Synthesis and Analysis of Petri Nets from Causal Specifications

Mateus de Oliveira Oliveira[✉]

University of Bergen, Bergen, Norway
mateus.oliveira@uib.no

Abstract. Petri nets are one of the most prominent system-level formalisms for the specification of causality in concurrent, distributed, or multi-agent systems. This formalism is abstract enough to be analyzed using theoretical tools, and at the same time, concrete enough to eliminate ambiguities that would arise at implementation level. One interesting feature of Petri nets is that they can be studied from the point of view of true concurrency, where causal scenarios are specified using partial orders, instead of approaches based on interleaving.

On the other hand, message sequence chart (MSC) languages, are a standard formalism for the specification of causality from a purely behavioral perspective. In other words, this formalism specifies a set of causal scenarios between actions of a system, without providing any implementation-level details about the system.

In this work, we establish several new connections between MSC languages and Petri nets, and show that several computational problems involving these formalisms are decidable. Our results fill some gaps in the literature that had been open for several years. To obtain our results we develop new techniques in the realm of slice automata theory, a framework introduced one decade ago in the study of the partial order behavior of bounded Petri nets. These techniques can also be applied to establish connections between Petri nets and other well studied behavioral formalisms, such as the notion of Mazurkiewicz trace languages.

Keywords: MSC Languages · Mazurkiewicz Traces · Petri Nets

1 Introduction

Petri nets are one of the most prominent system-level formalisms for the specification of causality in concurrent, distributed or multi-agent systems. This formalism is abstract enough to be analyzed using theoretical tools, and at the same time, concrete enough to eliminate ambiguities that would arise at implementation level. One interesting feature of Petri nets is that they can be studied from the point of view of true concurrency, where causal scenarios are specified using partial orders, instead of approaches based on interleaving [18,36]. On the other hand, message sequence chart (MSC) languages [16,19], are a standard formalism for the specification of causality from a purely behavioral perspective. In other words, this formalism specifies a set of causal scenarios between actions of a system, without providing any implementation-level details about the system.

In this work, we show that given an MSC automaton $\mathcal{M}$ specifying a set of partial orders $\mathcal{L}_{po}(\mathcal{M})$, and a b-bounded Petri net N with causal behavior $\mathcal{P}_{cau}(N)$, it is decidable whether $\mathcal{L}_{po}(\mathcal{M}) \cap \mathcal{P}_{cau}(N) \neq \emptyset$, and whether $\mathcal{L}_{po}(\mathcal{M}) \subseteq \mathcal{P}_{cau}(N)$ (Theorem 8). Additionally, for any given $b \in \mathbb{N}_+$, one can synthesize a b-bounded Petri net N that best captures the behavior specified by $\mathcal{M}$ (Theorem 9). More specifically, $\mathcal{L}_{po}(\mathcal{M}) \subseteq \mathcal{P}_{cau}(N)$, and there is no other b-bounded Petri net N' such that $\mathcal{L}_{po}(\mathcal{M}) \subseteq \mathcal{P}_{cau}(N') \subsetneq \mathcal{P}_{cau}(N)$. Finally, if the MSC automaton $\mathcal{M}$ is *locally synchronized*, a well studied property in the context of MSC language theory [1,19,31], then one can also test whether $\mathcal{P}_{cau}(N) \subseteq \mathcal{L}_{po}(\mathcal{M})$ (Theorem 8).

The feasibility of all computational problems described above have been open even for 1-bounded Petri nets, despite the fact that both Petri nets and MSC languages have been defined several decades ago. The key of our results is a new connection between MSC automata and slice automata, a formalism introduced in [33] in the study of the partial order behavior of bounded Petri nets. More specifically, we show that for each MSC automaton $\mathcal{M}$, one can construct a slice automaton $\mathcal{A}$ such that $\mathcal{L}_{po}(\mathcal{M}) = \mathcal{L}_{po}(\mathcal{A})$ (Theorem 7). A crucial feature of this construction is that it preserves good decidability properties. More precisely, if the input MSC automaton $\mathcal{M}$ is locally synchronized, then the obtained slice automaton $\mathcal{A}$ satisfies a property called saturation, which is crucial for the analysis of the causal behavior of Petri nets against safety specifications. To establish the connection mentioned above, we develop new slice-theoretic machinery of independent interest. In particular, we introduce the notions of slice-traces, and the notion of a locally synchronized slice automaton. In Sect. 8, we show that this new framework can also be used to establish connections between slice automata (and therefore, Petri nets), and the formalism of Mazurkiewicz trace languages [8,12,20,24,28,28], which is another well-studied formalism for the specification of sets of partial orders. In this case, it also holds that our reductions preserve good decidability properties, in the sense that finite automata accepting trace-closed languages are mapped to saturated slice automata.

Related Work. During the last four decades many partial order formalisms have been introduced and several connections have been established between these formalisms [9,13,17,21,25,29,36]. In particular, the expressiveness of finite message-passing automata with a priori unbounded FIFO channels was studied in [5], where it was shown that these automata capture exactly the class of MSC languages that are definable in existential monadic second-order logic interpreted over MSCs. Asynchronous cellular automata for traces were originally introduced by Zielonka [37]. A notion of asynchronous cellular automaton for pomsets without auto-concurrency was devised in [10]. Existentially bounded communicating automata have been considered in [14] where an equivalence was established between communication automata, globally cooperative compositional message sequence graphs and monadic second-order logic. Several connections between communicating automata with bounded channels and Mazurkiewicz traces have been considered in [15]. Generalizations of Mazurkiewicz traces have been considered in [22], and some extensions of message sequence graphs that are suitable for model checking under MSO specifications have been considered in [27]. Series parallel languages have been considered in [26]. It is important to note that the class of partial orders that can be accepted by slice automata are incomparable with the class of series parallel partial orders. On the one hand, series parallel partial orders are not necessarily k-bounded in the sense considered in this work. On the other hand, it is easy to construct k-bounded partial orders that are not series parallel. In particular, for $k \geq 4$, slice automata are able to define k-bounded partial orders whose underlying undirected graph have the complete graph K_4 as a minor, whereas it is known that no such partial order can be series parallel. It is also worth noting that none of the formalisms described in this paragraph are able to represent the causal behavior of arbitrary bounded Petri nets. Generalizations of finite automata accepting infinite words have been considered in several contexts. For instance, regular sets of infinite message sequence charts [23]. Automata over message sequence charts capable of accepting infinite MSCs were studied in [4]. We note that we do not consider automata capable of accepting infinite partial orders in this work.

2 The Causal Semantics of Petri Nets

In this section, we briefly define the classic notion of Petri-nets and describe their partial-order semantics. Within this semantics, partial orders are used to represent the causality between events in concurrent runs of a Petri net.

A Petri net is a tuple $N = (P, T, W, \mathfrak{m}_0)$ where P is a set of *places*, T is a set of *transitions* such that $P \cap T = \emptyset$, $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a function that assigns a weight $W(x, y)$ to each element $(x, y) \in (P \times T) \cup (T \times P)$, and $\mathfrak{m}_0 : P \to \mathbb{N}$ is a function that assigns a non-negative integer $m_0(p)$ to each place $p \in P$.

A marking for N is any function of the form $\mathfrak{m} : P \to \mathbb{N}$. Intuitively, a marking $\mathfrak{m}$ assigns a number of tokens to each place of N. The marking $\mathfrak{m}_0$ is called the *initial marking* of N. If $\mathfrak{m}$ is a marking and t is a transition in T, then we say that t is enabled at $\mathfrak{m}$ if $\mathfrak{m}(p) - W(p, t) \geq 0$ for every place $p \in P$. If this is the case, the firing of t yields the marking $\mathfrak{m}'$ which is obtained from $\mathfrak{m}$ by setting $\mathfrak{m}'(p) = \mathfrak{m}(p) - W(p, t) + W(t, p)$ for every place $p \in P$. A firing sequence for N is a mixed sequence of markings and transitions $\mathfrak{s} \equiv \mathfrak{m}_0 \xrightarrow{t_1} \mathfrak{m}_1 \xrightarrow{t_2} ... \xrightarrow{t_n} \mathfrak{m}_n$ such that for each $i \in \{1, ..., n\}$, t_i is enabled at $\mathfrak{m}_{i-1}$, and $\mathfrak{m}_i$ is obtained from $\mathfrak{m}_{i-1}$ by the firing of t_i. We say that such a firing sequence is b-bounded if for each $i \in \{0, ..., n\}$ and each $p \in P$, $\mathfrak{m}_i(p) \leq b$. We say that N is b-bounded if each of its firing sequences is b-bounded.

The partial order semantics of Petri nets is defined using the notion of Petri-net processes introduced by Goltz and Reisig in [18]. The information about the causality between events is extracted from objects called Petri net *processes*, which encode the production and consumption of tokens along a concurrent run of the Petri net in question. The definition of processes, in turn, is based on the notion of *occurrence net*.

An occurrence net is a DAG $O = (B \mathbin{\dot\cup} V, F)$ where the vertex set $B \mathbin{\dot\cup} V$ is partitioned into a set B, whose elements are called conditions, and a set V, whose elements are called events. The edge set $F \subseteq (B \times V) \cup (V \times B)$ is restricted in such a way that for every condition $b \in B$, $|\{(b,v) \mid v \in V\}| \leq 1$ and $|\{(v,b) \mid v \in V\}| \leq 1$. In other words, conditions in an occurrence net are unbranched. For each condition $b \in B$, we let $InDegree(b)$ denote the number of edges having b as target. A process of a Petri net N is an occurrence net whose conditions are labeled with places of N, and events are labeled with transitions of N. Processes are intuitively used to describe the token game in a concurrent execution of the net.

Definition 1 (Process [18]). *A process of a Petri net $N = (P, T, W, m_0)$ is a labeled DAG $\pi = (B \dot\cup V, F, \rho)$ where $(B \dot\cup V, F)$ is an occurrence net and $\rho : (B \cup V) \to (P \cup T)$ is a labeling function satisfying the following properties.*

1. *Places label conditions and transitions label events: $\rho(B) \subseteq P$ and $\rho(V) \subseteq T$.*
2. *For every $p \in P$, $|\{b : InDegree(b) = 0, \rho(b) = p\}| = m_0(p)$.*
3. *For every $v \in V$, and every $p \in P$, $|\{(b,v) \in F : \rho(b) = p\}| = W(p, \rho(v))$ and $|\{(v,b) \in F : \rho(b) = p\}| = W(\rho(v), p)$.*

Item 1 says that the conditions of a process are labeled with places, while the events are labeled with transitions. Item 2 says that the minimal vertices of the process, are conditions. Intuitively, each of these conditions represent a token in the initial marking of N. Thus for each place p of N the process has $m_0(p)$ minimal conditions labeled with the place p. Item 3, determines that the token game of a process corresponds to the token game defined by the firing of transitions in the Petri net N. Thus if a transition t consumes $W(p,t)$ tokens from place p and produces $W(t,p)$ tokens at place p, then each event labeled with t must have $W(p,t)$ in-neighbours that are conditions labeled with p, and $W(t,p)$ out-neighbours that are conditions labeled with p.

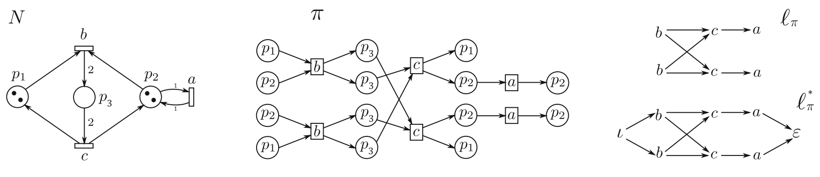

Fig. 1. A 2-bounded Petri net N. A process π of N. The partial order ℓ_π derived from π. The extension $\hat{\ell}_\pi$ of ℓ_π.

Let $R \subseteq X \times X$ be a binary relation over a set X. We denote by $\mathbf{tc}(R)$ the transitive closure of R. If $\pi = (B \cup V, F, \rho)$ is a process then the *causal order* of π is the partial order $\ell_\pi = (V, \mathbf{tc}(F)|_{V \times V}, \rho|_V)$ which is obtained by taking the transitive closure of F and subsequently by restricting $\mathbf{tc}(F)$ to pairs of events of V. In other words the causal order of a process π is the partial order induced by π on its events.

If $\ell = (V, <, l)$ is a partial order, then we let $\ell^* = (V', <', l')$ be the *extended version* of ℓ, where $V' = V \cup \{v_\iota, v_\varepsilon\}$, $<' = < \cup (\{v_\iota\} \times V) \cup (V \times \{v_\varepsilon\}) \cup \{(v_\iota, v_\varepsilon)\}$, $l'|_V = l$, $l'(v_\iota) = \iota$ and $l'(v_\varepsilon) = \varepsilon$. In other words, ℓ' is obtained from ℓ by the addition of an element v_ι that is smaller than all other elements, and an element v_ε that is greater than all other elements. The addition of these minimal and maximal elements to a partial order are made to avoid the consideration of special cases in some of our future lemmas. All of our results work if ignore this step, but at the expense of more repetitive proofs that deal with corner cases. We denote by $\mathcal{P}_{cau}(N)$ the set of all extended versions of partial orders derived from processes of N: $\mathcal{P}_{cau}(N) = \{\hat{\ell}_\pi | \pi \text{ is a process of } N\}$. We say that $\mathcal{P}_{cau}(N)$ is the causal language of N. We observe that several processes of N may correspond to the same partial order in $\mathcal{P}_{cau}(N)$.

Recall that the Hasse diagram of a partial order $\ell = (V, <, l)$ is the DAG $H = (V, E)$ with the smallest number of edges with the property that $<= \mathbf{tc}(E)$. It is a well known result in partial order theory that this DAG is unique. We say that ℓ is a k-partial-order, for some $k \in \mathbb{N}$, if there exist k paths $\mathfrak{p}_1, \ldots, \mathfrak{p}_k$ in H that cover all vertices and edges of H. In other words, $V = \bigcup_i V_i$ and $E = \bigcup_i E_i$ where for each $i \in \{1, \ldots, k\}$, V_i and E_i are the vertex-set and edge-set of the path $\mathfrak{p}_i$ respectively. We note that the paths in the cover are not necessarily vertex-disjoint nor edge-disjoint.

For each $k \in \mathbb{N}$, let $\mathcal{P}_{cau}(N, k)$ denote the set of k-partial-orders which are causal-orders of N. It can be shown that if N is b-bounded, then every causal-order of $N = (P, T, W, \mathfrak{m}_0)$ is a $(b \cdot |P|)$-partial-order. In other words, each causal-order of N can be covered by at most $b \cdot |P|$ paths. This implies that $\mathcal{P}_{cau}(N) = \mathcal{P}_{cau}(N, b \cdot |P|)$.

3 Message Sequence Chart Languages

Message Sequence Charts (MSCs) are a suitable formalism for the representation of the exchange of messages between processes of a concurrent systems. In particular, during the last two decades, MSCs have been used to specify runs of telecommunication protocols. Intuitively, an MSC can be formalized as a partial-order that represents the causality between messages exchanged in a given concurrent run. Infinite families of MSCs, and therefore infinite families of partial-orders, can be specified using equivalent formalisms such as *message sequence graphs*, *hierarchical (or high-level) message sequence charts (HMSCs)* [2,30,32], or *message sequence chart automata* which will be defined below.

We formalize MSCs according to the terminology used in [30]. Let $\mathcal{J}$ be a finite set of processes, also called instances. For each instance $i \in \mathcal{J}$, we associate a finite set of actions $\Sigma_i = \Sigma_i^{int} \cup \Sigma_i^! \cup \Sigma_i^?$. This set is partitioned into a set Σ_i^{int} of *internal actions*, a set $\Sigma_i^! = \{i!j : j \in \mathcal{J} \backslash \{i\}\}$ of *send actions*, and a set $\Sigma_i^? = \{i?j : j \in \mathcal{J} \backslash \{i\}\}$ of *receive actions*. We shall assume that for each two distinct instances $i, j \in \mathcal{J}$, $\Sigma_i \cap \Sigma_j = \emptyset$. The set of actions associated with $\mathcal{J}$ is defined as $\Sigma_{\mathcal{J}} = \bigcup_{i \in \mathcal{J}} \Sigma_i$. Given an action $a \in \Sigma_{\mathcal{J}}$, *Ins*$(a)$ denotes the unique instance i such that $a \in \Sigma_i$. For each $\Sigma_{\mathcal{J}}$-labeled partial-order $\ell = (V, <, l)$ and each vertex $v \in V$, we let $Ins(v) = Ins(l(v))$ be the instance of $\mathcal{J}$ where the action $l(v)$ occurs. For each $i, j \in \mathcal{J}$ with $i \neq j$, and each subset $X \subseteq V$ we let $\#^{i!j}(X) = |\{v \in X \mid l(v) = i!j\}|$ be the number of messages sent from i to j, and by $\#^{i?j}(X) = \{v \in X \mid l(v) = i?j\}$ be the number of messages received by i which were sent by j. We write $v \leq v'$ as a shortcut for $v < v' \vee v = v'$. For each $v \in V$ we let $\downarrow v = \{v' \mid v' \leq v\}$ be the set of all nodes of ℓ which are smaller or equal to v. We write $v \prec v'$ to indicate that $v < v'$ and for every $u \in V$, $v < u \leq v' \Rightarrow u = v'$. In other words, $v \prec v'$ if v' is an out-neighbour of v in the Hasse diagram of ℓ.

Definition 2 (Message Sequence Chart (MSC)). *Let $\mathcal{J}$ be a set of processes. A message sequence chart over $\mathcal{J}$ is a $\Sigma_{\mathcal{J}}$-labeled partial-order $M = (V, <, l)$ satisfying the following properties.*

1. *For every pair of actions $v, v' \in V$ if $Ins(v) = Ins(v')$ then either $v < v'$, $v' < v$ or $v = v'$.*
2. *For every $i, j \in \mathcal{J}$ with $i \neq j$, $\#^{i!j}(V) = \#^{j?i}(V)$.*
3. *For each $v \in V$ and each $i, j \in \mathcal{J}$, if $l(v) = i!j$ and $l(v') = j?i$ and $\#^{i!j}(\downarrow v) = \#^{j?i}(\downarrow v')$, then $v < v'$.*
4. *If $v \prec v'$ and $Ins(v) \neq Ins(v')$, then*
$$l(v) = i!j,\ l(v') = j?i \text{ and } \#^{i!j}(\downarrow v) = \#^{j?i}(\downarrow v').$$

Intuitively, Condition 1 states that actions occurring on the same process are linearly ordered. Condition 2 states that for each two distinct processes i, j, the number of messages send from i to j is equal to the number of messages received by j coming from i. Condition 3 states that for each $n \in \mathbb{N}$, the n-th message sent from i to j is received when the n-th action $j?i$ occurs, i.e., the channels in which these messages are transmitted are assumed to be FIFO. Finally, Condition 4 establishes a causal dependence between send and receive actions from distinct processes.

Let $M = (V, <, l)$ and $M' = (V', <', l')$ be MSCs over $\mathcal{J}$. The composition of M with M' is the MSC $M \circ M' = (V'', <'', l'')$ where $V'' = V \cup V'$, $l'' = l \cup l'$, and $<''$ is the transitive closure of the relation $< \ \cup \ <' \ \cup \{(v, v') \in V \times V' | Ins(v) = Ins(v')\}$.

To define infinite families of partial-orders, we use the notion of *message sequence chart automata* (MSC Automata). Let $\mathbb{M}_{\mathcal{J}}$ be the set of all finite MSCs over $\mathcal{J}$. Here, the set $\mathbb{M}_{\mathcal{J}}$ may be regarded as an (infinite) alphabet of MSCs.

Definition 3. *Let $\mathcal{J}$ be a set of processes. A message sequence chart automaton (MSC automaton) over $\mathcal{J}$ is a finite automaton $\mathcal{M} = (Q, \mathfrak{R}, Q_0, F)$ where Q is finite a set of states, $Q_0 \subseteq Q$ is a set of initial states, F is a set of final states and $\mathfrak{R} \subseteq Q \times \mathbb{M}_{\mathcal{J}} \times Q$.*

We say that a sequence $M_1 M_2 ... M_n$ of MSCs is accepted by $\mathcal{M}$ if there is a sequence $q_0 \xrightarrow{M_1} q_1 \xrightarrow{M_2} ... \xrightarrow{M_n} q_n$ where $q_0 \in Q_0$, $q_n \in F$ and $(q_{i-1}, M_i, q_i) \in \mathfrak{R}$ for each $i \in \{1, ..., n\}$. An MSC automaton generates two languages. At the syntactic level, $\mathcal{L}(\mathcal{M})$ is the set of all sequences $M_1M_2...M_n$ of MSCs accepted by $\mathcal{M}$. At the semantic level,

$$\mathcal{L}_{po}(\mathcal{M}) = \{M_1 \circ ... \circ M_n \mid n \in \mathbb{N}, M_1...M_n \in \mathcal{L}(\mathcal{M})\}$$

is the set of all MSCs obtained by composing each sequence of MSCs in $\mathcal{L}(\mathcal{M})$. We note that an MSC language can be represented by an MSC automaton if and only if it can be represented by the more traditionally used message sequence graphs [2,30,32]. Nevertheless, we choose to work with MSC automata due to the fact that the proof of our results will be shorter.

If M is an MSC, then the communication graph of M, denoted by $G(M)$, has the processes of M as vertices, and has one edge e with source in a process p and target in a process q if and only if p sends some message to q in M. We say that M is locally-synchronized if the graph $G(M)$ has a unique non-trivial[1] strongly connected component, and every vertex that is not in such component is isolated. We say that an MSC automaton $\mathcal{M}$ is locally-synchronized if for each loop $q_1 \xrightarrow{M_1} q_2 \xrightarrow{M_2} ...q_n \xrightarrow{M_n} q_1$ in $\mathcal{M}$, the MSC $M_1 \circ M_2 \circ ... \circ M_n$ is locally-synchronized.

The partial-order language accepted by an MSC automaton is linearization-regular [19] if the set of linearizations of partial-orders in $\mathcal{L}_{po}(\mathcal{M})$ can be recognized by a finite automaton over the alphabet $\Sigma_{\mathcal{J}}$. In other words, $\mathcal{L}_{po}(\mathcal{M})$ is linearization-regular if the following set of strings over $\Sigma_{\mathcal{J}}$ is regular in the usual sense of finite automata theory.

$$lin(\mathcal{M}) = \bigcup_{\ell \in \mathcal{L}_{po}(\mathcal{M})} lin(\ell). \tag{1}$$

It can be shown that an MSC language generated by an MSC automaton $\mathcal{M}$ is linearization-regular if and only if $\mathcal{M}$ is locally synchronized.

Theorem 1 ([2,31]). *Let $\mathcal{M}$ be an MSC automaton. Then $\mathcal{L}_{po}(\mathcal{M})$ is linearization-regular if and only if $\mathcal{M}$ is locally-synchronized.*

4 Slice Automata

In this section we define *slices* and *slice automata*. Slice automata will be used to provide a static representation of infinite families of DAGs and infinite families of partial-orders. We note that slices can be related to several formalisms such as, multi-pointed graphs, [11], co-span decompositions [7] and graph transformations [3,6,11,35].

In what follows, T denotes a finite set of labels. A slice $\mathbf{S} = (V, E, l, s, t, [I, C, O])$ is a $(T \cup \mathbb{N})$-labeled DAG where the vertex set $V = I \,\dot{\cup}\, C \,\dot{\cup}\, O$ is partitioned into an in-frontier I, a center C and an out-frontier O. The function $l : V \rightarrow T \cup \mathbb{N}$ labels the center vertices in C with elements of T, and the in- and out-frontier vertices with positive integers in such a way that $l(I) = \{1, ..., |I|\}$ and

[1] A strongly connected component is trivial if it has a unique vertex.

$l(O) = \{1, ..., |O|\}$. We require that each frontier-vertex v in $I \cup O$ is the endpoint of exactly one edge $e \in E$ and that the edges are directed from the in-frontier to the out-frontier. More precisely, for each edge $e \in E$, we assume that $s(e) \in I \cup C$ and that $t(e) \in C \cup O$. We may also speak of a slice $\mathbf{S}$ with frontiers (I, O) to indicate that the in-frontier of $\mathbf{S}$ is I and that the out-frontier of $\mathbf{S}$ is O.

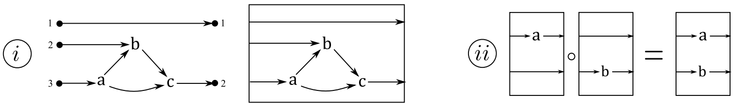

Fig. 2. *i*) A slice and its pictorial representation. *ii*) Composition of slices.

A slice $\mathbf{S}_1 = (V_1, E_1, l_1, s_1, t_1)$ with frontiers (I_1, O_1) can be glued to a slice $\mathbf{S}_2 = (V_2, E_2, l_2, s_2, t_2)$ with frontiers (I_2, O_2) provided $|O_1| = |I_2|$. In this case the glueing gives rise to the slice $\mathbf{S}_1 \circ \mathbf{S}_2 = (V_3, E_3, l_3, s_3, t_3)$ with frontiers (I_1, O_2) which is obtained by taking the disjoint union of $\mathbf{S}_1$ and $\mathbf{S}_2$, and by fusing, for each $i \in \{1, ..., |O_1|\}$, the unique edge $e_1 \in E_1$ for which $l_1(t_1(e_1)) = i$ with the unique edge $e_2 \in E_2$ for which $l_2(s_2(e_2)) = i$. Formally, the fusion of e_1 with e_2 is performed by creating a new edge e_{12} with source $s_3(e_{12}) = s_1(e_1)$ and target $t_3(e_{12}) = t_2(e_2)$, and by deleting e_1 and e_2. Thus in the glueing process the vertices in the glued frontiers disappear.

A *unit slice* is a slice with exactly one vertex in its center. A slice is *initial* if it has empty in-frontier and *final* if it has empty out-frontier. The width of a slice $\mathbf{S}$ with frontiers (I, O) is defined as $w(\mathbf{S}) = \max\{|I|, |O|\}$. A *slice alphabet* is any finite set of slices. In particular, for each finite set of symbols T and each $k \in \mathbb{N}$ we let $\overrightarrow{\boldsymbol{\Sigma}}(k, T)$ be the set of all unit slices $\mathbf{S}$ of width at most k whose center vertex is labeled with an element from T. Observe that the alphabet $\overrightarrow{\boldsymbol{\Sigma}}(k, T)$ is finite and has asymptotically $|T| \cdot 2^{O(k \log k)}$ slices. A sequence $\mathbf{U} = \mathbf{S}_1 \mathbf{S}_2 ... \mathbf{S}_n$ of unit slices is called a unit decomposition if $\mathbf{S}_i$ can be glued to $\mathbf{S}_{i+1}$ for each $i \in \{1, ..., n-1\}$. In this case, we let $\overset{\circ}{\mathbf{U}} = \mathbf{S}_1 \circ \mathbf{S}_2 \circ ... \circ \mathbf{S}_n$ be the DAG associated with $\mathbf{U}$, which is obtained by glueing each two consecutive slices in $\mathbf{U}$. The width of $\mathbf{U}$, denoted by $w(\mathbf{U})$, is defined as the maximum width of a slice occurring in $\mathbf{U}$. We let $\overrightarrow{\boldsymbol{\Sigma}}(k, T)^*$ be the set of all sequences of slices over $\overrightarrow{\boldsymbol{\Sigma}}(k, T)$, and $\overrightarrow{\boldsymbol{\Sigma}}(k, T)^{\circledast}$ be the set of all unit decompositions over $\overrightarrow{\boldsymbol{\Sigma}}(k, T)$.

Definition 4 (Slice Automaton). *Let T be a finite set of symbols and let $k \in \mathbb{N}$. A slice automaton over $\overrightarrow{\boldsymbol{\Sigma}}(k, T)$ is a finite automaton $\mathcal{A} = (Q, \mathfrak{R}, q_0, F)$ where Q is a set of states, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states, and $\mathfrak{R} \subseteq Q \times \overrightarrow{\boldsymbol{\Sigma}}(k, T) \times Q$ is a transition relation such that for every $q, q', q'' \in Q$ and every $\mathbf{S} \in \overrightarrow{\boldsymbol{\Sigma}}(k, T)$:*

1. *if $(q_0, \mathbf{S}, q) \in \mathfrak{R}$ then $\mathbf{S}$ is an initial slice,*
2. *if $(q, \mathbf{S}, q') \in \mathfrak{R}$ and $q' \in F$, then $\mathbf{S}$ is a final slice,*
3. *if $(q, \mathbf{S}, q') \in \mathfrak{R}$ and $(q', \mathbf{S}', q'') \in \mathfrak{R}$, then $\mathbf{S}$ can be glued to $\mathbf{S}'$.*

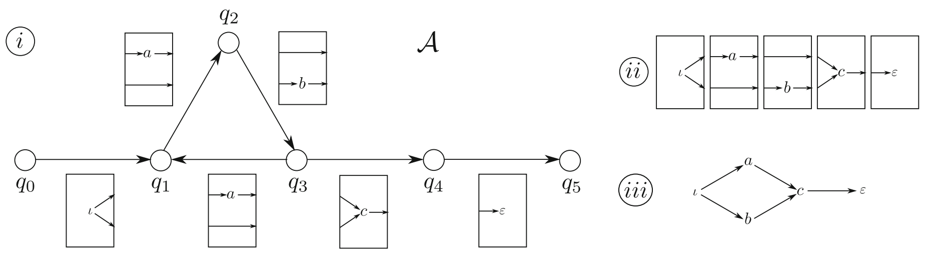

Fig. 3. *i*) A slice automaton $\mathcal{A}$. *ii*) A unit decomposition $\mathbf{U}$ accepted by $\mathcal{A}$. *iii*) The DAG $\mathring{\mathbf{U}}$ obtained by glueing each two consecutive slices in $\mathbf{U}$.

Languages of a Slice Automaton. A slice automaton $\mathcal{A}$ can be used to represent three types of languages. At a syntactic level, we have the slice language $\mathcal{L}(\mathcal{A})$ which consists of the set of all unit decompositions accepted by $\mathcal{A}$.

$$\mathcal{L}(\mathcal{A}) = \{\mathbf{S}_1\mathbf{S}_2...\mathbf{S}_n \mid \mathbf{S}_1\mathbf{S}_2...\mathbf{S}_n \text{ is accepted by } \mathcal{A}\} \tag{2}$$

At a semantic level, we have the graph language $\mathcal{L}_{\mathcal{G}}(\mathcal{A})$ which consists of all DAGs represented by unit decompositions in $\mathcal{L}(\mathcal{A})$, and the partial-order language $\mathcal{L}_{po}(\mathcal{A})$, which consists of all partial-orders which arise as the transitive closure (**tc**) of DAGs in $\mathcal{L}_{\mathcal{G}}(\mathcal{A})$. Formally, the graph language, and the partial-order languages accepted by $\mathcal{A}$ are defined as follows.

$$\mathcal{L}_{\mathcal{G}}(\mathcal{A}) = \{\mathring{\mathbf{U}} \mid \mathbf{U} \in \mathcal{L}(\mathcal{A})\} \qquad \mathcal{L}_{po}(\mathcal{A}) = \{\mathbf{tc}(\mathring{\mathbf{U}}) \mid \mathring{\mathbf{U}} \in \mathcal{L}_{\mathcal{G}}(\mathcal{A})\}. \tag{3}$$

Let H be a DAG whose vertices are labeled with elements from a finite set T. Then we let $\boldsymbol{ud}(H, \overrightarrow{\boldsymbol{\Sigma}}(k,T))$ denote the set of all unit decompositions $\mathbf{U}$ in $\overrightarrow{\boldsymbol{\Sigma}}(k,T)^{\circledast}$ for which $\mathring{\mathbf{U}}= H$. We say that a slice automaton $\mathcal{A}$ over $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$ is *saturated* if for every DAG $H \in \mathcal{L}_{\mathcal{G}}(\mathcal{A})$ we have that $\boldsymbol{ud}(H, \overrightarrow{\boldsymbol{\Sigma}}(k,T)) \subseteq \mathcal{L}(\mathcal{A})$.

The transitive reduction of a DAG $H = (V, E, l)$ is the (unique) minimal subgraph $\mathbf{tr}(H)$ of H with the same transitive closure as H. Note that $\mathbf{tc}(\mathbf{tr}(H)) = \mathbf{tc}(H)$. We say that a DAG H is transitively reduced if $H = \mathbf{tr}(H)$. Alternatively, we call a transitively reduced DAG a Hasse diagram. We say that a slice automaton $\mathcal{A}$ is transitively reduced if every DAG in $\mathcal{L}_{\mathcal{G}}(\mathcal{A})$ is transitively reduced. Theorem 2 states that any slice automaton $\mathcal{A}$ can be converted into a transitively reduced slice automaton $\mathbf{tr}(\mathcal{A})$ representing the same partial-order language in such a way that the saturation property is preserved.

Theorem 2 (**[34]**)**.** *Let $\mathcal{A}$ be a slice automaton over $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$. Then one can construct in time $2^{O(k \log k)} \cdot |\mathcal{A}|$ a transitively reduced slice automaton* $\mathbf{tr}(\mathcal{A})$ *such that $\mathcal{L}_{po}(\mathbf{tr}(\mathcal{A})) = \mathcal{L}_{po}(\mathcal{A})$. Additionally, if $\mathcal{A}$ is saturated, then so is* $\mathbf{tr}(\mathcal{A})$.

Transitively reduced saturated slice automata are important for our setting because they can be used to canonically represent infinite families of partial-orders, and because they enjoy several nice decidability/closure properties. For instance, inclusion and emptiness of intersection of partial-order languages represented by such slice automata are decidable.

Lemma 1 (Properties of Saturated Slice Automata). *Let $\mathcal{A}$ and $\mathcal{A}'$ be transitively-reduced slice automata over $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$. Assume that $\mathcal{A}'$ is saturated.*

1. *It is decidable whether $\mathcal{L}_{po}(\mathcal{A}) \cap \mathcal{L}_{po}(\mathcal{A}') \neq \emptyset$.*
2. *It is decidable whether $\mathcal{L}_{po}(\mathcal{A}) \subseteq \mathcal{L}_{po}(\mathcal{A}')$.*

Additionally, the partial order behavior of bounded Petri nets can be represented using transitively-reduced, saturated slice automata.

Theorem 3 ([33]). *Let $N = (P,T,W,\mathfrak{m}_0)$ be a b-bounded Petri net. Then for each $k \in \mathbb{N}$ one can construct in time $2^{O(|P|\cdot k \cdot \log b \cdot k)} \cdot |T|^{|P|}$ a transitively-reduced, saturated slice automaton $\mathcal{A}(N,k)$ over $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$ such that $\mathcal{L}_{po}(\mathcal{A}(N,k)) = \mathcal{P}_{cau}(N,k)$.*

We note that every partial-order in the causal language of a b-bounded Petri net is a k-partial-order for some $k \leq b \cdot |P|$. Therefore, if we set $\mathcal{A}(N) = \mathcal{A}(N, b \cdot |P|)$ then $\mathcal{L}_{po}(\mathcal{A}(N)) = \mathcal{P}_{cau}(N)$. Finally, synthesis of Petri nets from (any) slice automata is decidable.

Theorem 4 (Synthesis [33]). *Let $\mathcal{A}$ be a slice automaton over $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$. For each $b \in \mathbb{N}$ one can construct a b-bounded Petri net N satisfying the following properties.*

1. *$\mathcal{L}_{po}(\mathcal{A}) \subseteq \mathcal{P}_{cau}(N)$.*
2. *There is no other b-bounded Petri net N' with $\mathcal{L}_{po}(\mathcal{A}) \subseteq \mathcal{P}_{cau}(N') \subsetneq \mathcal{P}_{cau}(N)$.*

5 Weak Saturation

In this section, we introduce the notion of weak-saturation, a relaxation of the notion of saturation that is more suitable for applications involving other partial order formalisms. The main result of this section states that weak-saturated slice automata can be effectively transformed into saturated slice automata.

Let $H = (V,E,l,s,t)$ be a T-labeled DAG and $\omega = (v_1,...,v_n)$ be a topological ordering of the vertices of H. In other words, ω is a sequence of vertices from H such that for each i,j with $i<j$, there is no edge $e \in E$ with source $s(e) = v_j$ and target $t(e) = v_i$. We say that a unit decomposition $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2...\mathbf{S}_n$ over $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$ is compatible with ω if $\mathring{\mathbf{U}} = H$ and for each i, v_i is the center vertex of $\mathbf{S}_i$. Note that given a graph H and a topological ordering ω, there may be several unit decompositions of H compatible with ω. We denote by $\boldsymbol{ud}(H,\omega,\overrightarrow{\boldsymbol{\Sigma}}(k,T))$ the set of all unit decompositions of H over $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$ that are compatible with ω. Note that for each graph H, we have that

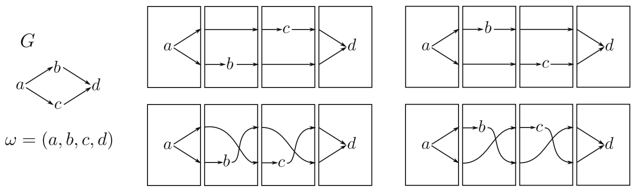

Fig. 4. A graph H, an ordering $\omega = (a, b, c, d)$ of the vertices of H, and all unit decompositions of H compatible with ω. For each unit decompositions $\mathbf{U}, \mathbf{U}'$ in $\boldsymbol{ud}(H, \omega, \overrightarrow{\boldsymbol{\Sigma}}(k, T))$, $\mathbf{U}$ is a twisting of $\mathbf{U}'$.

$$\boldsymbol{ud}(H, \overrightarrow{\boldsymbol{\Sigma}}(k, T)) = \bigcup_{\omega} \boldsymbol{ud}(H, \omega, \overrightarrow{\boldsymbol{\Sigma}}(k, T)) \tag{4}$$

where ω ranges over all topological orderings of H.

Definition 5 (Weak Saturation). *We say that a slice automaton $\mathcal{A}$ is* weakly saturated *if for each DAG H in $\mathcal{L}_{\mathcal{G}}(\mathcal{A})$, and each topological ordering ω of H,*

$$\mathcal{L}(\mathcal{A}) \cap \boldsymbol{ud}(H, \omega, \overrightarrow{\boldsymbol{\Sigma}}(k, T)) \neq \emptyset.$$

In other words, a slice automaton $\mathcal{A}$ is weakly saturated if for each graph H and each topological ordering ω of H there is at least one unit decomposition of H in $\mathcal{L}(\mathcal{A})$ which is compatible with ω. In Sect. 6 we will show that weak saturation is a decidable property. The following lemma states that each weakly saturated slice automaton can be transformed into a saturated slice automaton representing the same set of DAGs, and therefore the same set of partial-orders.

Lemma 2. *Let $\mathcal{A}$ be a weakly saturated slice automaton over $\overrightarrow{\boldsymbol{\Sigma}}(k, T)$. Then one can construct in time $2^{O(k \log k)} \cdot |\mathcal{A}|$ a saturated slice automaton $\mathcal{A}'$ such that $\mathcal{L}_{\mathcal{G}}(\mathcal{A}) = \mathcal{L}_{\mathcal{G}}(\mathcal{A}')$.*

Proof. For $w \geq 0$, let $[w] = \{1, ..., w\}$. We let $[0]$ be the empty set $\emptyset$. A permutation of $[w]$ is a bijective mapping $\boldsymbol{\pi} : [w] \rightarrow [w]$. We denote by $\emptyset$ the empty permutation $\boldsymbol{\pi} : [0] \rightarrow [0]$. Let $\mathbf{S}$ be a slice with frontiers (I, O) and let $\pi : [|I|] \rightarrow [|I|]$ and $\pi' : [|O|] \rightarrow [|O|]$ be permutations. We denote by $(\pi, \mathbf{S}, \pi')$ the slice that is obtained from $\mathbf{S}$ by permuting the labels of the in-frontier nodes according to π, and by permuting the labels of the out-frontier nodes according to π'.

Let $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2...\mathbf{S}_n$ be a unit decomposition over $\overrightarrow{\boldsymbol{\Sigma}}(k, T)$, where each slice $\mathbf{S}_i$ has frontiers (I_i, O_i). Let $\boldsymbol{\pi}_1, ..., \boldsymbol{\pi}_{n-1}$ be a sequence where for each $j \in \{1, ..., n-1\}$, $\boldsymbol{\pi}_j : [|O_j|] \rightarrow [|O_j|]$ is a permutation. Then we say that the unit decomposition

$$\mathbf{U}' = (\emptyset, \mathbf{S}_1, \boldsymbol{\pi}_1)(\boldsymbol{\pi}_1, \mathbf{S}_2, \boldsymbol{\pi}_2)...(\boldsymbol{\pi}_n, \mathbf{S}_n, \emptyset)$$

is a twisting of $\mathbf{U}$. Note that $\mathring{\mathbf{U}}=\mathring{\mathbf{U}}'$ (see Fig. 4). In other words, if $\mathbf{U}$ is a twisting of $\mathbf{U}'$ then both decompositions give rise to the same DAG. Conversely, if $\mathbf{U}$ and $\mathbf{U}'$ are compatible with the same topological ordering of a graph H then $\mathbf{U}$ and $\mathbf{U}'$ are twistings of each other. These remarks are formalized in the next proposition.

Proposition 1. *Let H be a DAG and $\mathbf{U}$ and $\mathbf{U}'$ be unit decompositions of H. Then $\mathbf{U}$ is a twisting of $\mathbf{U}'$ if and only if there is a topological ordering ω of H such that $\mathbf{U}, \mathbf{U}' \in \boldsymbol{ud}(H, \omega, \overrightarrow{\boldsymbol{\Sigma}}(k,T))$.*

We say that a slice automaton $\mathcal{A}$ over $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$ is *twisted* if whenever a unit decomposition $\mathbf{U}$ belongs to $\mathcal{L}(\mathcal{A})$ then all its twistings also belong to $\mathcal{L}(\mathcal{A})$. Alternatively, in view of Proposition 1, $\mathcal{A}$ is twisted if whenever

$$\mathcal{L}_{\mathcal{G}}(\mathcal{A}) \cap \boldsymbol{ud}(H, \omega, \overrightarrow{\boldsymbol{\Sigma}}(k,T)) \neq \emptyset$$

for a DAG H and a topological ordering ω of H, we have that $\boldsymbol{ud}(H, \omega, \overrightarrow{\boldsymbol{\Sigma}}(k,T)) \subseteq \mathcal{L}(\mathcal{A})$. Using Eq. 4 the notion of saturation can be redefined in terms of weak saturation and twisting.

Proposition 2. *Let $\mathcal{A}$ be a slice automaton over $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$. Then $\mathcal{A}$ is saturated if and only if it is both twisted and weakly saturated.*

Therefore, to prove Lemma 2 it is enough to devise a procedure that takes a slice automaton $\mathcal{A}$ and returns a slice automaton $tw(\mathcal{A})$ whose slice language $\mathcal{L}(tw(\mathcal{A}))$ consists of all twisted versions of unit decompositions in $\mathcal{L}(\mathcal{A})$. If $\mathcal{A}$ is weakly saturated, then $tw(\mathcal{A})$ is (fully) saturated.

Let $\mathcal{A} = (Q, \Delta, q^0, F)$. We assume that all states of $\mathcal{A}$ can be reached from the initial state q^0, and reach some final state in F. Let q be a state in Q. We say that the width of q is w if either there is a transition $(q, \mathbf{S}, q')$ such that the in-frontier of $\mathbf{S}$ has size w, or there is a transition $(q', \mathbf{S}, q)$ such that the out-frontier of $\mathbf{S}$ has size w. Note that conditions 1–3 of the definition of slice automaton (Definition 4) ensure that the notion of width of a state is well defined. Now the automaton $tw(\mathcal{A}) = (Q', \Delta', r_0', F')$ is defined as follows:

$$r_0' = q_\emptyset^0 \qquad F' = \{q_\emptyset \mid q \in F\}$$

$$Q' = \{q_\pi \mid \pi : [w(q)] \rightarrow [w(q)] \text{ is a permutation.}\} \tag{5}$$

$$\Delta' = \{(q_\pi, (\pi, \mathbf{S}, \pi'), q_{\pi'}) \mid (q, \mathbf{S}, q') \in \Delta,\ q_\pi, q_{\pi'} \in Q'\}$$

It is immediate to check that a unit decomposition $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2...\mathbf{S}_n$ is accepted by $\mathcal{A}$ if and only each twisting $\mathbf{U}' = (\emptyset, \mathbf{S}_1, \pi_1)(\pi_1, \mathbf{S}_2, \pi_2)...(\pi_{n-1}, \mathbf{S}_n, \emptyset)$ is accepted by $\mathcal{A}'$. Therefore, the automaton $tw(\mathcal{A})$ is twisted. Additionally, if $\mathcal{A}$ is weakly saturated, then by Eq. 4 we have that $tw(\mathcal{A})$ is saturated. Finally, we note that the size of $\mathcal{A}'$ is at most $2^{O(k \log k)} \cdot |\mathcal{A}|$, since there can be at most $O(k!) = 2^{O(k \log k)}$ permutations of a set of labels with at most k elements. $\square$

6 Slice Traces

In this section we introduce the notion of slice traces, and use this notion to show that the weak-saturation property for slice automata is decidable. This notion will also be used in Sect. 3 to establish connections between MSC languages and saturated slice languages.

We say that two slice strings $\mathbf{U}, \mathbf{U}' \in \overrightarrow{\boldsymbol{\Sigma}}(k,T)^*$ are locally $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$-equivalent, and denote this fact by $\mathbf{U} \overset{k}{\simeq} \mathbf{U}'$, if there exist $\mathbf{W}, \mathbf{W}' \in \overrightarrow{\boldsymbol{\Sigma}}(k,T)^*$ and $\mathbf{S}_1, \mathbf{S}_1', \mathbf{S}_2, \mathbf{S}_2' \in \overrightarrow{\boldsymbol{\Sigma}}(k,T)$ with $\mathbf{S}_1 \circ \mathbf{S}_2 = \mathbf{S}_1' \circ \mathbf{S}_2'$ such that $\mathbf{U} = \mathbf{W}\mathbf{S}_1\mathbf{S}_2\mathbf{W}'$ and $\mathbf{U}' = \mathbf{W}\mathbf{S}_1'\mathbf{S}_2'\mathbf{W}'$ (Fig. 5).

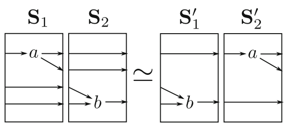

Fig. 5. Local Equivalence. $\mathbf{S}_1\mathbf{S}_2$ is 4-equivalent to $\mathbf{S}_1', \mathbf{S}_2'$.

We let $\overset{k}{\equiv} \subseteq \overrightarrow{\boldsymbol{\Sigma}}(k,T)^* \times \overrightarrow{\boldsymbol{\Sigma}}(k,T)^*$ be the equivalence relation defined on slice strings by taking the reflexive, symmetric and transitive closure of $\overset{k}{\simeq}$. We note that if $\mathbf{U}$ is a unit decomposition in $\overrightarrow{\boldsymbol{\Sigma}}(k,T)^{\circledast}$ then any slice string $\mathbf{U}'$ that is $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$-equivalent to $\mathbf{U}$ is also a unit decomposition in $\overrightarrow{\boldsymbol{\Sigma}}(k,T)^{\circledast}$, and additionally, $\mathring{\mathbf{U}}=\mathring{\mathbf{U}}'$. We note that there may exist unit decompositions in $\overrightarrow{\boldsymbol{\Sigma}}(k,T)^{\circledast}$ which are not $\overrightarrow{\boldsymbol{\Sigma}}(k,T)$-equivalent but which are $\overrightarrow{\boldsymbol{\Sigma}}(k',T)$-equivalent for some $k' > k$. Nevertheless, the following proposition states that for each k-coverable DAG H, $\overset{k}{\equiv}$-equivalence is already enough to relate any two unit decompositions of H.

Proposition 3. *Let $\mathbf{U}_1$ and $\mathbf{U}_2$ be unit decompositions in $\overrightarrow{\boldsymbol{\Sigma}}(k,T)^{\circledast}$ such that the DAGs $\mathring{\mathbf{U}}_1$ and $\mathring{\mathbf{U}}_2$ are k-coverable. Then $\mathring{\mathbf{U}}_1=\mathring{\mathbf{U}}_2$ if and only if $\mathbf{U}_1 \overset{k}{\equiv} \mathbf{U}_2$.*

There is a substantial difference between our notion of independence, defined on slice alphabets and the notion of independence in Mazurkiewicz trace theory. While the independence relation on slices is determined solely based on the structure of the slices (Fig. 5), without taking into consideration the events that label their center vertices, the Mazurkiewicz independence relation is defined directly on events. As a consequence, once an independence relation I is fixed, the nature of the partial-orders that can be represented as traces with respect to I is restricted. This is valid even for more general notions of traces, such as Diekert's semi-traces [8] and the context dependent traces of [20], in which for instance, partial-orders containing auto-concurrency[2] cannot be represented. In

[2] Auto-concurrency is the process of firing two transitions with the same label simultaneously.

our setting, any partial order ℓ labeled over a set of events T may be represented by a slice trace: namely the set of unit decompositions of its Hasse diagram.

Theorem 5. *Let $\mathcal{A}$ be a slice automaton over a slice alphabet $\overrightarrow{\mathbf{\Sigma}}(k,T)$ representing a set of k-partial-orders. Then we may effectively determine whether the slice language generated by $\mathcal{A}$ is weakly saturated.*

Proof. Assume without loss of generality that the slice automaton $\mathcal{A}$ is transitively reduced. Otherwise, just apply the transitive reduction algorithm from [34]. Since each partial-order $\ell \in \mathcal{L}_{po}(\mathcal{A})$ is a k-partial-order, the Hasse digram H of ℓ can be covered by k paths. Therefore, by Proposition 3, any unit decomposition of H has width at most k. Now let $tw(\mathcal{A})$ be automaton obtained from $\mathcal{A}$ by applying the twisting procedure in the proof of Lemma 2. Then by Proposition 2 the automaton $\mathcal{A}$ is weakly saturated if and only if $tw(\mathcal{A})$ is saturated. Therefore, it is enough to verify whether $tw(\mathcal{A})$ is saturated. With this in mind, it is enough to test the following condition. If a slice word $w\mathbf{S}_1\mathbf{S}_2u$ is generated by $\mathcal{A}'$ then every word $w\mathbf{S}_1'\mathbf{S}_2'u$ satisfying $\mathbf{S}_1' \circ \mathbf{S}_2' = \mathbf{S}_1 \circ \mathbf{S}_2$ is generated by $\mathcal{A}$ as well. Let $\mathcal{A}'$ be the minimal deterministic slice automaton generating the same slice language as $tw(\mathcal{A})$. Then any unit decomposition $\mathbf{S}_1\mathbf{S}_2\cdots\mathbf{S}_n \in \mathcal{L}(\mathcal{A}')$ corresponds to a unique computational path in $\mathcal{A}'$. Therefore to verify our condition, we just need to determine whether $\mathcal{A}'$ is "diamond" closed. In other words we need to test whether for each pair of transition rules $q\mathbf{S}_1r$ and $r\mathbf{S}_2q'$ of the $\mathcal{A}'$ and each unit decomposition $\mathbf{S}_1'\mathbf{S}_2'$ of $\mathbf{S}_1 \circ \mathbf{S}_2$, $\mathcal{A}'$ has a state r' and transitions $q\mathbf{S}_1'r'$ and $r'\mathbf{S}_2'q'$. Clearly this condition can be verified in polynomial time on the size of $\mathcal{A}'$, since $\mathbf{S}_1 \circ \mathbf{S}_2$ can have at most $2^{O(k\cdot\log k)}$ possible decompositions. □

7 From MSC Automata to Slice Automata

In this section we define the notion of locally-synchronized slice automata. Let $\mathbf{S}$ be a slice (possibly with several vertices in the center) with k in-frontier vertices $v_1,...,v_k$ and k out-frontier vertices $u_1,...,u_k$. For each $i \in \{1,...,k\}$, we say that a path $\mathfrak{p}_i$ from v_i to u_i is *trivial* if v_i and u_i are the only vertices in $\mathfrak{p}_i$. Let $\mathfrak{p}_1$,...,$\mathfrak{p}_k$ be paths such that for each i, $\mathfrak{p}_i$ is a path from v_i to u_i. We let the communication graph $\mathrm{comm}(\mathbf{S},\mathfrak{p}_1,...,\mathfrak{p}_k)$ be the directed graph whose vertices are paths in $\{\mathfrak{p}_1,...,\mathfrak{p}_k\}$, and such that for each $i,j \in \{1,...,k\}$, there is an edge from $\mathfrak{p}_i$ to $\mathfrak{p}_j$ if either these paths share a vertex or there is an edge with source in some vertex of $\mathfrak{p}_i$ and target in some vertex of $\mathfrak{p}_j$. We say that $\mathbf{S}$ is *locally-synchronized* $\mathrm{comm}(\mathbf{S},\mathfrak{p}_1,...,\mathfrak{p}_k)$ has at most one strongly connected component with more than one vertex. Note that trivial paths correspond to isolated vertices. This notion of local synchronization for slices, generalizes the notion of local synchronization for message sequence charts in the sense that processes correspond to paths, and isolated vertices in the communication graph of an MSC correspond to trivial paths.

Definition 6 (Locally-Synchronized Slice Automaton). *A slice automaton $\mathcal{A}$ is locally-synchronized if for every loop $q_1 \xrightarrow{\mathbf{S}_1} q_2 \xrightarrow{\mathbf{S}_2} \dots \xrightarrow{\mathbf{S}_{n-1}} q_n \xrightarrow{\mathbf{S}_n} q_1$ in $\mathcal{A}$, the slice $\mathbf{S}_1 \circ \mathbf{S}_2 \dots \circ \mathbf{S}_n$ is locally synchronized.*

The next theorem states that any locally-synchronized slice automaton can be transformed further into a saturated slice automaton representing the same partial order language as the original one.

Theorem 6. *Let T be a finite set of symbols, and $k \in \mathbb{N}$. Let $\mathcal{A}$ be a locally-synchronized slice automaton over $\overrightarrow{\boldsymbol{\Sigma}}(k, T)$. Then one can construct a saturated slice automaton $\mathcal{A}'$ such that $\mathcal{L}_{po}(\mathcal{A}') = \mathcal{L}_{po}(\mathcal{A})$.*

The following theorem is the main result of this section. It states that MSC automata can be converted into slice automata representing the same partial-order language. Additionally, this conversion transforms locally-synchronized MSC automata into saturated slice automata.

Theorem 7 (From MSC Automata to Slice Automata). *Let $\mathcal{M}$ be an MSC automaton over $\mathcal{J}$. Then one can construct a transitively-reduced slice automaton $\mathcal{A}(\mathcal{M})$ satisfying $\mathcal{L}_{po}(\mathcal{M}) = \mathcal{L}_{po}(\mathcal{A}(\mathcal{M}))$. Furthermore, if $\mathcal{M}$ is locally-synchronized, then $\mathcal{A}(\mathcal{M})$ is saturated.*

Proof. Let $\mathcal{J} = \{1, ..., k\}$ be a set of processes. We let $\mathbf{S}^{\iota}(\mathcal{J})$ be the slice with empty in-frontier $I = \emptyset$, k out-frontier vertices $O = \{v_1, ..., v_k\}$ where each v_i is labeled with the number i, and with a unique vertex v in the center which is connected to each vertex in O. We say that $\mathbf{S}^{\iota}(\mathcal{J})$ is the *initial slice* of $\mathcal{J}$.

Analogously, let $\mathbf{S}^{\varepsilon}(\mathcal{J})$ be the slice with empty out-frontier $O = \emptyset$, k in-frontier vertices $I = \{u_1, ..., u_k\}$ where each u_i is labeled with the number i, and with a unique center vertex v in the center. For each i there is an edge with source in u_i and target in v. We say that $\mathbf{S}^{\varepsilon}(\mathcal{J})$ is the *final slice* of $\mathcal{J}$.

Now let M be an MSC over $\mathcal{J}$. Then we let $\mathbf{S}(M)$ be the slice (not necessarily a unit slice) constructed as follows. $\mathbf{S}(M)$ has k in-frontier vertices $x_1, ..., x_k$, and k out-frontier vertices $y_1, ..., y_k$. Let H be the Hasse diagram of M. Then for each $i \in \{1, ..., k\}$ proceed as follows. If H has no vertex labeled with an element of Σ_i, then add an edge from the in-frontier vertex x_i to the out-frontier vertex y_i. Otherwise, if such a vertex exists, then add an edge from x_i to the (unique) minimal vertex of H labeled with an element of Σ_i, and an edge from the (unique) maximal vertex of H labeled with an element of Σ_i to the out-frontier vertex y_i. Note that the transitive closure of the slice $\mathbf{S}^{\iota}(\mathcal{J}) \circ \mathbf{S}(M) \circ \mathbf{S}^{\varepsilon}(\mathcal{J})$ is precisely the extension of the partial-order M (see Sect. 2). We let $\mathbf{W}(M) = \mathbf{S}_1\mathbf{S}_2...\mathbf{S}_n$ be an arbitrary sequence of unit slices such that $\mathbf{S}(M) = \mathbf{S}_1 \circ \mathbf{S}_2 \circ ... \circ \mathbf{S}_n$.

Now let $\mathcal{M}$ be an MSC automaton over $\mathcal{J}$. We will show how to construct a slice automaton $\mathcal{A}'(\mathcal{M})$ over $\overrightarrow{\boldsymbol{\Sigma}}(|\mathcal{J}|, \Sigma_{\mathcal{J}})$ such that $\mathcal{L}_{po}(\mathcal{A}'(\mathcal{M})) = \mathcal{L}_{po}(\mathcal{M})$. The conversion is done as follows. Let M be an MSC with m nodes and let $\mathbf{W}(M) = \mathbf{S}_1\mathbf{S}_2...\mathbf{S}_m$. We replace each transition (q, M, q') in $\mathcal{M}$ by a sequence of transitions

$$(q, \mathbf{S}_1, q_1)(q_1, \mathbf{S}_2, q_2)...(q_{n-1}, \mathbf{S}_n, q').$$

Now we create an initial state q_ι and add the transition $(q_\iota, \mathbf{S}^\iota(\mathcal{J}), q)$ for each initial state q of $\mathcal{M}$. Analogously, we create a final state q_ε and add the transition $(q, \mathbf{S}^\varepsilon(\mathcal{J}), q_\varepsilon)$ for each final state of $\mathcal{M}$. Now it is immediate to check that $\mathcal{M}$ accepts a sequence $M_1 M_2 ... M_n$ of $MSCs$ if and only if $\mathcal{A}'(\mathcal{M})$ accepts the unit decomposition

$$\mathbf{U} = \mathbf{S}^\iota(\mathcal{J})\mathbf{W}(M_1)\mathbf{W}(M_2)...\mathbf{W}(M_n)\mathbf{S}^\varepsilon(\mathcal{J}).$$

This implies that $\mathbf{tc}(\mathring{\mathbf{U}}) = \ell^*$ where $\ell = M_1 \circ M_2 \circ ... \circ M_n$. Therefore, $\mathcal{L}_{po}(\mathcal{A}'(\mathcal{M})) = \mathcal{L}_{po}(\mathcal{M})$.

Now assume that $\mathcal{L}_{po}(\mathcal{M})$ is linearization-regular. Then by Theorem 1, we may assume that $\mathcal{M}$ is locally-synchronized. Let $\mathbf{W}(M_1)\mathbf{W}(M_2)...\mathbf{W}(M_n)$ label a loop in $\mathcal{A}'(\mathcal{M})$. Then $M_1 M_2 ... M_n$ labels a loop in $\mathcal{M}$. Since by assumption $\mathcal{M}$ is locally-synchronized, we have that the MSC $M_1 \circ M_2 \circ ... \circ M_n$ is locally-synchronized. This implies that the slice $\mathbf{S}(M_1) \circ \mathbf{S}(M_2) \circ ... \circ \mathbf{S}(M_n)$ is also locally-synchronized. Since a sequence of MSCs $M_1 M_2 ... M_n$ labels a loop in $\mathcal{M}$ if and only if the sequence of unit slices $\mathbf{W}(M_1)\mathbf{W}(M_2)...\mathbf{W}(M_n)$ labels a loop in $\mathcal{A}'(\mathcal{M})$, we have that $\mathcal{A}'(\mathcal{M})$ is locally-synchronized. Therefore, as a last step we apply Theorem 6 to construct a slice automaton $\mathcal{A}(\mathcal{M})$ which is saturated and has the same partial-order language as $\mathcal{A}'(\mathcal{M})$, and therefore, the same partial-order language as $\mathcal{M}$. □

By combining Theorem 7 with Theorem 3 and Lemma 1, we have the following theorem.

Theorem 8. *Let $\mathcal{M}$ be an MSC automaton over $\mathcal{J}$ and N be b-bounded Petri net with transition set $T = \mathcal{J}$.*

1. *It is decidable whether $\mathcal{L}_{po}(\mathcal{M}) \cap \mathcal{P}_{cau}(N) \neq \emptyset$.*
2. *It is decidable whether $\mathcal{L}_{po}(\mathcal{M}) \subseteq \mathcal{P}_{cau}(N)$.*
3. *If $\mathcal{M}$ is locally synchronized, it is decidable whether $\mathcal{P}_{cau}(N) \subseteq \mathcal{L}_{po}(\mathcal{M})$.*

Proof. Let $\mathcal{A}$ be the slice automaton derived from $\mathcal{M}$ as in Theorem 7. Then $\mathcal{A}$ is transitively-reduced and $\mathcal{L}_{po}(\mathcal{M}) = \mathcal{L}_{po}(\mathcal{A})$. Let $\mathcal{A}'$ be the slice automaton constructed from N as in Theorem 3. Then $\mathcal{A}'$ is transitively-reduced, saturated and $\mathcal{L}_{po}(\mathcal{A}) = \mathcal{P}_{cau}(N)$. By Lemma 1, we have that it is decidable whether $\mathcal{L}_{po}(\mathcal{A}) \cap \mathcal{L}_{po}(\mathcal{A}') \neq \emptyset$ and whether $\mathcal{L}_{po}(\mathcal{A}) \subseteq \mathcal{L}_{po}(\mathcal{A}')$. Finally, if $\mathcal{M}$ is locally synchronized, then $\mathcal{A}$ is saturated, and therefore the inclusion $\mathcal{L}_{po}(\mathcal{A}') \subseteq \mathcal{L}_{po}(\mathcal{A})$ is also decidable. □

By combining Theorem 7 with Theorem 9 we have the following theorem.

Theorem 9 (Synthesis From MSC Automata). *Let $\mathcal{M}$ be an MSC automaton over $\mathcal{J}$. For each $b \in \mathbb{N}$ one can construct a b-bounded Petri net N satisfying the following properties.*

1. *$\mathcal{L}_{po}(\mathcal{M}) \subseteq \mathcal{P}_{cau}(N)$.*
2. *There is no other b-bounded Petri net N' with $\mathcal{L}_{po}(\mathcal{M}) \subseteq \mathcal{P}_{cau}(N') \subsetneq \mathcal{P}_{cau}(N)$.*

Additionally, if $\mathcal{M}$ is locally synchronized, then one can decide whether $\mathcal{P}_{cau}(N) = \mathcal{L}_{po}(\mathcal{M})$.

Proof. Let $\mathcal{A}$ be the slice automaton of Theorem 7. Then $\mathcal{A}$ is transitively-reduced and $\mathcal{L}_{po}(\mathcal{M}) = \mathcal{L}_{po}(\mathcal{A})$. By Theorem 4, one can construct a b-bounded Petri net N such that $\mathcal{L}_{po}(\mathcal{A}) \subseteq \mathcal{P}_{cau}(N)$, and such that there is no b-bounded Petri net N' with $\mathcal{L}_{po}(\mathcal{A}) \subseteq \mathcal{P}_{cau}(N') \subsetneq \mathcal{P}_{cau}(N)$.

Note that if there is a b-bounded Petri net whose causal behavior is equal to $\mathcal{L}_{po}(\mathcal{A})$, then by minimality, we have that $\mathcal{P}_{cau}(N) = \mathcal{L}_{po}(\mathcal{A})$. Nevertheless, in general it is not possible to verify whether equality is achieved. On the other hand, if $\mathcal{M}$ is locally synchronized, then by Item 3 of Theorem 8, one can also test whether the equality $\mathcal{P}_{cau}(N) \subseteq \mathcal{L}_{po}(\mathcal{A})$ holds. $\square$

8 From Mazurkiewicz Traces to Slice Languages

In Mazurkiewicz trace theory, partial-orders are represented as equivalence classes of words over an alphabet of events [28]. Given an alphabet T of events and a symmetric and anti-reflexive *independence relation* $I \subseteq T \times T$, a string $\alpha ab\beta$ is defined to be similar to the string $\alpha ba\beta$ ($\alpha ab\beta \simeq \alpha ba\beta$) provided $(a, b) \in I$. A trace is then an equivalence class of the transitive reflexive closure $\simeq^*$ of the relation $\simeq$. We denote by $[\alpha]_I$ the trace corresponding to a string $\alpha \in T^*$.

A partial-order $\ell_I(\alpha)$ is associated with a string $\alpha \in T^*$ of events in the following way: First we consider a dependence DAG $dep_I(\alpha) = (V, E, l)$ that has one vertex $v_i \in V$ labeled by the event α_i for each $i \in \{1, ..., |\alpha|\}$. An edge connects v_i to v_j in E if and only if $i < j$ and $(\alpha_i, \alpha_j) \notin I$. Then $\ell_I(\alpha)$ is the transitive closure of $dep_I(\alpha)$. One may verify that two strings induce the same partial-order if and only if they belong to the same trace. The trace language induced by a string language $\mathcal{L} \subseteq T^*$ with respect to an independence relation I is the set $[\mathcal{L}]_I = \{[\alpha]_I | \alpha \in \mathcal{L}\}$ and the trace closure of $\mathcal{L}$ is the language $\mathcal{L}^I = \cup_{\alpha \in \mathcal{L}} [\alpha]$.

Given a finite automaton $\mathcal{F}$ over an alphabet T and an independence relation $I \subset T \times T$, we denote by $\mathcal{L}(\mathcal{F})$ the regular language defined by $\mathcal{F}$ and by $\mathcal{L}_{po}(\mathcal{F}, \mathcal{I}) = \{\ell_I(\alpha) | \alpha \in \mathcal{L}(\mathcal{F})\}$ the partial-order language induced by $(\mathcal{F}, \mathcal{I})$. We call the pair $(\mathcal{F}, \mathcal{I})$ a Mazurkiewicz pair. We say that $\mathcal{L}(\mathcal{F})$ is trace-closed if $[\alpha]_I \subseteq \mathcal{L}(\mathcal{F})$ for each $\alpha \in \mathcal{L}(\mathcal{F})$. As an abuse of terminology, we may say that the Mazurkiewicz pair $(\mathcal{F}, \mathcal{I})$ is trace-closed.

We let $\hat{\mathcal{L}}_{po}(\mathcal{F}, \mathcal{I}) = \{\hat{\ell} \mid \ell \in \mathcal{L}_{po}(\mathcal{F}, \mathcal{I})\}$ be the set of extensions of partial-orders in $\mathcal{L}_{po}(\mathcal{F}, \mathcal{I})$. We note that extensions are only considered to make the construction of the automaton $\mathcal{A}(\mathcal{F}, \mathcal{I})$ slightly cleaner. With some easy case analysis one can construct slice automata whose partial-order language is $\mathcal{L}_{po}(\mathcal{F}, \mathcal{I})$ instead of $\hat{\mathcal{L}}_{po}(\mathcal{F}, \mathcal{I})$. The next theorem (Theorem 10) states that for any finite automaton $\mathcal{F}$ and independence relation I, one can construct a slice automaton $\mathcal{A}(\mathcal{F}, \mathcal{I})$ whose partial-order language is equal to $\hat{\mathcal{L}}_{po}(\mathcal{F}, \mathcal{I})$.

Theorem 10 (From Traces to Slices). *Let $\mathcal{F}$ be a finite automaton over an alphabet T, and $I \subset T \times T$ an independence relation. Then for some $k \leq |T|^2$ one can construct a transitively-reduced slice automaton $\mathcal{A}(\mathcal{F},\mathcal{I})$ over $\overrightarrow{\boldsymbol{\Sigma}}(k, T \cup \{\iota, \varepsilon\})$ such that $\mathcal{L}_{po}(\mathcal{A}(\mathcal{F},\mathcal{I})) = \hat{\mathcal{L}}_{po}(\mathcal{F},\mathcal{I})$. Additionally, if $(\mathcal{F},\mathcal{I})$ is trace-closed, then $\mathcal{A}(\mathcal{F},\mathcal{I})$ is saturated.*

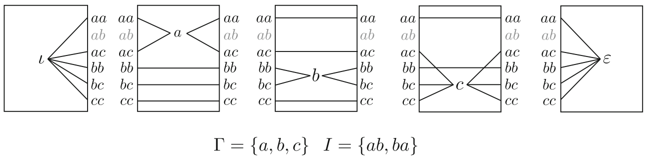

Fig. 6. Mapping an independence alphabet $(T,\mathcal{I})$ to a slice alphabet $\overrightarrow{\boldsymbol{\Sigma}}(T,\mathcal{I}) \subseteq \overrightarrow{\boldsymbol{\Sigma}}(k,T)$ where $k \leq |T|^2$.

In the remainder of this section we prove Theorem 10. We note that the difficulty in the construction of the automaton $\mathcal{A}(\mathcal{F},\mathcal{I})$ lies in showing that $\mathcal{A}(\mathcal{F},\mathcal{I})$ is saturated whenever $\mathcal{L}(\mathcal{F})$ is trace-closed. As a first step in the proof, we will use the independence alphabet $(T,\mathcal{I})$ to construct a slice alphabet $\overrightarrow{\boldsymbol{\Sigma}}(T,\mathcal{I}) = \{\mathbf{S}_a | a \in T\} \cup \{\mathbf{S}_\iota, \mathbf{S}_\varepsilon\}$ with the following property: For each string $\alpha = \alpha_1\alpha_2...\alpha_n \in T^*$ the partial-order defined by the unit decomposition $\mathbf{U}_\alpha = \mathbf{S}_\iota \mathbf{S}_{\alpha_1} \mathbf{S}_{\alpha_2} \cdots \mathbf{S}_{\alpha_n} \mathbf{S}_\varepsilon$ is precisely the extension of the partial-order $\ell_I(\alpha)$ induced by α (Fig. 6).

Let $\rho : T \rightarrow \{1, ..., |T|\}$ be an arbitrary ordering of the elements of T. Let $D = \{ab \mid a, b \in T,\ \rho(a) \leq \rho(b),\ (a,b) \notin I\}$ be the set of pairs of non-independent elements of T. Let $\overline{\rho} : D \rightarrow \{1, ..., |D|\}$ be the natural lexicographic ordering induced on D by the ordering ρ. For each symbol $a \in T$ we define the slice $\mathbf{S}_a$ as follows: Both the in-fronter I and the out-frontier O of $\mathbf{S}_a$ have $|D|$ vertices, and the center of $\mathbf{S}_a$ has a unique vertex v_a which is labeled by a. In symbols $I = \{I_{ab} | ab \in D\}$ and $O = \{O_{ab} | ab \in D\}$. For each $ab \in D$, both the in-frontier vertex I_{ab} and the out-frontier vertex O_{ab} are labeled with the number $\overline{\rho}(ab)$. For each pair $bc \in D$ with $a \neq b$ and $a \neq c$ we add an edge to $\mathbf{S}_a$ with source in I_{bc} and target in O_{bc}, and for each pair $ax \in D$ ($xa \in D$) we add an edge with source in I_{ax} (I_{xa}) and target in v_a, and an edge with source in v_a and target in O_{ax} (O_{xa}) (Fig. 6). We associate with the symbol ι an initial slice $\mathbf{S}_\iota$, with center vertex v_ι labeled by ι, and out-frontier O. Analogously, with the symbol ε, we associate a final slice $\mathbf{S}_\varepsilon$ with center vertex v_ε labeled by ε, and in-frontier I. We note that the slice alphabet $\overrightarrow{\boldsymbol{\Sigma}}(T,\mathcal{I})$ is a subset of $\overrightarrow{\boldsymbol{\Sigma}}(k, T \cup \{\iota, \varepsilon\})$ where $k = |D| \leq |T|^2$.

Now let $\alpha = \alpha_1\alpha_2...\alpha_n$ be a string in T, $dep_I(\alpha)$ be the dependence graph of α and $\mathbf{U}_\alpha = \mathbf{S}_\iota \mathbf{S}_{\alpha_1} ... \mathbf{S}_{\alpha_n} \mathbf{S}_\varepsilon$. Let v_i be the i-th vertex of $dep_I(\alpha)$, and u_i be the center vertex of the slice $\mathbf{S}_{\alpha_i}$. Then it is straightforward to check that for each $i, j \in \{1, ..., n\}$ with $i < j$, there is a path from u_i to u_j in the graph $\mathring{\mathbf{U}}_\alpha$ if and only if there is a path from v_i to v_j in $dep_I(\alpha)$. This implies that the partial-order $\mathbf{tc}(\mathring{\mathbf{U}}_\alpha)$ is the extension of the partial-order $\ell_I(\alpha)$ induced by α. In other words, $\mathbf{tc}(\mathring{\mathbf{U}}_\alpha) = \hat{\ell}_I(\alpha)$.

Now, from the pair $(\mathcal{F}, \mathcal{I}) = (Q, \mathfrak{R}, Q_0, F)$ we construct an auxiliary slice automaton $\mathcal{A}'(\mathcal{F}, \mathcal{I}) = (Q', \mathfrak{R}', Q_0', F')$ as follows. We let $Q' = Q \cup \{q_\iota, q_\varepsilon\}$, $Q_0' = \{q_\iota\}$, $F' = \{q_\varepsilon\}$, and $\mathfrak{R}' = \{(q_\iota, \mathbf{S}_\iota, q) \mid q \in Q_0\} \cup \{(q, \mathbf{S}_\varepsilon, q_\varepsilon) \mid q \in F\} \cup \{(q, \mathbf{S}_a, q') \mid (q, a, q') \in \mathfrak{R}\}$. Then we have that $\mathcal{F}$ accepts a string α if and only if $\mathcal{A}'(\mathcal{F}, \mathcal{I})$ accepts the unit decomposition $\mathbf{U}_\alpha$. This implies that $\mathcal{L}_{po}(\mathcal{A}'(\mathcal{F}, \mathcal{I})) = \hat{\mathcal{L}}_{po}(\mathcal{F}, \mathcal{I})$.

Now assume that $\mathcal{L}(\mathcal{F})$ is trace closed. Then for each string $\gamma \in [\alpha]_I$, we have that $\gamma \in \mathcal{L}(\mathcal{F})$ and therefore $\mathbf{U}_\gamma \in \mathcal{L}(\mathcal{A}(\mathcal{F}, \mathcal{I}))$. Since for each topological ordering ω of the graph $\mathring{\mathbf{U}}_\alpha$, there is a $\gamma \in [\alpha]_I$ such that $\mathbf{U}_\gamma$ is compatible with ω, we have that $\mathcal{A}'(\mathcal{F}, \mathcal{I})$ is weakly saturated. Therefore, by Lemma 2 we can construct a saturated slice automaton $\mathcal{A}(\mathcal{F}, \mathcal{I})$ with $\mathcal{L}_{po}(\mathcal{A}(\mathcal{F}, \mathcal{I})) = \mathcal{L}_{po}(\mathcal{A}'(\mathcal{F}, \mathcal{I}))$. $\square$

9 Conclusion

In this work, we have established connections between the causal semantics of Petri nets and message sequence chart languages. In particular, we showed that message sequence chart automata can be used as a tool for the study of the causal behavior of Petri nets. Despite the fact that each of these formalisms have been defined several decades ago, the connections established in our work were unknown. In order to prove our results we have introduced new slice theoretic machinery of independent interest. In particular, our techniques pave the way for the use of slice automata as a bridge between bounded Petri nets and behavioral formalisms. Further evidence for this assessment is given in Sect. 8, where we show how to map Mazurkiewicz Trace languages to trace languages in such a way that trace closure implies saturation. This means that the results in Theorems 8 and 9 also hold if instead of MSC automata we use Mazurkiewicz pairs.

References

1. Adsul, B., Mukund, M., Kumar, K.N., Narayanan, V.: Causal closure for MSC languages. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 335–347. Springer, Heidelberg (2005). https://doi.org/10.1007/11590156_27
2. Alur, R., Yannakakis, M.: Model checking of message sequence charts. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 114–129. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48320-9_10
3. Bauderon, M., Courcelle, B.: Graph expressions and graph rewritings. Math. Syst. Theory **20**(2–3), 83–127 (1987)
4. Bollig, B., Kuske, D.: Muller message-passing automata and logics. Inf. Comput. **206**(9–10), 1084–1094 (2008)
5. Bollig, B., Leucker, M.: Message-passing automata are expressively equivalent to EMSO logic. Theor. Comput. Sci. **358**(2–3), 150–172 (2006)
6. Brandenburg, F.J., Skodinis, K.: Finite graph automata for linear and boundary graph languages. Theor. Comput. Sci. **332**(1–3), 199–232 (2005)

7. Bruggink, H.J.S., König, B.: On the recognizability of arrow and graph languages. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 336–350. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87405-8_23
8. Diekert, V.: A partial trace semantics for Petri Nets. Theor. Comput. Sci. **134**(1), 87–105 (1994)
9. Droste, M.: Concurrent automata and domains. Int. J. Found. Comput. Sci. **3**(4), 389–418 (1992)
10. Droste, M., Gastin, P., Kuske, D.: Asynchronous cellular automata for pomsets. Theor. Comput. Sci. **247**(1–2), 1–38 (2000)
11. Engelfriet, J., Vereijken, J.J.: Context-free graph grammars and concatenation of graphs. Acta Informatica **34**, 773–803 (1997)
12. Fanchon, J., Morin, R.: Pomset languages of finite step transition systems. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, pp. 83–102. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02424-5_7
13. Gaifman, H., Pratt, V.R.: Partial order models of concurrency and the computation of functions. In: Proceedings of the 2nd Symposium on Logic in Computer Science (LICS 1987), pp. 72–85 (1987)
14. Genest, B., Kuske, D., Muscholl, A.: A Kleene theorem and model checking algorithms for existentially bounded communicating automata. Inf. Comput. **204**(6), 920–956 (2006)
15. Genest, B., Kuske, D., Muscholl, A.: On communicating automata with bounded channels. Fundamenta Informaticae **80**(1–3), 147–167 (2007)
16. Genest, B., Muscholl, A., Seidl, H., Zeitoun, M.: Infinite-state high-level MSCS: model-checking and realizability. J. Comput. Syst. Sci. **72**(4), 617–647 (2006)
17. Gischer, J.L.: The equational theory of pomsets. Theor. Comput. Sci. **61**, 199–224 (1988)
18. Goltz, U., Reisig, W.: Processes of place/transition-nets. In: Diaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 264–277. Springer, Heidelberg (1983). https://doi.org/10.1007/BFb0036914
19. Henriksen, J.G., Mukund, M., Kumar, K.N., Sohoni, M., Thiagarajan, P.: A theory of regular MSC languages. Inf. Comput. **202**(1), 1–38 (2005)
20. Hoogers, P., Kleijn, H., Thiagarajan, P.: A trace semantics for Petri Nets. Inf. Comput. **117**(1), 98–114 (1995)
21. Jategaonkar Jagadeesan, L., Jagadeesan, R.: Causality and true concurrency: a data-flow analysis of the Pi-Calculus. In: Alagar, V.S., Nivat, M. (eds.) AMAST 1995. LNCS, vol. 936, pp. 277–291. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60043-4_59
22. Kuske, D.: Contributions to a trace theory beyond Mazurkiewicz traces (2000)
23. Kuske, D.: Regular sets of infinite message sequence charts. Inf. Comput. **187**(1), 80–109 (2003)
24. Kuske, D., Morin, R.: Pomsets for local trace languages. J. Automata Lang. Combinatorics **7**(2), 187–224 (2002)
25. Langerak, R., Brinksma, E., Katoen, J.-P.: Causal ambiguity and partial orders in event structures. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 317–331. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63141-0_22
26. Lodaya, K., Weil, P.: Series-parallel languages and the bounded-width property. Theor. Comput. Sci. **237**(1–2), 347–380 (2000)

27. Madhusudan, P., Meenakshi, B.: Beyond message sequence graphs. In: Hariharan, R., Vinay, V., Mukund, M. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 256–267. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45294-X_22
28. Mazurkiewicz, A.: Trace theory. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) ACPN 1986. LNCS, vol. 255, pp. 278–324. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-17906-2_30
29. Montanari, U., Pistore, M.: Minimal transition systems for history-preserving bisimulation. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 413–425. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0023477
30. Morin, R.: On regular message sequence chart languages and relationships to Mazurkiewicz trace theory. In: Honsell, F., Miculan, M. (eds.) FoSSaCS 2001. LNCS, vol. 2030, pp. 332–346. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45315-6_22
31. Muscholl, A., Peled, D.: Message sequence graphs and decision problems on Mazurkiewicz traces. In: Kutyłowski, M., Pacholski, L., Wierzbicki, T. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 81–91. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48340-3_8
32. Muscholl, A., Peled, D., Su, Z.: Deciding properties for message sequence charts. In: Nivat, M. (ed.) FoSSaCS 1998. LNCS, vol. 1378, pp. 226–242. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0053553
33. de Oliveira Oliveira, M.: Hasse diagram generators and Petri Nets. Fundamenta Informaticae **105**(3), 263–289 (2010)
34. Oliveira Oliveira, M.: Canonizable partial order generators. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 445–457. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28332-1_38
35. Thomas, W.: Finite-state recognizability of graph properties. Theorie des Automates et Applications **172**, 147–159 (1992)
36. Vogler, W. (ed.): Modular Construction and Partial Order Semantics of Petri Nets. LNCS, vol. 625. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55767-9
37. Zielonka, W.: Notes on finite asynchronous automata. RAIRO-Theor. Inform. Appl. **21**(2), 99–135 (1987)