \beta \sqsubset \alpha$. Each such $\Delta = \langle \alpha \rangle$ will be called a *folded action* and their set denoted by $\widehat{occ}(\tau)$. Then the *folded so-structure induced* by a comtrace τ is defined as $\widehat{sos}(\tau) = (\widehat{occ}(\tau), \widehat{\prec}, \widehat{\sqsubset})$, where, for all $\Delta, \Delta' \in \widehat{occ}(\tau)$:

- $\Delta \widehat{\prec} \Delta'$ if $(\Delta \times \Delta') \cap \prec \neq \varnothing$; and
- $\Delta \widehat{\sqsubset} \Delta'$ if $(\Delta \times \Delta') \cap \sqsubset \neq \varnothing$ and $\Delta \neq \Delta'$.

Note that, by *S4* and $\Delta \widehat{\prec} \Delta'$, $\Delta \times \Delta'$ is included in $\prec$, and, by *S3* and $\Delta \widehat{\sqsubset} \Delta'$, $\Delta \times \Delta'$ is included in $\sqsubset$.

By *S2–S4*, $\widehat{sos}(\tau)$ is an so-structure, and $\widehat{\sqsubset}$ is a poset containing $\widehat{\prec}$. Moreover, different comtraces induce different folded so-structures, and there is a straightforward way of recovering $sos(\tau)$ from $\widehat{sos}(\tau)$ as we have:

- $\alpha \prec \beta$ iff $\langle \alpha \rangle \widehat{\prec} \langle \beta \rangle$, and
- $\alpha \sqsubset \beta$ iff $\langle \alpha \rangle \widehat{\sqsubset} \langle \beta \rangle$, or $\alpha \neq \beta \wedge \langle \alpha \rangle = \langle \beta \rangle$.

It turns out that action occurrences occurring in a single step sequence can be partitioned into folded actions.

Proposition 5. *Let $w = A_1 \ldots A_n$ be a step sequence, $i \leq n$, and SCC be the set of strongly connected components of the directed graph*

$$(occ_i(w), \sqsubset_w |_{occ_i(w) \times occ_i(w)}) .$$

Then SCC is a set of folded actions partitioning $occ_i(w)$, and $\widehat{\sqsubset}|_{SCC \times SCC}$ is an acyclic relation. Moreover, if we take any linearisation $(TS_1, \ldots, TS_m)$ of SCC then each comtrace $[A_1 \ldots A_{i-1} TS_1 \ldots TS_j]$, for $j \leq m$, is a prefix of $[w]$.

Proof. We observe that, for all $\alpha \in occ_i(w)$ and $\beta \in occ(w)$, if $\alpha \sqsubset_w^+ \beta \sqsubset_w^+ \alpha$ then $\langle \alpha \rangle \subseteq occ_i(w)$. Hence each strongly connected component in SCC is a folded action. The second and third parts of the result follow directly from the definitions. □

Folded so-structures can also be used to recover in a direct way all the step sequences belonging to a given comtrace.

Proposition 6. *Let τ be a comtrace.*
Then we have that $\tau = \{\widehat{\ell}(v) \mid v \text{ is a stratification of } \widehat{sos}(\tau)\}$, where:

$$\widehat{\ell}(v) = \left(\bigcup_{\Delta \in \Gamma_1} \ell(\Delta) \right) \ldots \left(\bigcup_{\Delta \in \Gamma_k} \ell(\Delta) \right) ,$$

for every stratification $v = \Gamma_1 \ldots \Gamma_k$ of $\widehat{sos}(\tau)$.

Proof. ($\supseteq$) Let $v = \Gamma_1 \ldots \Gamma_k$ be a stratification of $\widehat{sos}(\tau)$:

$$w = A_1 \ldots A_k = \left(\bigcup_{\Delta \in \Gamma_1} \Delta \right) \ldots \left(\bigcup_{\Delta \in \Gamma_k} \Delta \right) .$$

We have that $\Gamma_1, \ldots, \Gamma_k$ are non-empty disjoint sets of folded actions such that $\widehat{occ}(\tau) = \Gamma_1 \cup \ldots \cup \Gamma_k$,

- $(\Gamma_j \times \Gamma_i) \cap \widehat{\prec} = \varnothing$, for all $1 \leq i \leq j \leq k$; and
- $(\Gamma_j \times \Gamma_i) \cap \widehat{\sqsubset} = \varnothing$, for all $1 \leq i < j \leq k$.

Moreover, each folded action occurring in v is an equivalence class, and so different folded actions occurring in v are disjoint sets. Hence $A_1, \ldots, A_k$ are non-empty disjoint sets of actions occurrences such that $occ(\tau) = A_1 \cup \ldots \cup A_k$,

- $(A_j \times A_i) \cap \prec = \varnothing$, for all $1 \leq i \leq j \leq k$; and
- $(A_j \times A_i) \cap \sqsubset = \varnothing$, for all $1 \leq i < j \leq k$.

As a result, w is a stratification of $sos(\tau)$ satisfying $\widehat{\ell}(v) = \ell(w)$. Moreover, we have $\tau = \{\ell(w) \mid w \text{ is a stratification of } sos(\tau)\}$. Hence $\widehat{\ell}(v) \in \tau$.

($\subseteq$) To show the reverse inclusion, assume that $w = A_1 \cup \ldots \cup A_k$ is a stratification of $sos(\tau)$. Then each A_i can be partitioned into a set of folded actions Γ_i (see Proposition 5). One can then see that $v = \Gamma_1 \ldots \Gamma_k$ is a stratification of $\widehat{sos}(\tau)$ satisfying $\widehat{\ell}(v) = \ell(w)$. The inclusion then follows from $\tau = \{\ell(w) \mid w \text{ is a stratification of } sos(\tau)\}$. □

The next result shows that folded actions can be derived directly from the step sequences forming a comtrace.

Proposition 7. *Let α and β be two action occurrences of a comtrace τ. Then $\langle\alpha\rangle = \langle\beta\rangle$ iff $pos_w(\alpha) = pos_w(\beta)$, for every step sequence $w \in \tau$.*

Proof. ($\Longrightarrow$) $\langle\alpha\rangle = \langle\beta\rangle$ implies that, for every stratification $v = X_1 \dots X_n$ of $sos(\tau)$, α and β belong to the same set X_i. Hence α and β occur in the same set $\ell(X_i)$ in $\ell(v)$. Thus, by $\tau = \{\ell(v) \mid v \text{ is a stratification of } sos(\tau)\}$, $pos_w(\alpha) = pos_w(\beta)$, for every step sequence $w \in \tau$.
($\Longleftarrow$) Let $sos(\tau) = (occ(\tau), \prec, \sqsubset)$. The implication follows from the fact that if $\alpha \not\sqsubset \beta$ then there is a step sequence $w \in \tau$ such that $pos_w(\alpha) > pos_w(\beta)$. □

4.3 Hasse diagrams of comtraces

As a folded so-structure $\widehat{sos}(\tau) = (\widehat{occ}(\tau), \widehat{\prec}, \widehat{\sqsubset})$ comprises two nested posets, defining the *folded Hasse diagram* of a comtrace τ is straightforward:

$$\widehat{H}(\tau) = (\widehat{occ}(\tau), \widehat{\prec}^{\mathsf{h}}, \widehat{\sqsubset}^{\mathsf{h}}) \ ,$$

where:

$$\begin{aligned}\widehat{\prec}^{\mathsf{h}} &= \widehat{\prec} \setminus ((\widehat{\prec} \circ \widehat{\prec}) \cup (\widehat{\prec} \circ \widehat{\sqsubset}) \cup (\widehat{\sqsubset} \circ \widehat{\prec}))\\ \widehat{\sqsubset}^{\mathsf{h}} &= (\widehat{\sqsubset} \setminus (\widehat{\sqsubset} \circ \widehat{\sqsubset})) \setminus \widehat{\prec}^{\mathsf{h}} \ .\end{aligned}$$

In particular, one can see that $\widehat{H}(\tau)$ is the smallest triple $(\widehat{occ}(\tau), \widehat{\prec}^{\mathsf{pre}}, \widehat{\sqsubset}^{\mathsf{pre}})$ whose so-closure yields $\widehat{sos}(\tau)$; in other words, $\widehat{H}(\tau)$ is the smallest $\widehat{sos}(\tau)$-*diagram*. Note that if $(\widehat{occ}(\tau), \widehat{\prec}^{\mathsf{pre}}, \widehat{\sqsubset}^{\mathsf{pre}})$ is an $\widehat{sos}(\tau)$-diagram and $\widehat{\sqsubset}^{\mathsf{pre}}$ is irreflexive, then $\widehat{\prec}^{\mathsf{h}}$ and $\widehat{\sqsubset}^{\mathsf{h}}$ are respectively included in $\prec^{\mathsf{pre}}$ and $\sqsubset^{\mathsf{pre}} \setminus \prec^{\mathsf{pre}}$.

If Δ and Δ' are two distinct vertices of an $\widehat{sos}(\tau)$-diagram $(\widehat{occ}(\tau), \widehat{\prec}^{\mathsf{pre}}, \widehat{\sqsubset}^{\mathsf{pre}})$, then we say that:

- there is a *strong path* from Δ to Δ' if there are $\Delta = \Delta_1, \dots, \Delta_n = \Delta'$ such that, for every $i < n$, $\Delta_i \widehat{\prec}^{\mathsf{pre}} \Delta_{i+1}$ or $\Delta_i \widehat{\sqsubset}^{\mathsf{pre}} \Delta_{i+1}$; moreover, $\Delta_j \widehat{\prec}^{\mathsf{pre}} \Delta_{j+1}$ for at least one $j < n$.
- there is a *weak path* from Δ to Δ' if there are $\Delta = \Delta_1, \dots, \Delta_n = \Delta'$ such that, for every $i < n$, $\Delta_i \widehat{\sqsubset}^{\mathsf{pre}} \Delta_{i+1}$; moreover, there is no strong path from Δ to Δ'.

Note that if there is a weak path from Δ to Δ' then there is no weak path from Δ' to Δ.

As far as the original so-structure induced by τ is concerned, the *Hasse diagram* of τ is defined as:

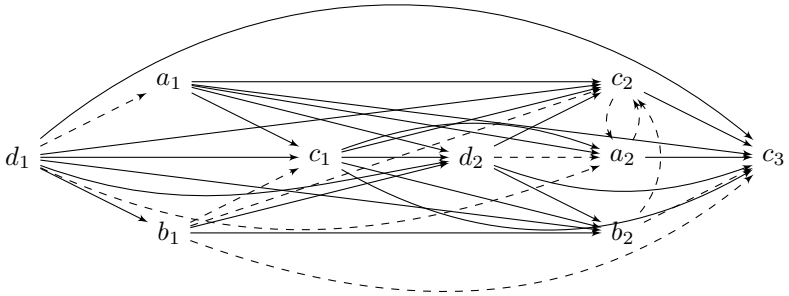
$$H(\tau) = (occ(\tau), \prec^{\mathsf{h}}, \sqsubset^{\mathsf{h}}) \ ,$$

where $\alpha \prec^{\mathsf{h}} \beta$ if $\langle\alpha\rangle \widehat{\prec}^{\mathsf{h}} \langle\beta\rangle$, and $\alpha \sqsubset^{\mathsf{h}} \beta$ if $\langle\alpha\rangle \widehat{\sqsubset}^{\mathsf{h}} \langle\beta\rangle$ or $\alpha \neq \beta \wedge \langle\alpha\rangle = \langle\beta\rangle$. Note that $H(\tau)$ is not guaranteed to be the smallest $sos(\tau)$-diagram because,

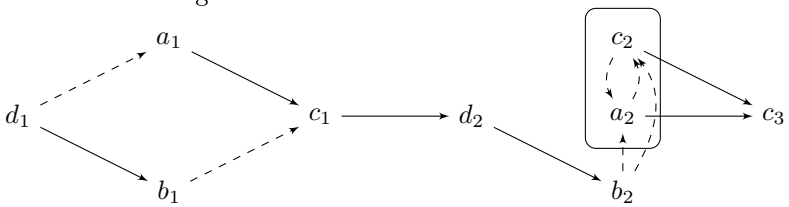
in general, a minimal $sos(\tau)$-diagram does not exist as weak causality is only a pre-order.

Henceforth we will work with two different yet equivalent representations of comtraces, as well as two different kinds of Hasse diagrams, always choosing that which is more convenient for technical considerations.

Example 8. The following is the $sos(w)$-diagram $(occ(w), \prec_w, \sqsubset_w)$ for the comtrace $[w]$ from Example 4:



and its Hasse diagram is:



Note that $\{a_2, c_2\}$ indicated by a border in the above diagram is the only non-singleton folded action. □

□

We now provide some general properties of Hasse diagrams of comtraces. In what follows, for every step sequence w in a comtrace τ, we denote $\widehat{H}(w) = \widehat{H}(\tau)$ and $H(w) = H(\tau)$.

Proposition 9. *Let τ and τ' be two comtraces. Then the (folded) Hasse diagrams of τ and τ' are vertex-induced subgraphs of the (folded) Hasse diagram of $\tau \circ \tau'$, assuming that in ($\widehat{H}(\tau')$) $H(\tau')$ each action occurrence (a, i) has been replaced by $(a, i + \#_a(\tau))$.*

Proof. Follows from the definitions. We only observe that each folded action of $\widehat{H}(\tau \circ \tau')$ is either a folded action in $\widehat{H}(\tau)$, or a a folded action in $\widehat{H}(\tau')$ with each action occurrence (a, i) replaced by $(a, i + \#_a(\tau))$ (see Proposition 7). □

Corollary 10. *For every step sequence wA, the vertices of $\widehat{H}(w)$ and $H(w)$ are also vertices of $\widehat{H}(wA)$ and $H(wA)$, respectively.*

Proposition 11. *Let w be a step sequence, $(\langle(a,i)\rangle, \langle\beta\rangle)$ be an arc in $\widehat{H}(w)$, $(a, \ell(\beta)) \notin ind$, and $\#_a(w) > i$. Then $pos_w(a, i+1) \geq pos_w(\beta)$.*

Proof. Suppose that $pos_w(a,i+1) < pos_w(\beta)$ which implies $pos_w(a,i) < pos_w(\beta)$. Then, by Proposition 7, $\langle(a,i+1)\rangle \neq \langle\beta\rangle$. We also have $(a,i+1) \prec_w \beta$ or $(a,i+1) \sqsubset_w \beta$, and so there is a path from $\langle(a,i+1)\rangle$ to $\langle\beta\rangle$ in $\widehat{H}(w)$. We then observe that $(a,i) \prec_w (a,i+1)$ and $\langle(a,i)\rangle \neq \langle(a,i+1)\rangle$, and so there is a strong path from $\langle(a,i)\rangle$ to $\langle(a,i+1)\rangle$ in $\widehat{H}(w)$. Hence there is no arc $(\langle(a,i)\rangle, \langle\beta\rangle)$ in $\widehat{H}(w)$, yielding a contradiction. □

Proposition 12. *Let w be a step sequence. Moreover, let Δ and Δ' be two distinct vertices of $\widehat{H}(w) = (\widehat{occ}, \widehat{\prec}^{\mathsf{h}}, \widehat{\sqsubset}^{\mathsf{h}})$.*

1. *If $(\Delta, \Delta') \in \widehat{\prec}^{\mathsf{h}}$, then there are $\alpha \in \Delta$ and $\alpha' \in \Delta'$ such that $\alpha \prec_w \alpha'$.*
2. *If $(\Delta, \Delta') \in \widehat{\sqsubset}^{\mathsf{h}}$, then there are $\alpha \in \Delta$ and $\alpha' \in \Delta'$ such that $\alpha \sqsubset_w \alpha'$, and there are no $\beta \in \Delta$ and $\beta' \in \Delta'$ such that $\beta \prec_w \beta'$.*
3. *If there are $\alpha \in \Delta$ and $\alpha' \in \Delta'$ such that $\alpha \prec_w \alpha'$, then the following statements are equivalent:*
   - *$(\Delta, \Delta') \notin \widehat{\prec}^{\mathsf{h}}$.*
   - *In $\widehat{H}(w)$, there is a vertex Δ'' different from Δ and Δ' such that there is a weak or strong path from Δ to Δ'', and a weak or strong path from Δ'' to Δ'; moreover, at least one of these paths is strong.*
4. *If there are $\alpha \in \Delta$ and $\alpha' \in \Delta'$ such that $\alpha \sqsubset_w \alpha'$ and there are no $\beta \in \Delta$ and $\beta' \in \Delta'$ such that $\beta \prec_w \beta'$, then the following statements are equivalent:*
   - *$(\Delta, \Delta') \notin \widehat{\sqsubset}^{\mathsf{h}}$.*
   - *In $\widehat{H}(w)$, there is a vertex Δ'' different from Δ and Δ' such that there is a weak or strong path from Δ to Δ'', and a weak or strong path from Δ'' to Δ'.*

Proof. (1) and (2) follow directly from the definitions. We only observe that if $\alpha \in \Delta$ and $\alpha' \in \Delta'$ are such that $\alpha \prec_w \alpha'$ then $(\Delta, \Delta') \in \widehat{\prec}$, and if $\alpha \in \Delta$ and $\alpha' \in \Delta'$ are such that $\alpha \sqsubset_w \alpha'$ then $(\Delta, \Delta') \in \widehat{\sqsubset}$.

(3) Suppose that $\alpha \in \Delta$, $\alpha' \in \Delta'$ and $\alpha \prec_w \alpha'$. If $(\Delta, \Delta') \notin \widehat{\prec}^{h}$ then, directly from the definition,

$$(\Delta, \Delta') \in (\widehat{\prec} \circ \widehat{\prec}) \cup (\widehat{\prec} \circ \widehat{\sqsubset}) \cup (\widehat{\sqsubset} \circ \widehat{\prec}) \ ,$$

and so there exists Δ'' such that $\Delta \widehat{\prec} \Delta'' \widehat{\prec} \Delta'$ or $\Delta \widehat{\prec} \Delta'' \widehat{\sqsubset} \Delta'$ or $\Delta \widehat{\sqsubset} \Delta'' \widehat{\prec} \Delta'$. As $\widehat{\prec}$ is a subset of $\widehat{\sqsubset}$, we have $\Delta \widehat{\sqsubset} \Delta'' \widehat{\sqsubset} \Delta'$ and $\Delta \widehat{\prec} \Delta'' \vee \Delta'' \widehat{\prec} \Delta'$, so there is a weak or strong path from Δ to Δ'', and a weak or strong path from Δ'' to Δ', and at least one of these paths is strong.

To show the reverse implication, suppose that there is a vertex Δ'' different from Δ and Δ' such that there is a weak or strong path from Δ to Δ'', and a weak or strong path from Δ'' to Δ', and at least one of these paths is strong. Then $(\Delta, \Delta') \in (\widehat{\prec} \circ \widehat{\prec}) \cup (\widehat{\prec} \circ \widehat{\sqsubset}) \cup (\widehat{\sqsubset} \circ \widehat{\prec})$, and so $(\Delta, \Delta') \notin \widehat{\prec}^{h}$.

(4) Similar to the proof of (3). □

Corollary 13. *For every step sequence wA, $\widehat{H}(w)$ and $H(w)$ are vertex-induced subgraphs of $\widehat{H}(wA)$ and $H(wA)$, respectively.*

Proposition 14. *Let wA be a step sequence and:*

$$\widehat{H}(w) = (\widehat{occ}(w), \widehat{\prec}^{\mathsf{h}}_w, \widehat{\sqsubset}^{\mathsf{h}}_w) \quad \textit{and} \quad \widehat{H}(wA) = (\widehat{occ}(wA), \widehat{\prec}^{\mathsf{h}}_{wA}, \widehat{\sqsubset}^{\mathsf{h}}_{wA})$$

be the folded Hasse diagrams of w and wA, respectively. Then:

- $\widehat{\prec}^{\mathsf{h}}_{wA} \setminus \widehat{\prec}^{\mathsf{h}}_w \subseteq \widehat{tail}(w) \times (\widehat{occ}(wA) \setminus \widehat{occ}(w))$, *and*
- $\widehat{\sqsubset}^{\mathsf{h}}_{wA} \setminus \widehat{\sqsubset}^{\mathsf{h}}_w \subseteq (\widehat{tail}(w) \cup (\widehat{occ}(wA) \setminus \widehat{occ}(w))) \times (\widehat{occ}(wA) \setminus \widehat{occ}(w))$,

where $\widehat{tail}(w) = \{\Delta \in \widehat{occ}(w) \mid \Delta \cap tail(w) \neq \varnothing\}$.

Proof. Follows directly from the definitions. We only observe that if $(\Delta, \Delta') \in \widehat{\prec}^{\mathsf{h}}_{wA}$ then, by Proposition 12, there are $\alpha \in \Delta$ and $\alpha' \in \Delta'$ such that $\alpha \prec_w \alpha'$. Moreover, if $(\Delta, \Delta') \notin \widehat{\prec}^{\mathsf{h}}_w$ then $\alpha' \in occ_{|wA|}(wA)$. Hence, if $\alpha \neq (\ell(\alpha), \#_{\ell(\alpha)}(wA))$ then we obtain a contradiction with Proposition 11.
The case $(\Delta, \Delta') \in \widehat{\sqsubset}^{\mathsf{h}}_{wA} \setminus \widehat{\sqsubset}^{\mathsf{h}}_w$ is similar. □

4.4 Constructing comtrace graphs

Given a step sequence w with with $|occ(w)| = n$, perhaps the easiest way of constructing the graph of the Hasse diagram $H(w) = H(\tau) = (occ(w), \prec^{\mathsf{h}}, \sqsubset^{\mathsf{h}})$ induced by the comtrace $\tau = [w]$ is to simply follow the definitions. First, we take the set of action occurrences $occ(w)$ to be the vertices of $H(w)$, and process one-by-one all possible pairs of distinct vertices. For each such pair (α, β), we check whether $pos_w(\alpha) < pos_w(\beta)$ and $(\ell(\alpha), \ell(\beta)) \notin ser$ (which yields $\alpha \prec_w \beta$), or whether $pos_w(\alpha) \leq pos_w(\beta)$ and $(\ell(\beta), \ell(\alpha)) \notin ser$ (which yields $\alpha \sqsubset_w \beta$). This can be done in $O(n^2)$ time, and the resulting graph $(occ(w), \prec_w, \sqsubset_w)$ has the size $O(n^2)$. Example 8 shows the result of carrying such checks for the comtrace $[w]$ of Example 4.

Next, we apply the so-closure to get $sos(w) = (occ(w), \prec, \sqsubset)$, and then remove all the arcs implied by *S2–S4*, in order to generate $H(w)$. Applying the so-closure operation is a straightforward generalisation of the transitive closure of a binary relation. Removing unnecessary arcs, however, is not so. First, we temporarily remove all the arcs joining the action occurrences belonging to the same folded action, obtaining $\sqsubset' = \sqsubset \setminus (\sqsubset \cap \sqsubset^{-1})$. Then, we delete from $\prec$ all the arcs (α, β) for which there exists δ satisfying $\alpha \prec \delta \prec \beta$ or $\alpha \sqsubset' \delta \prec \beta$ or $\alpha \prec \delta \sqsubset' \beta$, obtaining $\prec^{\mathsf{h}}$. Then we delete from $\sqsubset'$ all the arcs (α, β) belonging to $\prec^{\mathsf{h}}$, and those for which there exists δ satisfying $\alpha \sqsubset' \delta \sqsubset' \beta$, obtaining $\sqsubset''$. Finally, $\sqsubset^{\mathsf{h}}$ is calculated as $\sqsubset'' \cup (\sqsubset \cap \sqsubset^{-1})$.

As far as complexity is concerned, as we computed $(occ(w), \prec_w, \sqsubset_w)$, the overall complexity of the algorithm is at least linear in terms of n. In the rest of the paper, we will provide much better solution that takes advantage of the properties of the Hasse diagrams of consecutive prefixes of a given step sequence, resulting in what can be regarded as a linear algorithm.

5 Direct Construction of Hasse Diagrams of Comtraces

In this section we will show how to construct the (folded) Hasse diagram directly from a given representative v of a comtrace. More precisely, the input to the algorithm is a comtrace alphabet (Σ, sim, ser) and a step sequence $v \in \mathbb{S}^*$. We first describe the algorithm and provide its pseudo-code. After that, we discuss its complexity.

The algorithm is on-line (i.e., its consecutive phases generate correct Hasse diagrams for all the prefixes of v), and exploits the knowledge of the structure of the intermediate diagrams. A key observation concerns the tail of an intermediate folded Hasse diagram (denoted as $\widehat{tail}$) captured by Proposition 14. Such a tail comprises the latest occurrences of each action which has so far been 'seen' by the algorithm. Only the elements from the tail and those arising from the currently processed step may be connected by newly generated arcs.

We therefore introduce an auxiliary data structure that keeps information about the tails of intermediate diagrams. For every action $a \in \Sigma$, the data structure consists of an ordered dependence list ORD_a, followed by two sets of visible sources, SVS_a and WVS_a, giving strongly visible sources and properly weakly visible sources of a, as well as a pointer to the latest occurrence of a. The lists and sets have the size at most k. During the computation, we also use temporary copies of sets of sources (visibility sets).

In contrast to the case of Mazurkiewicz traces, when constructing Hasse diagrams of comtraces, we have to deal not only with two kinds of dependencies between action occurrences, but also with their local contexts. We say that occurrences α and β are *locally dependent* if their folded actions are not completely independent by which we mean that $(\ell(\langle\alpha\rangle) \times \ell(\langle\beta\rangle)) \cap ind^{-1} \neq \varnothing$. The list ORD_a contains all the actions whose latest occurrences are locally dependent with latest occurrence of a.

The set SVS_a contains all the actions b whose latest occurrences were *strongly visible* from the latest occurrence labelled with a, or are in the same folded action as the latest occurrence of a. In other words, there exists a strong path from β to α, where β and α are respectively the latest occurrences of actions b and a, or $\langle\alpha\rangle = \langle\beta\rangle$. Each such β is called a *strong source* of α.

Similarly, the set WVS_a contains all the actions b whose latest occurrences were weakly visible from the latest vertex labelled with a. In other words, there exists a weak path from β to α, where β and α are respectively the latest occurrences of actions b and a in the diagram constructed so far. Each such β is called a *weak source* of α.

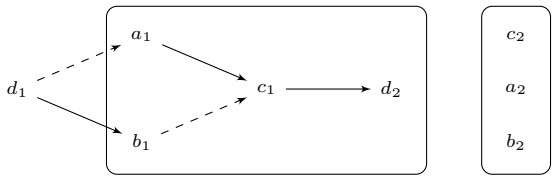
The last element in the data structure, LST_a, is a pointer to the most recent vertex labelled with the action a in the diagram being constructed.

Before generating the Hasse diagram of $[v]$, we set all the pointers to *null*, and make all the sets of sources and all dependence lists empty. It is worth stressing that dependence lists are computed dynamically and some actions present in the list ORD_a may be even independent with action a.

We then process consecutive steps of the input sequence v, updating the auxiliary data structure (possibly many times during a single phase), and creating

new vertices and arcs. We will now describe a single phase of the algorithm which starts with the Hasse diagram $H(w) = (occ(w), \prec_w^h, \sqsubset_w^h)$ of a step sequence w of length m together with an auxiliary data structure for its tail $tail(w)$. To construct the Hasse diagram of wA, for $A \in \mathbb{S}$, we proceed in two stages.

Example 15. Consider the comtrace alphabet form Example 4, a step sequence $w = \{d\}\{a,b\}\{c\}\{d\}$ and a step $A = \{a,b,c\}$. Then the Hasse diagram of $H(w)$ together with an auxiliary data structure as well as the set A look as follows:

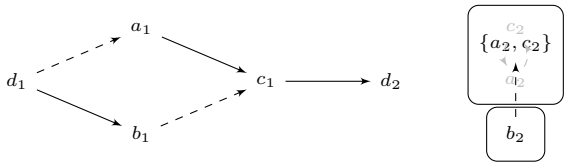


	ORD	*SVS*	*WVS*	*LST*
a	(d,c,a)	$\{a\}$	$\varnothing$	$\rightarrow a_1$
b	(d,c,b)	$\{b\}$	$\varnothing$	$\rightarrow b_1$
c	(d,c,b,a)	$\{a,c\}$	$\{b\}$	$\rightarrow c_1$
d	(d,c,a,b)	$\{a,b,c,d\}$	$\varnothing$	$\rightarrow d_2$

□

In the first stage, we identify [2] all the strongly connected components SCC of the directed graph $(occ_{m+1}(wA), \sqsubset_{wA} |_{occ_{m+1}(wA)\times occ_{m+1}(wA)})$ which gives us a partition of $occ_{m+1}(wA)$ into folded actions (see Proposition 5). We use them to produce new vertices. In this way, we construct the missing part of $\widehat{tail}(wA)$. According to Proposition 5, the directed graph $DAG = (SCC, \widehat{\sqsubset}|_{SCC\times SCC})$ is a poset. We compute any of its linearisations using the topological sort [2] and store the results in the list $TS = (TS_1\, TS_2, \ldots, TS_q)$. This is crucial for the correctness of whole procedure. Let us just notice that each $[wTS_1TS_2\ldots TS_k]$, for $1 \leq k \leq q$, is a correct prefix of $[wA]$ (in other words, $A = TS_1\ldots TS_k \circ TS_{k+1}\ldots TS_q$).

Example 16. Continuing Example 15, after the first stage we obtain:



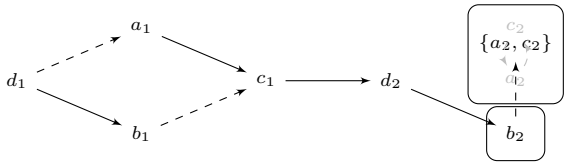
□

In the second stage, we scan the list TS and add new arcs. We divide the processing of each vertex $\Delta = TS_i$ into two parts. Firstly, we check the necessity of adding new arcs from $\widehat{tail}$ to Δ. Secondly, we update the structure of visibility sets and dependence lists. It is important to emphasize that the structure of the diagram and its tail is updated many times in the second stage of each phase.

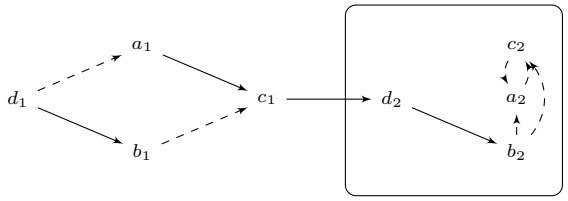
At the beginning of the first part we initialise two auxiliary sets SVS and WVS with the value $\varnothing$. To check the necessity of adding an arc from $\widehat{tail}$ to vertex Δ, we need to compute the dependence list for this vertex. Notice that all linear orders stored in dependence lists ORD_a are compatible and it is possible to compute a minimal linear order that contains them. We do this in an arbitrary way producing a single list ORD_Δ. After that we scan the list ORD_Δ. For each b from ORD_Δ, we check whether the pointer LST_b is not *null*. If it is not, there are two possibilities, as we can try to add new strong arc or new weak arc. To check the type of local dependence between folded action $\Delta = \langle\alpha\rangle$ and the latest occurrence of b, β, we have to fold action occurrence β into folded action $\langle\beta\rangle$ as well. We do that by checking all outgoing arcs. All occurrences δ connected with a short loop ($\beta \sqsubset \gamma \sqsubset \beta$) are in the same vertex of folded action as β. Now, if $(\ell(\Delta) \times \ell(\langle\beta\rangle)) \cap (wdp \cup dep \cup ssm) \neq \varnothing$ then these folded actions are strongly dependent ($\langle\beta\rangle \widehat{\prec}^{\mathsf{h}} \Delta$). Otherwise they are weakly dependent ($\langle\beta\rangle \widehat{\sqsubset}^{\mathsf{h}} \Delta$).

In the first case (trying to add a strong arc), we check if $\ell(\langle\beta\rangle) \cap SVS = \varnothing$. If so, we add a new strong arc from $\langle\beta\rangle$ to Δ and update the sets SVS and WVS by adding $SVS_b \cup WVS_b$ to SVS and subtracting SVS from WVS. In the second case (trying to add a weak arc), we have to check if $\ell(\langle\beta\rangle) \cap (SVS \cup WVS) = \varnothing$. If so, we add a new weak arc from $\langle\beta\rangle$ to Δ and update the set SVS and WVS by adding $SVS_b \setminus \ell(\langle\beta\rangle)$ to SVS and $WVS_b \cup \ell(\langle\beta\rangle)$ to WVS. To preserve the separation of the sets SVS and WVS, we also subtract SVS from WVS. At the end, we split the vertex Δ into $|\Delta|$ vertices (splitting also all newly added arcs), and make from the resulting set of new vertices a weakly connected clique.

Example 17. Continuing Example 16, during the second stage we have:



After the second stage we obtain:

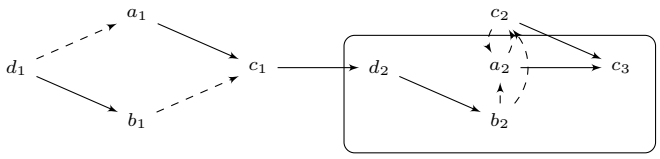


	ORD	SVS	WVS	LST
a	(c, a, d)	$\{a, c, d\}$	$\{b\}$	$\to a_2$
b	(c, b, d)	$\{b, d\}$	$\varnothing$	$\to b_2$
c	(c, a, b, d)	$\{a, c, d\}$	$\{b\}$	$\to c_2$
d	(c, a, b, d)	$\{d\}$	$\varnothing$	$\to d_2$

□

At this point we have added all the necessary arcs, but the tail of the diagram has changed and therefore the visibility sets should also be updated. The second part starts, by subtracting the labels $\ell(\Delta)$ from every visibility set SVS_a and WVS_a. Then we update all the lists ORD_a by moving, one by one, the labels of the elements of the folded action Δ to the beginning of the list or removing them if their latest occurrences are no more locally dependent with the latest occurrences of labels of Δ. These updates can be done in arbitrary but fixed order of labels of the action occurrences belonging to Δ. The last operation is the updating of the visibility sets for the labels of the elements of Δ. For each a that belongs to $\ell(\Delta)$, we set SVS_a to $SVS \cup \ell(\Delta)$, and WVS_a to $WVS \cup \ell(\Delta)$. We also update (for all $a \in \ell(\Delta)$) the values of the pointers LST_a.

Example 18. Continuing Example 17, after adding the next step $\{c\}$ we get:



Algorithm 1: Hasse diagram

INPUT: step sequence $v = A_1 \dots A_q$ over comtrace alphabet (Σ, sim, ser)
OUTPUT: Hasse diagram of $H(v)$

1: **for all** $a \in \Sigma$ **do**
2: $LST_a := -1$; $SVS_a := \varnothing$ $WVS_a := \varnothing$
3: **end for**
4: Compute relations dep, ind, ssm, wdp
5: Compute sets ORD_a using relation ind
6: **for** $i := 1$ to q **do**
7: Compute all strongly connected components SCC of $(occ_i(A_1 \dots A_i), \sqsubset_{A_1 \dots A_i}|_{occ_i(A_1 \dots A_i) \times occ_i(A_1 \dots A_i)})$
8: Compute list TS using topological sorting on $DAG = (SCC, \widehat{\sqsubset}|_{SCC \times SCC})$
9: **for all** $\Delta \in TS$ **do**
10: Add new strong and weak arcs {Part 1}
11: Update data structure {Part 2}
12: **end for**
13: **end for**

Algorithm 2: Stage 2, Part 1: Add new arcs

1: Add vertex Δ
2: $SVS = WVS = \varnothing$
3: Combine lists ORD_a for all actions $a \in \ell(\Delta)$ into ORD_Δ
4: **for all** $b \in ORD_\Delta$ **do**

```
5:    β = LST_b
6:    Compute ⟨β⟩
7:    if ⟨β⟩ is strongly dependent with Δ and ⟨β⟩ ∩ SVS = ∅ then
8:       Add a strong arc (⟨β⟩, Δ)
9:       SVS := SVS ∪ SVS_b ∪ WVS_b
10:      WVS := WVS \ SVS
11:   end if
12:   if ⟨β⟩ is weakly dependent with Δ and ⟨β⟩ ∩ (SVS ∪ WVS) = ∅ then
13:      Add a weak arc (⟨β⟩, Δ)
14:      SVS := SVS ∪ (SVS_b \ ℓ(⟨β⟩))
15:      WVS := (WVS ∪ WVS_b) \ SVS
16:   end if
17: end for
18: for all α ∈ Δ do
19:   Add vertex α
20:   for all arc (Δ', Δ) do
21:      for all β ∈ Δ' do
22:         Add arc (β, α)
23:      end for
24:   end for
25: end for
26: Remove vertex Δ (with all arcs)
```

Algorithm 3: Stage 2, Part 2: Update the data structure

```
1: for all b ∈ Σ do
2:    SVS_b := SVS_b ∪ ℓ(Δ)
3:    WVS_b := WVS_b ∪ ℓ(Δ)
4:    for all a ∈ ℓ(Δ) do
5:       Move a to the head of ORD_b
6:    end for
7: end for
8: for all a ∈ ℓ(Δ) do
9:    SVS_a := SVS \ ℓ(Δ)
10:   WVS_a := WVS \ ℓ(Δ)
11:   Update LST_a
12: end for
```

We will now discuss the complexity of the proposed algorithm. In what follows, d denote the size of currently precessed folded action, k denotes the size of the alphabet Σ, and n denotes the size $|occ(v)|$ of the input step sequence v.

The algorithm is on-line, and so we do not have to store the entire step sequence nor the Hasse diagram. All we need to maintain is the structure that describes the tail of the current diagram. In this structure we store k lists ORD_a of size k each, which contributes $O(k^2)$ to the memory complexity. We also store

k sets SVS_a, k sets WVS_a, k pointers LST_a, and a constant number of other variables, but these not increase the memory complexity.

Estimating time complexity is more involved. Let us start by evaluating the two parts of the second stage (see Algorithm 2 and Algorithm 3). In the first part we combine dk lists of size k while preserving the dependencies stored in these lists (line 3). It gives $O(dk^2)$ time complexity. After that, for every action $b \in Ord_\Delta$ we attempt to add some arcs. The tests carried out in lines 7 and 11 gives time complexity $O(dk)$. The set operations in lines 9,10,13 and 14 and computing $\langle\beta\rangle$ in line 6 give $O(k)$ time complexity each, while the time involved in adding an actual arc is constant. The whole loop (lines 4-14) has therefore time complexity of $O(dk^2)$. The time complexity of the second loop (lines 15-19), and of removing a vertex in line 20 are clearly $O(dk^2)$. As a result, we have $O(dk^2)$ total time complexity for the first part of the second phase.

The second part has also complexity of $O(dk^2)$. In Algorithm 3, we have two loops with at most k iterations. In each iteration, we carry out some set operations of the complexity $O(k)$ each, and d rearrangements of lists ORD_a which can also be done in $O(dk)$ time.

Let us now calculate the total time complexity of Algorithm 1. The pre-processing phase (lines 1-5) has the time complexity of $O(k^2)$. Then we have a main loop with at most n iterations. The first phase involves some operations on graphs of the size $O(k^2)$ or $O(k)$ (by the size of a graph we mean its total number of vertices and arcs). Detecting strongly connected components and topological sorting of a directed acyclic graph are both linear in its size (see [14]), and so we can carry out the first phase for each step in $O(k^2)$ time. This contributes $O(nk^2)$ component to the overall time complexity.

Finally, we notice that the number of the iterations in loop from line 9 is linear in the size of the original step sequence. Moreover, every folded action is counted only once, and so when calculating the total complexity, we can skip the d factor of the complexity of the second stage. This contributes another $O(nk^2)$ component to the overall time complexity. Hence the time complexity is equal to $O(nk^2)$. As in the case of traces, k is bounded by $|\Sigma|$ (which is fixed for a given system) and much smaller than n. Hence, the above algorithm can in practice be considered as linear in the size of its input, and so optimal.

6 Applications

A major advantage of Hasse diagrams of comtraces and the data structure that is used by the algorithm described in the previous section is a convenient and efficient representation of comtraces. In fact, it is worth adding information about the head of a comtrace. In the extended structure, we store pointers to all the elements belonging to $head(w)$. Intuitively, when concatenating two comtraces, all new connections between their folded Hasse diagrams would in fact be between the tail of first diagram and the head of the second one.

Another application of Hasse diagrams is a method of comparing comtraces as two step sequences, w and v, belong to the same comtrace if and only if their

Hasse diagrams are equal. More formally:

$$w \equiv_\Theta v \iff H(w) = H(v) \iff \widehat{H}(w) = \widehat{H}(v) \ .$$

Testing for equality of two graphs is linear in their size, and so using Hasse diagrams allows testing for comtrace equivalence in $O(nk^2)$ time.

Using Hasse diagram $H(w) = (occ(w), \prec^{\mathsf{h}}, \sqsubset^{\mathsf{h}})$ with the head structure $head(w)$, we can also compute the Foata normal form of $[w]$. To do so, we only need to compute the maximal step F_1 containing the head. The step F_1 is the first step of the new step sequence F. After that, we proceed by computing the set F_2' of all action occurrences that are the targets of the strong arcs originating in F_1. To compute the second step F_2, we need to compute the transitive closure of F_2' (closing the relation $\sqsubset^{\mathsf{h}}$), testing that none of the vertices contained in F_2 is in the relation $\prec^{\mathsf{h}}$ with F_2'. The correctness of the resulting procedure follows directly from the definition of Foata normal form. Its time complexity is linearly dependent on the size of Hasse diagram and therefore equal to $O(nk)$.

7 Conclusions

In this paper, we presented an efficient way of generating graph theoretic representations of comtraces. We also provided a number of properties of folded stratified order structures.

In our future work we plan extend our current results to cover also generalised comtraces and generalised so-structures [5, 6]. Another direction is the development of more efficient algorithm for checking equivalence of two step sequences. Yet another, arguably most challenging, problem is to use Hasse diagrams of comtraces to define an algebra of comtraces with a suitable iteration operator.

References

1. Pierre Cartier and Dominique Foata. *Problèmes Combinatoires de Commutation et Réarrangements*, volume 85 of *LNM*. Springer, Berlin, 1969.
2. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1994.
3. Volker Diekert and Yves Métivier. Partial commutation and traces. In *Handbook of Formal Languages*, volume 3, pages 457–533. Springer, 1997.
4. Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.
5. Ryszard Janicki. Relational structures model of concurrency. *Acta Inf.*, 45(4):279–320, 2008.
6. Ryszard Janicki, Jetty Klein, and Maciej Koutny. Quotient monoids and concurrent behaviours. In Carlos Martín-Vide, editor, *Scientific Applications of Language Methods [8]*, chapter 6, pages 313–386. Imperial College Press, London, 2011.
7. Ryszard Janicki and Maciej Koutny. Semantics of inhibitor nets. *Inf. Comput.*, 123(1):1–16, 1995.
8. Carlos Martín-Vide, editor. *Scientific Applications of Language Methods*. Imperial College Press, 2011.

9. Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. Daimi report pb-78, Aarhus University, 1977.
10. Łukasz Mikulski. Projection representation of Mazurkiewicz traces. *Fundamenta Informaticae*, 85:399–408, 2008.
11. Łukasz Mikulski and Maciej Koutny. Hasse diagrams of combined traces. Technical report cs-tr-1301, Newcastle University, 2011.
12. Łukasz Mikulski, Marcin Piątkowski, and Sebastian Smyczyński. Algorithmics of posets generated by words over partially commutative alphabets. In Jan Holub and Jan Žďárek, editors, *Proceedings of the Prague Stringology Conference 2011*, pages 209–219, Czech Technical University in Prague, Czech Republic, 2011.
13. Grzegorz Rozenberg and Joost Engelfriet. Elementary net systems. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 12–121. Springer, 1996.
14. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2), 1992.
15. Henri Vogt. *Leçons sur la résolution algébrique des équations.* Cornell University Library historical math monographs. Nony, 1895.