Aggregating Causal Runs into Workflow Nets

Boudewijn F. van Dongen[1], Jörg Desel[2], and Wil M.P. van der Aalst[1]

[1] Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, The Netherlands
{B.F.v.Dongen,W.M.P.v.d.Aalst}@tue.nl
[2] Department of Software Engineering,
FernUniversität in Hagen, Germany
joerg.desel@fernuni-hagen.de

Abstract. This paper provides three aggregation algorithms for deriving system nets from sets of partially-ordered causal runs. The three algorithms differ with respect to the assumptions about the information contained in the causal runs. Specifically, we look at the situations where labels of conditions (i.e. references to places) or events (i.e. references to transitions) are unknown. Since the paper focuses on aggregation in the context of process mining, we solely look at workflow nets, i.e. a class of Petri nets with unique start and end places. The difference of the work presented here and most work on process mining is the assumption that events are logged as partial orders instead of linear traces. Although the work is inspired by applications in the process mining and workflow domains, the results are generic and can be applied in other application domains.

1 Introduction

This paper proposes different approaches to "discover" process models from observed runs, i.e., runs (also known as causal nets or occurrence nets, cf. [14]) are aggregated into a single Petri net that captures the observed behavior. Runs provide information about events together with pre- and post-conditions which constitute a (partial) order between these events. This is useful in many domains where processes are studied based on their recorded behavior, such as:

- Discovering administrative processes by following the document flows in the organization with the goal to improve efficiency.
- Auditing processes in organizations in order to make sure that they conform to some predefined rules.
- Constructing enterprise models by observing transaction logs or document flows in enterprise systems such as SAP, Peoplesoft and Oracle.
- Monitoring the flow of SOAP messages between web-services to see how different services interact.
- Observing patient flows in hospitals to improve careflows and to verify medical guidelines.

K. Jensen et al. (Eds.): ToPNoC VI, LNCS 7400, pp. 334–363, 2012.

There are many techniques to discover process models based on sequential event logs (also known as transaction logs, audit trails, etc). People working on process mining techniques [6] generally tackle situations where processes may be concurrent and the set of observations is incomplete. Especially since the set of possible sequences is typically larger than the number of process instances, it is unrealistic to assume that all possible sequences have been observed.

In many applications, event logs are assumed to be linear, for example since all events are ordered in time. However, there are many processes where it is possible to monitor causal dependencies (e.g., by analyzing the dataflows). In the examples mentioned before, it is easy to identify situations where activities are causally linked by documents or explicit messages which can be monitored and hence explicit information about the causal dependencies between events is available. Consider for example service-oriented systems where one service calls another service. These services have input and output data. Using these dataflow one can find explicit causal dependencies. Furthermore, we encountered several Business Process Management (BPM) systems that actually log behavior using a representation similar to runs. The ad-hoc workflow management system InConcert of Tibco (formerly Xerox) allows end users to define and modify process instances (e.g., customer orders) while capturing the causal dependencies between the various activities. The representation used by these systems directly corresponds to the notion of runs. The analysis tool ARIS PPM (Process Performance Monitor) of IDS Scheer can extract runs represented as so-called *instance EPCs* (Event-driven Process Chains) from systems such as SAP R/3 and Staffware. These examples show that in real-life systems and processes runs can be recorded or already are being recorded, thus motivating the work presented in this contribution.

The remainder of this paper is structured as follows. After discussing related work in Section 2 and some preliminary definitions in Section 3, we provide algorithms for the aggregation of runs. In Section 4, three algorithms are presented for the aggregation of runs for the situations depicted in Figures 1 to 3. Figures 1 to 3 each show two runs on the left-hand side and the most likely candidate for the aggregated model on the right hand side. The first algorithm we present assumes we have full knowledge of each event, its preconditions and its postconditions. This is shown in Figure 1, where all events and conditions are labeled and these labels uniquely identify the corresponding transition or place in the aggregated model. Then, we assume that we cannot uniquely identify events, i.e. the label of an event may refer to multiple transitions, as shown in Figure 2, where *send_goods* and *send_bill* cannot be distinguished, since both of them are recorded as *send_something*. In the aggregated model however, two occurrences of the transition *send_something* have been identified. Finally, we provide an algorithm that assumes less knowledge about pre- and post-conditions, as shown in Figure 3, where no conditions have labels, while the corresponding aggregated model shows the same structure as in Figure 1. In Section 5, we formally prove that the algorithms we presented in Section 4 are correct, i.e. that the aggregated nets can reproduce the original causal nets. We conclude the paper in Section 6.

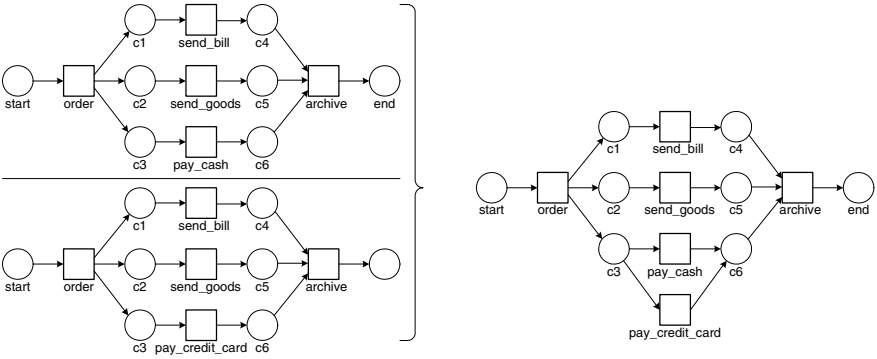


Fig. 1. Example of aggregating runs with known event and condition labels (Section 4.1)

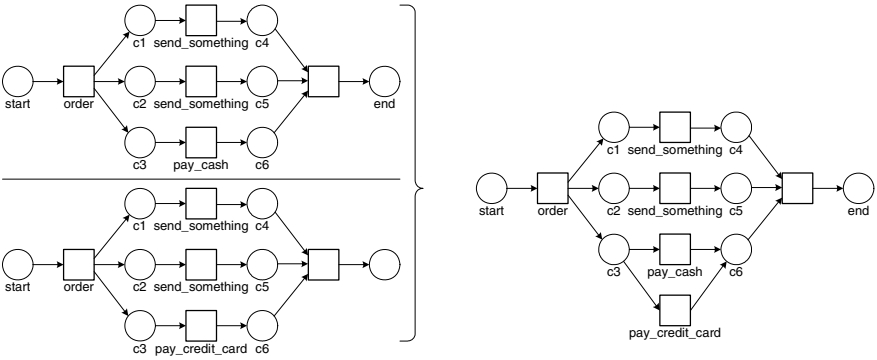


Fig. 2. Example of aggregating runs with known condition labels and unknown or non-unique event labels (Section 4.2)

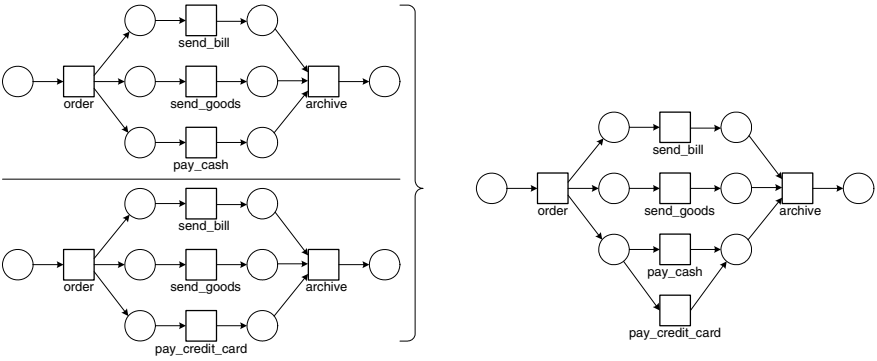


Fig. 3. Example of aggregating runs with known event labels and unknown condition labels (Section 4.3)

2 Related Work

For an extensive overview of the process mining domain, we refer to the recent book on process mining [2].

Since the mid-nineties several groups have been working on techniques for automated process discovery based on event logs [5, 7, 8, 12, 13, 20, 28, 29]. In [6] an overview is given of the early work in this domain. The idea to apply process mining in the context of workflow management systems was introduced in [8]. In parallel, Datta [13] looked at the discovery of business process models. Cook et al. investigated similar issues in the context of software engineering processes [12]. Herbst [22] was one of the first to tackle more complicated processes, e.g., processes containing duplicate tasks. Most of the classical approaches have problems dealing with concurrency. The α-algorithm [7] is an example of a simple technique that takes concurrency as a starting point. However, this simple algorithm has problems dealing with complicated routing constructs and noise (like most of the other approaches described in literature).

In all of the algorithms mentioned above, these tasks (i.e., events) in each case are *totally ordered* (typically based on the timestamps). In this paper, we take a different approach. We start by looking at so-called runs. These runs are a *partial* ordering on the tasks within each case. However, in addition to the partial ordering of tasks, we may have information about the local states of the system from which the logs originated, i.e. for each event the pre- and post-conditions are known. This closely relates to the process mining algorithms presented in [17] and [18]. However, also in these papers only causal dependencies between events are considered and no state information is assumed to be known.

The generation of system nets from their causal runs has been investigated before. The first publication on this topic is [27]. Here the basis is assumed to be the set of all runs. These runs are folded, i.e., events representing the occurrence of the same transition are identified, and so are conditions representing a token on the same place. In [15] a similar folding approach is taken, but there the authors start with a set of causal runs, as we do in the present paper. [15] does not present algorithms in details for the aggregation of runs but rather concentrates on correctness criteria for the derived system net. [11] presents an aggregation algorithm that constructs event-driven process chains from sets of partially ordered sets of events (without conditions).

The problem tackled in this paper is closely related to the so-called synthesis problem of Petri nets (see [16] and [19] for the synthesis of elementary net systems and [9] for more general cases). In this work, the behavior is given in the form of state graphs (where the events are known but the states are anonymous). In process mining, the observed behavior is not complete and it is not known, which process executions lead to identical global states. More recently, [26] extracts Petri nets from models which are based on Message Sequence Charts (MSCs), a concept quite similar to causal runs. Less related is the work presented in [21], where a special variant of MSCs is used to generate a system implementation.

In [24], so-called regions are defined for partial orders of events representing runs. These regions correspond to anonymous places of a synthesized place/

transition net, which can generate these partial orders. In contrast to our work, the considered partial orders are any linearizations of causal orders, i.e., two ordered events can either occur in a sequence (then there is a causal run with a condition "between" the events) or they can occur concurrently. Consequently, conditions representing tokens on places are not considered in these partial orders whereas our approach heavily depends on these conditions. More recently, this region-based approach was used for the synthesis of place/transition nets from sets of finite [10] or infinite [11] partially ordered sets of events.

3 Preliminaries

In this section, we introduce some basic definitions used in the remainder of this paper and formalize the starting point for the aggregation of partially ordered runs. Typically, a partial order is represented by a graph, and therefore we introduce some concepts related to graphs, such as a complete subgraph and a graph coloring. A graph-coloring is a way to label the nodes of a graph in such a way that no two neighboring nodes (i.e. nodes connected by an edge) have the same color.

Definition 3.1. (Graphs)
Let $G = (N, E)$ be a directed graph, i.e. N is the set of nodes and $E \subseteq N \times N$ is the set of edges. If $N' \subseteq N$, we say that $G' = (N', E \cap (N' \times N'))$ is a subgraph of G. G is a *complete* graph if and only if $E = (N \times N)$.

In the sequel, we assume $G = (N, E)$ is a directed graph.

Definition 3.2. (Undirected path in a graph)
Let $a \in N$ and $b \in N$. We define an undirected path from a to b as a sequence of nodes denoted by $< n_1, \ldots, n_k >$ with $k \geq 1$ such that $n_1 = a$ and $n_k = b$ and $\forall_{i \in \{1 \ldots k-1\}}((n_i, n_{i+1}) \in E \ \vee \ (n_{i+1}, n_i) \in E)$.

Definition 3.3. (Connected graph)
G is a *connected* graph if for all $n_1, n_2 \in N$ holds that there is an undirected path from n_1 to n_2. A set of vertices $N' \subseteq N$ generates a *maximal connected subgraph* if it is a maximal set of vertices generating a connected subgraph.

Definition 3.4. (Graph coloring)
Let μ be a set of colors. A function $f : N \to \mu$ is a coloring function if, for all $(n_1, n_2) \in E$, either $n_1 = n_2$ or $f(n_1) \neq f(n_2)$.

Lemma 3.5. (Colorings on subgraphs can be combined)
Let $E_1, E_2 \subseteq E$, such that $E_1 \cup E_2 = E$. Furthermore, let $f : N \to \mu$ be a coloring function on the graph (N, E_1) as well as a coloring function on the graph (N, E_2). Then f is also a coloring function on G.

Proof. Let $(n_1, n_2) \in E$ and $n_1 \neq n_2$. Since $E = E_1 \cup E_2$, we either have $(n_1, n_2) \in E_1$ or $(n_1, n_2) \in E_2$. Since f is a coloring function on both (N, E_1) and (N, E_2), $f(n_1) \neq f(n_2)$. □

In graphs, we would like to be able to talk about predecessors and successors of nodes. Therefore, we introduce a special notation for that.

Definition 3.6. (Pre-set and post-set)
Let $n \in N$. We define ${}^{G}\!\bullet n = \{m \in N \mid (m, n) \in E\}$ as the pre-set and $n^{G}\!\bullet = \{m \in N \mid (n, m) \in E\}$ as the post-set of n with respect to the graph G. If the context is clear, the superscript G may be omitted, resulting in $\bullet n$ and $n\bullet$.

As stated in the introduction, our starting point is not only a partial order of events within a case, but also information about the state of a case. Since we want to be able to represent both events and states, Petri nets provide a natural basis for our approach. In this paper, we use the standard definition of finite marked place/transition (P/T-nets) nets $N = (P, T, F, M_0)$.

Definition 3.7. (Bag)
Let S be a set. A *bag* over S is a function from S to the natural numbers $\mathbb{N}$.

Definition 3.8. (Place/Transition net)
$N = (P, T, F, M_0)$ is a marked place/transition net (or P/T-net) if:

- P is a finite set of places,
- T is a finite, non-empty set of transitions, such that $P \cap T = \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation of the net,
- $M_0 : P \to \mathbb{N}$ represents the initial marking of the net, where a marking is a bag over the set of places P.

Note that any P/T-net $N = (P, T, F, M_0)$ defines a directed graph $((P \cup T), F)$. In this paper, we restrict ourselves to P/T-nets where for each transition t holds that $\bullet t \neq \emptyset$ and $t\bullet \neq \emptyset$.

Definition 3.9. (Bag notations)
We use square brackets for the enumeration of the elements of a bag representing a marking of a P/T-net. The sum of two bags ($X \uplus Y$), the presence of an element in a bag ($a \in X$), and the notion of subbags ($X \leq Y$) are defined in a straightforward way, and they can handle a mixture of sets and bags. Furthermore, $\biguplus_{a \in A} \big(f(a)\big)$ denotes the sum over the bags that are results of function f applied to the elements a of a bag A.

Petri nets specify processes. The behavior of a Petri net is given in terms of causal nets, representing process instances (i.e. cases). Therefore, we introduce some concepts (notation taken from [14]). First, we introduce the notion of a causal net, this is a specification of one process instance.

Definition 3.10. (Causal net)
The P/T-net (C, E, K, S_0) is called a causal net if:

- for every place $c \in C$ holds that $|\bullet c| \leq 1$ and $|c \bullet| \leq 1$,
- the transitive closure of K is irreflexive, i.e. it is a partial order on $C \cup E$,
- for each place $c \in C$ holds that $S_0(c) = 1$ if $\bullet c = \emptyset$ and $S_0(c) = 0$ if $\bullet c \neq \emptyset$.

In causal nets, we refer to places as *conditions* and to transitions as *events*.

Each event of a causal net should refer to a transition of a corresponding P/T-net and each condition should refer to a token on some place of the P/T-net. These references are made by mapping the conditions and the events of a causal net onto places and transitions, respectively, of a Petri net. We call the combination of a causal net and such a mapping a *run*.

Definition 3.11. (Run)
A run (N, α, β) of a P/T-net (P, T, F, M_0) is a causal net $N = (C, E, K, S_0)$, together with two mappings $\alpha : C \to P$ and $\beta : E \to T$, such that:

- For each event (transition) $e \in E$, the mapping α induces a bijection from $\bullet e$ to $\bullet\beta(e)$ and a bijection from $e\bullet$ to $\beta(e)\bullet$,
- $\alpha(S_0) = M_0$ where α is generalized to markings by $\alpha : (C \to \mathbb{N}) \to (P \to \mathbb{N})$, such that $\alpha(S_0)(p) = \sum_{c|\alpha(c)=p} S_0(c)$.

The causal behavior of the P/T-net (P, T, F, M_0) is defined as its set of runs. To avoid confusion, the P/T-net (P, T, F, M_0) is called *system net* in the sequel.

In this paper, we take a set of runs as a starting point. From these runs, we generate a system net describing the behavior of all individual runs. Remember that we do not assume to have all runs as a starting point.

4 Aggregation of Runs

In this section, we introduce an approach that takes a set of runs as a starting point. From this set of runs, a system net is constructed. Moreover, we need to find a mapping from all the events and conditions in the causal nets to the transitions and places in the system net. From Definition 3.11, we know that there should exist a bijection between all conditions in the pre- or post-set of an event in the causal net and the pre- or post-set of a transition in a system net. *Therefore, two conditions belonging to the pre- or post-set of a single event should not be mapped onto the same place.* This restriction is in fact merely another way to express the fact that our P/T-nets do not allow for more than one edge between a place and a transition or vice versa. More generally, we define a labeling function on the nodes of a graph as a function that does not give the same label to two nodes that have a common element in their pre-sets or a common element in their post-sets.

Definition 4.1. (Labeling function)
Let μ be a set of labels. Let $G = (N, E)$ be a graph. Let $R = \{(n_1, n_2) \subseteq N \times N \mid n_1 {}^{G}\!\bullet \cap n_2 {}^{G}\!\bullet \neq \emptyset \vee {}^{G}\!\bullet n_1 \cap {}^{G}\!\bullet n_2 \neq \emptyset\}$. We define $f : N \to \mu$ to be a *labeling function* if f is a coloring function on the graph (N, R).

We focus on the aggregation of runs that originate from a Petri net with clearly defined starting state and completion state, i.e. processes that describe a lifespan of some case. This assumption is very natural in the context of workflow management systems. However, it applies to many other domains where processes are instantiated for specific cases. Hence, we will limit ourselves to a special class of Petri nets, namely workflow nets.

Definition 4.2. (Workflow nets)
A P/T-net $N = (P, T, F, M_0)$ is a *workflow net* (WF-net) if:

1. P contains an input place p_{ini} such that $\bullet p_{ini} = \emptyset$,
2. P contains an output place p_{out} such that $p_{out}\bullet = \emptyset$,
3. there is a path from p_{ini} to every node and a path from every node to p_{out},
4. $M_0 = [p_{ini}]$, i.e. the initial marking marks only p_{ini}.

As a consequence, a WF-net has exactly one input place. When looking at a run of a WF-net, we can therefore conclude that there is exactly one condition containing a token initially and all other conditions do not contain tokens. A set of causal nets fulfilling this condition and some structural consequences is called a *causal set*.

Definition 4.3. (Causal set)
Let $n \in \mathbb{N}$ and let $\Phi = \{(C_i, E_i, K_i, S_i) \mid 0 \leq i < n\}$ be a set of causal nets. We call this set a *causal set* if the sets C_i, E_i and K_i are pairwise disjoint and, for $0 \leq i < n$ holds:

- $\sum_{c \in C_i} S_i(c) = 1$, i.e. exactly one condition has an empty pre-set,
- If for some $c \in C_i$, holds that $S_i(c) = 1$ and $e \in c\bullet$, then $\{c\} = \bullet e$, i.e. each event in the postset of an initially marked condition has only this condition in its preset,
- If for some $c \in C_i$, holds that $c\bullet = \emptyset$ and $e \in \bullet c$, then $e\bullet = \{c\}$, i.e. each event in the preset of a condition with empty postset (representing a token on the place p_{out}) has only this condition in its postset.

The concept of constructing a system net from a causal set is called *aggregation*. This concept can be applied if we assume that each causal net in the given set can be called a run of some system net. From Definition 3.11 we know that we need two mappings α and β satisfying the two properties mentioned. Using the definition of a system net and the relation between system nets and runs, we can conclude that any aggregation algorithm should have the following functionality:

- it should provide the set of places P of the system net,
- it should provide the set of transitions T of the system net,
- it should provide the flow relation F of the system net,
- it should provide the initial marking M_0 of the system net,
- for each causal net in the causal set, it should provide the mappings $\alpha_i : C_i \to P$ and $\beta_i : E_i \to T$, in such a way that for all causal nets, $\alpha_i(S_i)$ is the same (i.e. they have the same initial marking) and they induce bijections between pre- and post-sets of events and their corresponding transitions.

Each event that appears in a causal net has a corresponding transition in the original system net. Moreover, bijections exist between the pre- and post-sets of this event and the corresponding transitions. In order to express this in terms of labeling functions of causal nets, we formalize this concept using the notion of *transition equivalence*.

Definition 4.4. (Transition equivalence)
Let μ, ν be two disjoint sets of labels. Let $\Phi = \{N_i = (C_i, E_i, K_i, S_i) \mid 0 \leq i < n\}$ be a causal set, and let $\Psi = \{(\alpha_i : C_i \to \mu, \beta_i : E_i \to \nu) \mid 0 \leq i < n\}$ be a corresponding set of labeling functions for each (C_i, E_i, K_i, S_i). We define (Φ, Ψ) to *respect transition equivalence* if and only if for each $e_i \in E_i$ and $e_j \in E_j$ with $\beta_i(e_i) = \beta_j(e_j)$ the following holds:

- for each $(c_i, e_i) \in K_i$ there is a $(c_j, e_j) \in K_j$ such that $\alpha_i(c_i) = \alpha_j(c_j)$,
- for each $(e_i, c_i) \in K_i$ there is a $(e_j, c_j) \in K_j$ such that $\alpha_i(c_i) = \alpha_j(c_j)$.

Using the concepts of a causal set and transition equivalence, we introduce three aggregation algorithms with different requirements on the available information. First, in Section 4.1 we introduce an algorithm to aggregate causal nets where all places and transitions have known labels. Then, in Section 4.2, we show an algorithm that can deal with the situation where different transitions have the same label. The final algorithm, presented in Section 4.3, deals with the situation where transitions are correctly labeled, but places are not labeled at all.

4.1 Aggregation with Known Labels

In this section, we present an aggregation algorithm that assumes that we know all mapping functions, and that these mapping functions adhere to the definition of a run. To illustrate the aggregation process, we make use of a running example. Consider Figure 4 where four parts of runs are shown. We assume that the events *A*,*B*,*C*,*D*,*E*,*F* and *G* do not appear in any other part of each run.

Our first aggregation algorithm is called *ALK* (short for "All Labels Known"). This algorithm assumes known labels for events and known labels for conditions, such as in Figure 4. These labels refer to concrete transitions and places in the aggregated system net.

Definition 4.5. (ALK aggregation algorithm)
Let μ, ν be two disjoint sets of labels. Let Φ be a causal set of size n with causal nets (C_i, E_i, K_i, S_i) $(0 \leq i < n)$.

Furthermore, let $\{(\alpha_i : C_i \to \mu, \beta_i : E_i \to \nu) \mid 0 \leq i < n\}$ be a set of labeling functions respecting transition equivalence, such that for all causal nets $\alpha_i(S_i)$ is the same. We construct the system net (P, T, F, M_0) belonging to these runs as follows:

- $P = \bigcup_{0 \leq i < n} \mathrm{rng}(\alpha_i)$ is the set of places (note that $P \subseteq \mu$)[1],
- $T = \bigcup_{0 \leq i < n} \mathrm{rng}(\beta_i)$ is the set of transitions (note that $T \subseteq \nu$),
- $F = \bigcup_{0 \leq i < n} \{(\alpha_i(c), \beta_i(e)) \in P \times T \mid (c, e) \in K_i \cap (C_i \times E_i)\} \cup \bigcup_{0 \leq i < n} \{(\beta_i(e), \alpha_i(c)) \in T \times P \mid (e, c) \in K_i \cap (E_i \times C_i)\}$ is the flow relation,
- $M_0 = \alpha_0(S_0)$ is the initial marking.

[1] With rng we denote the range of a function, i.e. $\mathrm{rng}(f) = \{f(x) \mid x \in \mathrm{dom}(f)\}$.

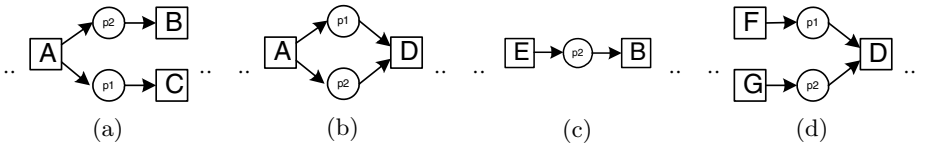


Fig. 4. Four examples of parts of runs

The result of the ALK aggregation algorithm applied to the parts presented in Figure 4 is shown in Figure 5. Another example is given in Figure 1.

The aggregated net shown in Figure 5 can generate the runs of Figure 4. However, it also allows for the possibility to execute transitions F followed by C. The token flow from F to C through place $p1$ was never directly observed in any of the runs. Nonetheless, from the run in Figure 4(a) we can see that the C can fire using a token from $p1$ and from the run in Figure 4(d) we can derive that transition F indeed produces this token, hence no "new" behavior is introduced.

The ALK algorithm is a rather trivial aggregation over a set of runs. Although we prove its correctness in Section 5.1, the algorithm relies on the assumption that the mapping functions α_i and β_i are known for each causal net. Furthermore, we assume two sets of labels μ and ν to be known. However, when applying these techniques in the context of process mining, it is often not realistic to assume that all of these are present. Therefore, in the remainder of this paper, we relax some of these assumptions to obtain more usable aggregation algorithms for process mining.

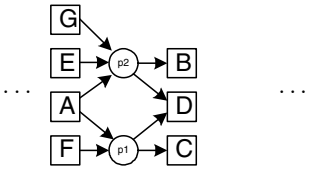


Fig. 5. The aggregated Petri net

4.2 Aggregation with Duplicate or Missing Transition Labels

In this section, we assume that the causal set used to generate the system net and the labeling functions do not respect transition equivalence (Definition 4.4). We introduce an algorithm to change the labeling function for events in such a way that this property holds again. In the domain of process mining, the problem of so-called "duplicate transitions" (i.e. several transitions with the same label) is well-known (cf. [3, 23, 25]). Therefore, there is a need for algorithms to find out which events actually belong to which transition. We assume that we have causal nets with labeling functions, where some events have the same label, even though they may refer to different transitions (see Figure 6). Note that this

figure is similar to Figure 4, except that we now labeled the events previously labeled with F and G with a new label X.

Since the previous aggregation algorithm given in Definition 4.5 assumes that transition equivalence holds, we provide an algorithm to redefine the labeling functions for events it this is not the case.

Definition 4.6. (Relabeling algorithm)
Let μ, ν be two disjoint sets of labels. Let $\Phi = \{N_i \mid N_i = (C_i, E_i, K_i, S_i) \wedge 0 \leq i < n\}$ be a causal set and let $\Psi = \{(\alpha_i : C_i \rightarrow \mu, \beta_i : E_i \rightarrow \nu) \mid 0 \leq i < n\}$ be a set of labeling functions in (C_i, E_i, K_i, S_i) such that $\alpha_i(S_i)$ is the same for all causal nets. Furthermore, assume that μ and ν are minimal, i.e. $\bigcup_{0 \leq i < n} \text{rng}(\alpha_i) = \mu$ and $\bigcup_{0 \leq i < n} \text{rng}(\beta_i) = \nu$. Let $E^\star = \bigcup_{0 \leq i < n} E_i$ be the set of all events in the causal set.

We define the relabeling algorithm as follows:

1. Define $\bowtie \subseteq E^\star \times E^\star$ as an equivalence relation on the elements of $E^\star$ in such a way that $e_i \bowtie e_j$ with $e_i \in E_i$ and $e_j \in E_j$ if and only if $\beta_i(e_i) = \beta_j(e_j)$, $\alpha_i({}^{N_i}\bullet e_i) = \alpha_j({}^{N_i}\bullet e_j)$, and $\alpha_i(e_i {}^{N_i}\bullet) = \alpha_j(e_j {}^{N_i}\bullet)$.
2. For each $e \in E^\star$, we say $\text{eqvl}(e) = \{e' \in E^\star \mid e \bowtie e'\}$.
3. Let ν' be the set of equivalence classes of $\bowtie$, i.e. $\nu' = \{\text{eqvl}(e) \mid e \in E^\star\}$.
4. For all causal nets (C_i, E_i, K_i, S_i) and labeling functions α_i, define a labeling function $\beta'_i : E_i \rightarrow \nu'$ such that for an event e_i, $\beta'_i(e_i) = \text{eqvl}(e_i)$, i.e. it returns the equivalence class of $\bowtie$ containing e_i.

After re-labeling the events, the part of the run shown in Figure 6(d) is relabeled to include the pre- and post-conditions. Figure 7 shows the fragment after relabeling. (We only show the relabeling with respect to the post-conditions.) Applying the ALK algorithm of Definition 4.5 to the relabeled runs yields the result as shown in Figure 8. Note that we do not show the ν' labels explicitly, i.e. B refers to the equivalence class of events labeled B.

What remains to be shown is that our algorithm does not only work for our small running example, but also in the general case. The only difference between the assumptions in Definition 4.5 and Definition 4.6 is the requirement with respect to transition equivalence. Therefore, if suffices to show that after applying the relabeling algorithm on a causal set, we can establish transition equivalence.

Property 4.7. (Transition equivalence holds after relabeling)

(a) (b) (c) (d)

Fig. 6. Four examples of parts of runs

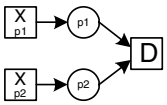


Fig. 7. The relabeled part of Figure 6(d)

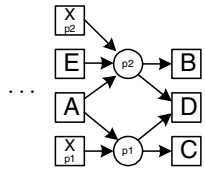


Fig. 8. Part of the aggregated net

Let μ, ν be two disjoint sets of labels. Let $\Phi = \{(C_i, E_i, K_i, S_i) \mid 0 \leq i < n\}$ be a causal set, and let $\Psi = \{(\alpha_i : C_i \to \mu, \beta_i : E_i \to \nu) \mid 0 \leq i < n\}$ be a set of labeling functions in (C_i, E_i, K_i, S_i), such that $\alpha_i(S_i)$ is the same for all causal nets. After applying the relabeling algorithm, the property of transition equivalence holds for (Φ, Ψ'), with $\Psi' = \{(\alpha_i : C_i \to \mu, \beta_i' : E_i \to \nu') \mid 0 \leq i < n\}$, and β_i' as defined in Definition 4.6.

Proof. We prove that Property 4.4 holds for (Φ, Ψ') after applying the relabeling function. Assume (C_i, E_i, K_i, S_i) and (C_j, E_j, K_j, S_j) are two causal nets from Φ. The new function β_i' is indeed a function, since for each event $e_i \in E_i$ there exists exactly one equivalence class containing e_i. Furthermore, let $e_i \in E_i$ and $e_j \in E_j$, such that $\beta_i'(e_i) = \beta_j'(e_j)$. We know that $e_i \bowtie e_j$ and from the definition of $\bowtie$, we know that $\alpha_i(\bullet e_i) = \alpha_j(\bullet e_j)$ and $\alpha_i(e_i \bullet) = \alpha_j(e_j \bullet)$, which directly implies transition equivalence. □

The algorithm presented above is capable of finding events that have the same label, but correspond to different transitions in the system net. When *no transition labels are known at all*, it can be applied to *find all transition labels*, by using an initial $\nu = \{\tau\}$ and initial mapping functions β_i, mapping everything onto τ. However, in that case, no distinction can be made between events that have the same pre- and post-set, but should have different labels. After applying this relabeling algorithm, the ALK algorithm of Section 4.1 can be used to find the system net belonging to the given causal nets.

4.3 Aggregation with Unknown Place Labels

In Section 4.2, we have shown a way to identify the transitions in a system net, based on the labels of events in causal nets. However, what if condition labels are not known? Notice that the difference to other approaches based on partial orders is that here we do know the conditions constituting the order between events but do *not* know which two conditions refer to a token in the same place of the P/T-net representing the process.

So, in this section, we take one step back. We assume all events to refer to the correct transition, as we did in Section 4.1 and we try to identify the labels of conditions. We introduce an algorithm to aggregate causal nets to a system net, such that the original causal nets are indeed runs of that system net.

In Figure 9, we again show our small example of the aggregation problem, only this time there are no labels for conditions $p1$ and $p2$, which we did have in Figures 4 and 6.

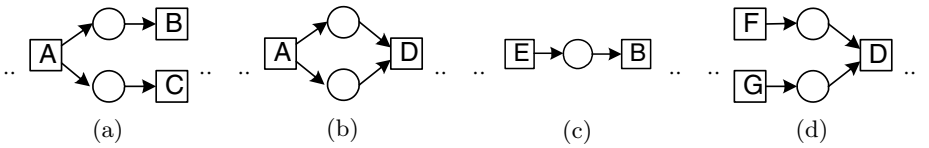


Fig. 9. Four examples of parts of runs

Consider the four runs of Figure 9. Remember that they are *parts* of causal nets, in such a way that the tasks A, B, C, D, E, F and G do not appear in any other way in another causal net. In contrast to the algorithms presented in previous sections, we cannot always derive a unique aggregated system net for causal nets if we do not have labels for the conditions. Instead, we define an *aggregation class*, describing a class of WF-nets that could have generated these causal nets. The following table shows some requirements all WF-nets in the aggregation class of our example should satisfy.

Table 1. Information derived from runs shown in Figure 9

Fragment	Conclusions
Fig. 9(a)	$A\bullet = \bullet B \uplus \bullet C$
Fig. 9(b)	$A\bullet = \bullet D$
Fig. 9(c)	$E\bullet = \bullet B$
Fig. 9(d)	$F \bullet \uplus G\bullet = \bullet D$

The information in Table 1 is derived from the runs of Figure 9 in the following way. Figure 9(a) shows that the transition A produces two tokens in two places and that transitions B and C consume these two tokens, while at the same time they do not need more input. Hence, we can conclude that in any aggregated net, the multiset of tokens produced by A should be equal to the multiset of tokens consumed by B and C together, which is stated in the first line of Table 1.

In the general case, this information can be derived using the concept of a segment, which can be considered to be the context of a condition in a causal net. A segment consists of two sets of events (an input set and an output set), such that the tokens produced by the transitions in the system net, corresponding to the events in the input set are exactly the tokens consumed by the transitions corresponding to the events in the output set, i.e. we formally capture the relations described in Table 1.

Definition 4.8. (Segment)
Let $N = ((C, E, K), S_0)$ be a causal net and let $N' = (C', E_{in}, E_{out})$ be such that $C' \subseteq C$, $E_{in} \cup E_{out} \subseteq E$, $E_{in} \neq \emptyset$ and $E_{out} \neq \emptyset$. We call N' a *segment* if:

- for all $c \in C'$ holds that $\bullet c \subseteq E_{in}$ and $c\bullet \subseteq E_{out}$, and
- for all $e \in E_{in}$ holds that $e\bullet \subseteq C'$, and

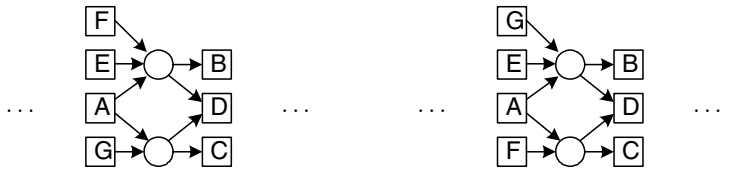


Fig. 10. Two aggregated nets

- for all $e \in E_{out}$ holds that $\bullet e \subseteq C'$, and
- the subgraph of N made up by $C' \cup E_{in} \cup E_{out}$ is connected.

We call the events in E_{in} the input events and the events in E_{out} the output events.

A segment is called *minimal* if C' is minimal, i.e. if there does not exist a segment $N'' = (C'', E''_{in}, E''_{out})$ with $C'' \subset C'$ and $C'' \neq \emptyset$.

For the fragments of Figure 9, it is easy to see that each of them contains only one minimal segment, where the input events are the events on the left hand side and the output events are the events on the right hand side.

The meaning of a segment is as follows. If we have a run and a segment in that run, then we know that after all events in the input set of the segment occurred, all the events in the output set occurred in the execution represented by this run. This translates directly to a marking in a system net, since the occurrence of a set of transitions would lead to some marking (i.e. a bag over places), which enables another set of transitions. Furthermore, each transition only produces one token in each output place. Combining this leads to the fact that for each minimal segment in a causal net the bag of places following the transitions corresponding to the input events of the segment should be the same as the bag of places preceding the transitions corresponding to the output set of events, as indicated in Table 1.

Clearly, when looking only at these fragments, what we are looking for are the places that should be put between tasks A, E, F and G on the one hand, and B, C and D on the other hand. Therefore, we only focus on this part of the causal nets. For this specific example, there are two possibilities, both of which are equally correct, namely the two WF-net fragments shown in Figure 10.

From the small example, we have seen that it is possible to take a set of causal nets without labels for any of the conditions (but with labels for all the events) and to define a class of potential system nets of the causal nets. In the remainder of this section, we show that this is indeed possible for all causal sets. For this, we first introduce the NCL algorithm.

4.4 NCL Algorithm

Before presenting the NCL algorithm (which stands for "No Condition Labels"), we first take a look at a more intuitive example. Consider Figure 11, where we

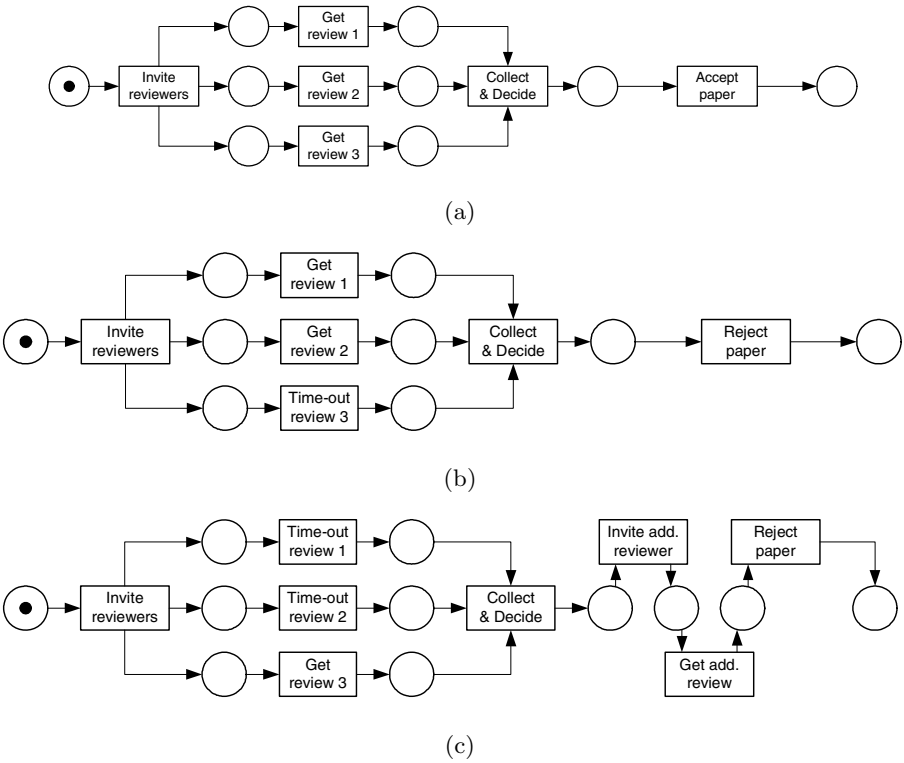


Fig. 11. Three causal nets of a review process of a paper

present three causal nets, each of which corresponds to a paper review process. In the first causal net, three reviewers are invited to review the paper and after the three reviews are received, the paper is accepted. In the second causal net, only two reviews are received (the third one is not received on time), but the paper is rejected nonetheless (apparently the two reviewers that replied rejected the paper). In the third example only one review is received in time, and therefore an additional reviewer is invited, which hands in his review in time, but does not accept the paper.

As we stated before, we define an aggregation class of a causal set that contains all WF-nets that are capable of generating the causal nets in the causal set. The information needed for this aggregation class comes directly from the causal nets, using minimal segments. In Table 2, we present the conclusions we can draw based on the three causal nets of Figure 11. In this table we consider *bags* of pre- and post-sets of transitions in the aggregation class. The information in this table is obtained from the causal nets in the following way. Consider for example Figure 11(a), where *Invite reviewers* is followed by *Get review 1*,*Get review 2* and *Get review 3*. This implies that the bag of output places of *invite reviewers* should be the same as the sum over the bags of the input places of *Get review 1*, *Get review 2* and *Get review 3*.

Table 2. Information derived from review example

Causal net	Conclusions on transitions in the aggregation class
Fig. 11(a)	•"Invite reviewers" $= [p_{ini}]$
	"Invite reviewers"• $= \bullet$"Get review 1" $\uplus$
	$\bullet$"Get review 2" $\uplus$
	$\bullet$"Get review 3"
	"Get review 1" • $\uplus$ $= \bullet$"Collect & Decide"
	"Get review 2" • $\uplus$
	"Get review 3"•
	"Collect & Decide"• $= \bullet$"Accept paper"
	|"Accept paper" • | $= 1$
Fig. 11(b)	•"Invite reviewers" $= [p_{ini}]$
	"Invite reviewers"• $= \bullet$"Get review 1" $\uplus$
	$\bullet$"Get review 2" $\uplus$
	$\bullet$"Time-out review 3"
	"Get review 1" • $\uplus$ $= \bullet$"Collect & Decide"
	"Get review 2" • $\uplus$
	"Time-out review 3"•
	"Collect & Decide"• $= \bullet$"Reject paper"
	|"Reject paper" • | $= 1$
Fig. 11(c)	•"Invite reviewers" $= [p_{ini}]$
	"Invite reviewers"• $= \bullet$"Time-out review 1" $\uplus$
	$\bullet$"Time-out review 2" $\uplus$
	$\bullet$"Get review 3"
	"Time-out review 1" • $\uplus$ $= \bullet$"Collect & Decide"
	"Time-out review 2" • $\uplus$
	"Get review 3"•
	"Collect & Decide"• $= \bullet$"Invite add. reviewer"
	"Invite add. reviewer"• $= \bullet$"Get add. review"
	"Get add. review"• $= \bullet$"Reject paper"
	|"Reject paper" • | $= 1$

Definition 4.9. (NCL algorithm: Aggregation Class)
Let $\Phi = \{(C_i, E_i, K_i, S_i) \mid 0 \leq i < n\}$ be a causal set, and let $N = (P, T, F, M_0)$ be a marked WF-net. For each causal net $N_i \in \Phi_i$, let $\beta_i : E_i \to T$ be a mapping from the events of that causal net to T, such that β_i is a labeling function for E_i. We define $\mathcal{A}_\Phi$, the *aggregation class* of Φ, as the set of all pairs $(N, \mathcal{B})$ such that the following conditions are satisfied:

1. $T = \bigcup_{0 \leq i < n} \mathrm{rng}(\beta_i)$ is the set of transitions, i.e. each transition appears as an event at least once in some causal net,

2. For all $p \in P$ holds that ${}^{N}\bullet p \cup p^{N}\bullet \neq \emptyset$,
3. $M_0 = [p_{ini}]$ and ${}^{N}\bullet p_{ini} = \emptyset$,
4. $\mathcal{B}$ is the set of all labeling functions, i.e. $\mathcal{B} = \{\beta_i \mid 0 \leq i < n\}$. We use $\beta_i \in \mathcal{B}$ to denote the labeling function for events belonging to $N_i \in \Phi$,
5. For each causal net $N_i = (C_i, E_i, K_i, S_i)$, with $e \in E_i$ and $\beta_i(e) = t$ holds that if $S_i({}^{N_i}\bullet e) = 1$ then $p_{ini} \in {}^{N}\bullet t$,
6. For each causal net $N_i = (C_i, E_i, K_i, S_i)$, with $e \in E_i$ and $\beta_i(e) = t$ holds that $|t^{N}\bullet| = |e^{N_i}\bullet|$ and $|{}^{N}\bullet t| = |{}^{N_i}\bullet e|$,
7. For each causal net $N_i = (C_i, E_i, K_i, S_i)$, with $e \in E_i$, $\beta_i(e) = t$ and $T' \subseteq T$ holds that $|t^{N}\bullet \cap \bigcup_{t' \in T'}({}^{N}\bullet t')| \geq \sum_{e' \in E_i, \beta(e') \in T'} |e^{N_i}\bullet \cap {}^{N_i}\bullet e'|$,
8. For each causal net $N_i = (C_i, E_i, K_i, S_i)$, with $e \in E_i$, $\beta_i(e) = t$ and $T' \subseteq T$ holds that $|\bigcup_{t' \in T'}(t'^{N}\bullet) \cap {}^{N}\bullet t| \geq \sum_{e' \in E_i, \beta(e') \in T'} |e'^{N_i}\bullet \cap {}^{N_i}\bullet e|$,
9. For each causal net $N_i = (C_i, E_i, K_i, S_i)$ and any minimal segment (C_i', E_{in}, E_{out}) of N_i, holds that $\biguplus_{e \in E_{in}} (\beta_i(e)^{N}\bullet) = \biguplus_{e \in E_{out}} ({}^{N}\bullet \beta_i(e))$.

Definition 4.9 defines an aggregation class of models in the following way:

- For each workflow net in the class, Items 1 to 4 define the transitions, places, initial marking and the labeling functions, labeling all events and conditions of each causal net with the transitions and places of that workflow net.
- Item 5 guarantees that all events in causal sets consuming the initial tokens are labeled with output transitions of the initially marked place in the workflow net.
- Item 6 guarantees that, for all events, the numbers of input and output conditions correspond to the numbers of input and output places of the corresponding transition.
- Items 7 and 8 refer to the token flow in the model, in relation to the causal nets, i.e. when considering the flow between a set of transitions and one other transitions (in any direction), the number of tokens ever observed in any causal set cannot be larger than the number of tokens allowed according to the model. Hence, choices in the model do not correspond to parallel behavior in any causal net.
- Figure 12 is used to gain more insight into Item 9 of Definition 4.9. In the lower causal net of that figure, there is a token traveling from A to D and another one from B to C. The upper causal net only connects A and C. Assuming that these are the only causal nets in which these transitions appear, we know that the conditions between A and D and between B and C should represent a token in the same place, since there is a minimal segment $(\{c_4, c_5, c_6\}, \{A, B\}, \{C, D\})$ in the lower causal net and therefore, $A\bullet \uplus B\bullet = \bullet C \uplus \bullet D = [p_1, 2p_2]$.

Consider the information presented in Table 2 and the two Petri nets in Figure 13. Both nets in Figure 13 adhere to all constraints of Table 2. As this

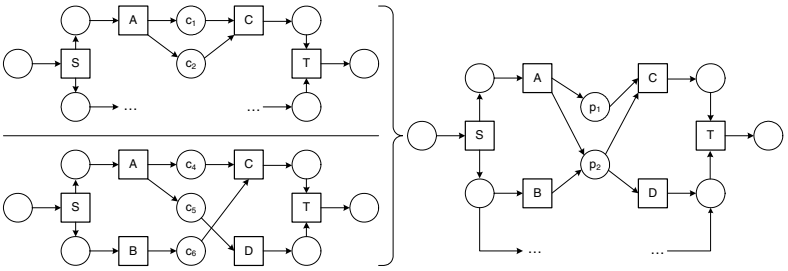


Fig. 12. Example explaining the use of bags

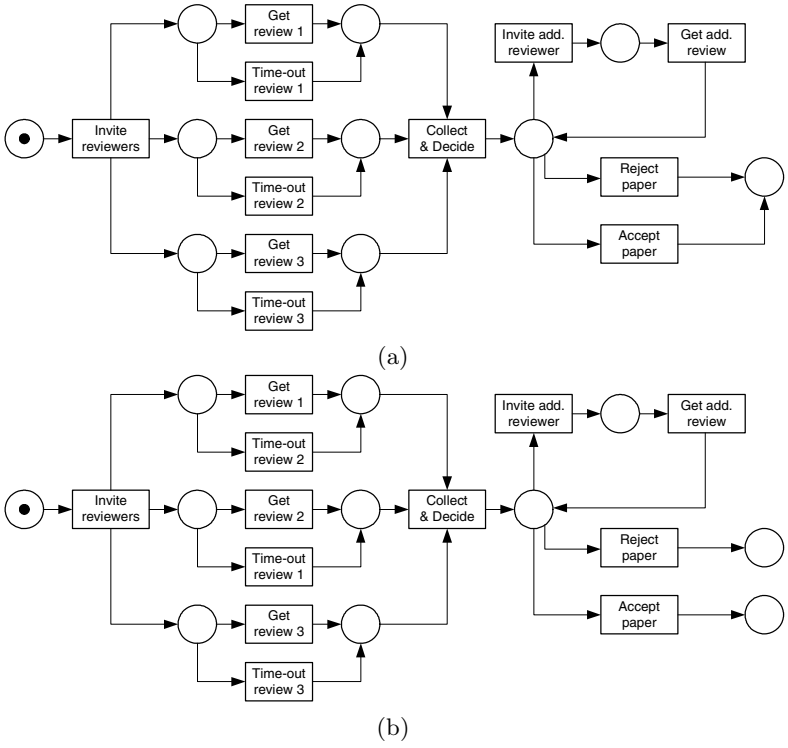


Fig. 13. Two possible aggregated nets, both obeying the constraints of Table 2

example shows, there is no unique Petri net satisfying all constraints. Instead, there is a class of nets satisfying all constraints.

The condition provided in Item 9 of Definition 4.9 provides the key to constructing the actual elements of the aggregation class. By considering all minimal segments in the provided runs that refer to the same transitions, possible sets of places can be identified that satify this condition. However, in this paper, we do not provide construction steps for constructing the aggregation class. Instead,

in the next section, we show that if a set of runs is generated by a system net, then that system net is a member of the aggregation class.

5 Correctness of the Aggregation Algorithms

In Section 4, we described three scenarios for which we can construct an aggregated net from a set of runs. In Section 4.1, we showed the ALK algorithm, which assumes that in the runs, all conditions and events are labeled with the corresponding places and transitions of the aggregated net. In Section 4.2, we showed that in case some transition labels are duplicated or missing, we can still use the ALK algorithm after relabeling the transitions using the surrounding places. Finally, in Section 4.3, we presented the NCL algorithm that provides an aggregation class of nets that are all capable of reproducing the given set of runs in which none of the conditions is labeled.

In this section, we formally prove correctness of the ALK and the NCL algorithms.

5.1 Correctness of the ALK Algorithm

The ALK algorithm defines a single aggregated net for a given set of runs. In order to prove its correctness, we show that the runs used as input can be generated by the resulting aggregated net.

Property 5.1. (The ALK algorithm is correct)
For all $0 \leq i < n$ and $N_i = (C_i, E_i, K_i, S_i)$, the tuple (N_i, α_i, β_i) is indeed a run of $N = (P, T, F, M_0)$ (i.e., the requirements stated in Definition 3.11 are fulfilled).

Proof. Since we assumed that all causal nets $N_i = (C_i, E_i, K_i, S_i)$ are elements of the causal set Φ, we need to prove the following for each α_i and β_i.

1. α_i is a function from C_i onto P. This trivially follows from Definition 4.5.
2. β_i is a function from E_i onto T. This trivially follows from Definition 4.5.
3. $\alpha_i(S_i) = M_0$ holds by definition, since it holds for S_0 and for all causal nets, $\alpha_i(S_i)$ is the same.
4. For each event $e \in E_i$, the mapping α_i induces a bijection from $\bullet e$ to $\bullet\beta_i(e)$ and a bijection from $e\bullet$ to $\beta_i(e)\bullet$.

   Let $e \in E_i$. We start by showing that $\alpha_i({}^{N_i}\bullet e) = {}^{N}\bullet\beta_i(e)$ and $\alpha_i(e^{N_i}\bullet) = \beta_i(e)^{N}\bullet$. Assume $p \in \alpha_i({}^{N_i}\bullet e) \setminus {}^{N}\bullet\beta_i(e)$, i.e. there exists a $c \in C_i$ with $(c, e) \in K_i$, such that $p = \alpha_i(c)$, $\beta_i(e) = t$ and $(p, t) \notin F$. Clearly this contradicts with the definition of F in Definition 4.5. Now assume $p \in {}^{N}\bullet\beta_i(e) \setminus \alpha_i({}^{N_i}\bullet e)$, i.e. there is a $(p, t) \in F$ such that $\beta_i(e) = t$ and there is no $c \in C_i$ with $\alpha_i(c) = p$, such that $(c, e) \in K_i$. If this is the case in all causal nets for $0 \leq i < n$, then this leads to a contradiction since this would imply $(p, t) \notin F$ (cf. Definition of F in Definition 4.5). If there is a $0 \leq j < n$, such that $(c', e') \in K_j$ with

$\beta_j(e') = t$ and $\alpha_j(c') = p$, then there has to be a $c \in C_i$ such that $(c, e) \in K_i$, since $\alpha_i({}^{N_i}\bullet e) = \alpha_j({}^{N_i}\bullet e')$ (cf. Definition 4.4). Combined with the fact that α_i and β_i are labeling functions, $\alpha_i({}^{N_i}\bullet e) = {}^{N}\bullet\beta_i(e)$ and $\alpha_i(e^{N_i}\bullet) = \beta_i(e)^{N}\bullet$ yields the bijection. Similar arguments apply for the post-set.

□

Property 5.1 shows that the ALK algorithm indeed results in a system net of which the causal nets used as input are runs.

5.2 Correctness of the NCL Algorithm

In case that no condition labels are present, the NCL algorithm defines an equivalence class of aggregated nets. In this section, we show that for each net in this aggregation class, the causal nets used as inputs can be considered runs. Furthermore, we show that if we take the runs of a sound workflow model as input, then that model is part of the aggregation class.

Definition 4.9 defines a *finite* class of WF-nets for a causal set. What remains to be given are the conditions under which it is a finite *non-empty* class of Petri nets and the proof that each Petri net with its mappings is indeed a system net for the causal set. To prove this, we first introduce the concept of a *condition graph*.

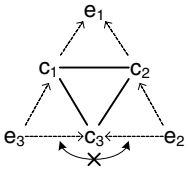
Definition 5.2. (Condition graph)
Let $N_i = (C_i, E_i, K_i, S_i)$ be a causal net. The undirected graph $\Delta_{N_i} = (C_i, A)$, with $A = \{(c_1, c_2) \in C_i \times C_i \mid \exists_{e \in E_i} \{c_1, c_2\} \subseteq {}^{N_i}\bullet e \ \vee \ \{c_1, c_2\} \subseteq e^{N_i}\bullet\}$ is called a *condition graph*. Note that $(c_1, c_2) \in A$ implies that $(c_2, c_1) \in A$.

We use condition graphs to prove that each Petri net with its mappings in the aggregation class of a causal set is indeed a system net for that causal set. For this, we first introduce some lemmas on these condition graphs that show the relation between condition graphs and causal nets. We start by showing that pre- and post-sets of events correspond to complete subgraphs in the condition graph, i.e. subgraphs where each pair of nodes is connected by an edge.

Lemma 5.3. (Pre- and post sets relate to complete subgraphs in condition graphs)
Let $N_i = (C_i, E_i, K_i, S_i)$ be a causal net and $\Delta_{N_i} = (C_i, A)$ its condition graph. We show that, for each $e \in E_i$, holds that Δ_{N_i} restricted to ${}^{N_i}\bullet e$ is a complete subgraph and Δ_{N_i} restricted to $e^{N_i}\bullet$ is a complete subgraph. Furthermore, for each complete subgraph (C', A'), there exists an $e \in E_i$ such that $C' \subseteq {}^{N_i}\bullet e$ or $C' \subseteq e^{N_i}\bullet$.

Proof. Since for all $\{c_1, c_2\} \subseteq {}^{N_i}\bullet e$, holds that $(c_1, c_2) \in A$ by definition, the first part is correct. The same applies to $e^{N_i}\bullet$. Now assume (C', A') is a complete subgraph. Assume $\{c_1, c_2\} \subseteq C'$. and $c_1 \neq c_2$. Since we are looking at a complete subgraph, we know $(c_1, c_2) \in A'$, therefore there exists an $e_1 \in E_i$, such that $\{c_1, c_2\} \subseteq {}^{N_i}\bullet e_1$ or $\{c_1, c_2\} \subseteq e_1{}^{N_i}\bullet$.

Assume $\{c_1, c_2\} \subseteq {}^{N_i}\bullet e_1$ (The proof is symmetrical for $e_1{}^{N_i}\bullet$). Now assume $c_3 \in C'$ such that $c_1 \neq c_3$ and $c_2 \neq c_3$. Let $c_3 \notin {}^{N_i}\bullet e_1$. We show that this leads to a contradiction. Since for all $c \in C$ holds that $|c{}^{N_i}\bullet| \leq 1$ and $\{c1, c2\} \subseteq {}^{N_i}\bullet e_1$, we know that there must be an $e_2 \in E_i$, such that $\{c_2, c_3\} \subseteq e_2{}^{N_i}\bullet$.



Similarly, we know that there is an $e_3 \in E_i$, such that $\{c_1, c_3\} \subseteq e_3{}^{N_i}\bullet$. However, since $|{}^{N_i}\bullet c_3| \leq 1$, this implies that $e_2 = e_3$ and thus $\{c_1, c_2, c_3\} \subseteq e_2{}^{N_i}\bullet$. □

Using the fact that each pre- and post-set correspond to a complete subgraph, we can infer that each minimal segment in a causal net corresponds to a connected subgraph in the condition graph, i.e. a subgraph such that there is a path between each two nodes. Furthermore, we show that these connected subgraphs are maximal, i.e. all nodes in the subgraph are only connected to nodes inside the subgraph.

Lemma 5.4. (Minimal segments correspond to maximal connected subgraphs in condition graphs)
Let $N_i = (C_i, E_i, K_i, S_i)$ be a causal net and $\Delta_{N_i} = (C_i, A)$ its condition graph. Let (C', E_{in}, E_{out}) be a minimal segment in N_i. We show that $(C', A \cap (C' \times C'))$ is a maximal connected subgraph of Δ_{N_i}.

Proof. From Definition 4.8 we know that the graph $(C' \cup E_{in} \cup E_{out}, K_i \cap ((C' \cup E_{in} \cup E_{out}) \times (C' \cup E_{in} \cup E_{out})))$ is a connected graph. Now, let $c \in C'$ be a condition in the minimal segment and assume that $\{e_{in}\} = {}^{N_i}\bullet c$ and $\{e_{out}\} = c{}^{N_i}\bullet$. From Lemma 5.3, we know that $e_{in}{}^{N_i}\bullet$ and ${}^{N_i}\bullet e_{out}$ make up a complete subgraph in Δ_{N_i} and since $c \in {}^{N_i}\bullet e_{out} \cap e_{in}{}^{N_i}\bullet$ that these two complete subgraphs make up a connected subgraph. By induction over the elements of C', it is easy to show that C' makes up a connected subgraph in Δ_{N_i}. Therefore, each minimal segment defines a complete subgraph G' in Δ_{N_i}. Furthermore, let $G' = (C', A')$ be the connected subgraph of ΔN_i corresponding to the segment. Let $c \in C_i \setminus C'$ and assume there exists a $c' \in C'$, such that $(c, c') \in A$. This implies that there is an $e \in E_i$, such that $\{c, c'\} \subseteq {}^{N_i}\bullet e$ or $\{c, c'\} \subseteq e{}^{N_i}\bullet$. However, this implies that $e \in E_{in}$ or $e \in E_{out}$, either of which imply that $c \in C'$. Therefore, such a c does not exist and G' is maximal. □

At this point, we look at the definitions of Section 3 again. If we assume that we have a system net and the causal behavior of this system net, we can derive the next lemma using Definition 3.4.

Lemma 5.5. (System nets color condition graphs)
Let $N = (P, T, F, M_0)$ be a system net and (N_i, α_i, β_i) be a run of that system net, with $N_i = (C_i, E_i, K_i, S_i)$. Furthermore, let $\Delta_{N_i} = (C_i, A)$ be the condition graph of N_i. The mapping $\alpha_i : C_i \to P$ is a coloring function of Δ_{N_i}, with the set of colors being P.

Proof. Let $n_1, n_2 \in C_i$ be two nodes in Δ_{N_i} with $n_1 \neq n_2$. For α_i to be a coloring, $\alpha_i(n_1) \neq \alpha_i(n_2)$ should hold if $(n_1, n_2) \in A$. Assume $(n_1, n_2) \in A$. This

means that there is an $e \in E_i$ such that $\{n_1, n_2\} \subseteq {}^{N_i}\bullet e$ or $\{n_1, n_2\} \subseteq e^{N_i}\bullet$. From Definition 3.11, we know that α_i induces a bijection from ${}^{N_i}\bullet e$ to ${}^{N}\bullet\beta_i(e)$ and from $e^{N_i}\bullet$ to $\beta_i(e)^{N}\bullet$. Therefore, $\alpha_i(n_1) \neq \alpha_i(n_2)$. □

We have shown that system nets color condition graphs. However, we can go one step further and introduce the concept of a condition coloring, which is a coloring on the condition graph, such that the coloring function, when applied to the conditions in a causal net, induces local bijections for the input and output sets of events.

Definition 5.6. (Condition coloring)
Let Φ be a causal set and let $\mathcal{A}_\Phi$ be the aggregation class of Φ. Moreover, let $(N, \mathcal{B}) \in \mathcal{A}_\Phi$, with $N = (P, T, F, M_0)$ and let $N_i = (C_i, E_i, K_i, S_i) \in \Phi$ be a causal net and $\Delta_{N_i} = (C_i, A)$ be the condition graph of N_i. Assume $\alpha_i : C_i \to P$ is a function, such that α_i is a coloring on Δ_{N_i} and for all $c \in C_i$ holds that $\alpha_i(c) \in \{p \in P \mid \beta({}^{N_i}\bullet c) \subseteq {}^{N}\bullet p \ \wedge \ \beta(c^{N_i}\bullet) \subseteq p^{N}\bullet\}$. [2] We then call α_i a condition coloring of Δ_{N_i}.

The concept of a condition coloring we introduced here is often referred to in mathematics as a list coloring.

Lemma 5.7. (Condition coloring induces bijections)
Let Φ be a causal set and let $\mathcal{A}_\Phi$ be the aggregation class of Φ, and let $(N, \mathcal{B}) \in \mathcal{A}_\Phi$, with $N = (P, T, F, M_0)$. Let $N_i = (C_i, E_i, K_i, S_i) \in \Phi$ be a causal net and $\Delta_{N_i} = (C_i, A)$ be the condition graph of N_i. Let $\alpha_i : C_i \to P$ be a condition coloring of Δ_{N_i}. We show that for all $e \in E_i$, α_i induces a bijection from ${}^{N_i}\bullet e$ to ${}^{N}\bullet\beta_i(e)$ and from $e^{N_i}\bullet$ to $\beta(e_i)^{N}\bullet$.

Proof. The requirements stated in Definition 4.9, imply that $|\,{}^{N_i}\bullet e| = |\,{}^{N}\bullet\beta(e)|$. Furthermore, since Δ_{N_i} restricted to ${}^{N_i}\bullet e$ is a complete graph (Lemma 5.3), and α_i is a coloring function (Lemma 5.5), we know that $|\alpha_i({}^{N_i}\bullet e)| = |\,{}^{N_i}\bullet e|$ Since all elements in ${}^{N_i}\bullet e$ are mapped to different colors. Combining both implies that $|\alpha_i({}^{N_i}\bullet e)| = |\,{}^{N}\bullet\beta_i(e)|$.

For all $c \in {}^{N_i}\bullet e$ holds that $\alpha_i(c) \in \{p \in P \mid \beta_i({}^{N_i}\bullet c) \subseteq {}^{N}\bullet p \ \wedge \ \beta_i(c^{N_i}\bullet) \subseteq p^{N}\bullet\}$ (Definition 5.6) and $c^{N_i}\bullet = \{e\}$, because N_i is a causal net we know that $\alpha_i(c) \in \{p \in P \mid \beta_i(e) \in p^{N}\bullet\}$ and thus $\alpha_i(c) \in {}^{N}\bullet\beta_i(e)$. Since this holds for all $c \in {}^{N_i}\bullet e$, we can conclude that $\alpha_i({}^{N_i}\bullet e) \subseteq {}^{N}\bullet\beta_i(e)$. By combining the above, we can conclude that $\alpha_i({}^{N_i}\bullet e) = {}^{N}\bullet\beta(e)$, and thus that α_i induces a bijection from ${}^{N_i}\bullet e$ to ${}^{N}\bullet\beta_i(e)$. A similar proof holds for the mapping from $e^{N_i}\bullet$ to $\beta_i(e)^{N}\bullet$. □

At this point we still need to prove the following for an arbitrary WF-net in the aggregation class. For each causal net in a causal set, we should be able to color its condition graph using a condition coloring. If we are able to construct such a coloring, we have satisfied the first requirement stated in Definition 3.11.

[2] Note that β is generalized, i.e. for a set E holds that $\beta(E) = \{\beta(e) \mid e \in E\}$.

Lemma 5.8. (Condition coloring exists)
Let Φ be a causal set, let $\mathcal{A}_\Phi$ be the aggregation class of Φ, and let $(N, \mathcal{B}) \in \mathcal{A}_\Phi$, with $N = (P, T, F, M_0)$. Let $N_i = (C_i, E_i, K_i, S_i) \in \Phi$ be a causal net and $\Delta_{N_i} = (C_i, A)$ be the condition graph of N_i. Let $\beta_i \in \mathcal{B}$ be the labeling function belonging to N_i. We show that we can construct a mapping $\alpha_i : C_i \to P$, such that α_i is a condition coloring of Δ_{N_i}.

Proof. First, we look at the initial condition, i.e. the initially marked source condition. Assume $c \in C_i$ such that ${}^{N_i}\bullet c = \emptyset$. We call $c {}^{N_i}\bullet = \{e\}$. From the definition of a causal set (Def. 4.3), we know that $\{c\} = {}^{N_i}\bullet e$ and thus that there is no $c' \in C_i$ with $c \neq c'$ and $(c, c') \in A$. We know that ${}^{N}\bullet\beta_i(e) = \{p_{ini}\}$ (Def. 4.9). By setting $\alpha_i(c) = p_{ini}$, we have a correct coloring for the initial condition c in N.

Second, we look at the final conditions, i.e. the sink conditions. Assume $c \in C_i$ such that $c {}^{N_i}\bullet = \emptyset$. We call ${}^{N_i}\bullet c = \{e\}$. From the definition of a causal set (Def. 4.3), we know that $\{c\} = e {}^{N_i}\bullet$ and thus that there is no $c' \in C_i$ with $c \neq c'$ and $(c, c') \in A$. We know that $|\beta_i(e){}^{N}\bullet| = 1$ (Def. 4.9). We say that $\beta_i(e){}^{N}\bullet = \{p\}$. By setting $\alpha_i(c) = p$, we have a correct coloring for any final condition c in N_i.

Finally, we split the graph up into two subgraphs. Let $A_{in} = \{(c_1, c_2) \in A \mid {}^{N_i}\bullet c_1 = {}^{N_i}\bullet c_2\}$ and let $A_{out} = \{(c_1, c_2) \in A \mid c_1 {}^{N_i}\bullet = c_2 {}^{N_i}\bullet\}$. Using the definition of a condition graph it is easy to see that $A_{in} \cup A_{out} = A$. We now show that for each subgraph $\delta_{in}(N_i) = (C, A_{in})$ and $\delta_{out}(N_i) = (C, A_{out})$ we can construct at least one condition coloring. Then, we show that there is at least one condition coloring that is the same for both subgraphs after which we can use Lemma 3.5 to show that this is a condition coloring on the complete graph.

Consider the subgraph $\delta_{in}(N_i) = (C, A_{in})$. Using Lemma 5.3, it is easy to see that this graph consists of several complete components and that each component is a complete graph. Let $e \in E_i$. We know that $e {}^{N_i}\bullet \subseteq C$ and that $e {}^{N_i}\bullet$ defines a complete component in $\delta_{in}(N_i)$. Now, let $V_1, \ldots, V_n$ be maximal sets, such that for each $0 < i \leq n$ holds that $V_i \subseteq E {}^{N_i}\bullet$ and for all $c_1, c_2 \in V_i$ holds that $c_1 {}^{N_i}\bullet = c_2 {}^{N_i}\bullet$. For each V_i and $c \in V_i$, we say that $V_{i,in} = \{e\}$ and $V_{i,out} = c {}^{N_i}\bullet$.

From Definition 5.6, we know that for each $c \in V_i$ must hold that $\alpha_i(c) \in \{p \in P \mid \beta_i(\{e\}) \subseteq {}^{N}\bullet p \wedge \beta(V_{i,out}) \subseteq p {}^{N}\bullet\}$. Using Item 7 of Definition 4.9, we first prove a necessary condition for this. Assume $\beta_i(\{e\}) = \{t\}$, and $\beta_i(V_{i,out}) = T' = \{t'\}$. Item 7 shows us that $|t{}^{N}\bullet \cap {}^{N}\bullet t'| \geq \sum_{e' \in V_{i,out}} |e {}^{N_i}\bullet \cap {}^{N_i}\bullet e'|$. From the definition of partition V, we know that $\sum_{e' \in V_{i,out}} |e {}^{N_i}\bullet \cap {}^{N_i}\bullet e'| = |V_i|$. Furthermore, $t{}^{N}\bullet \cap {}^{N}\bullet t' = \{p \in P \mid \beta_i(V_{i,in}) \subseteq {}^{N}\bullet p \ \wedge \ \beta_i(V_{i,out}) \subseteq p {}^{N}\bullet\}$. Therefore we know that there are at least enough colors available for each partition V_i. The same way of reasoning can be used to show that there are at least enough colors available for each set of partitions $\upsilon \subseteq \{V_1, \ldots, V_n\}$. (The latter requires the use of Item 8 instead of 7 of Definition 4.9). Therefore, there exists at least one condition coloring for the entire subgraph $\delta_{in}(N_i)$. Similarly, this can be shown for $\delta_{out}(N_i)$.

At this point, we have shown that we can construct condition colorings for two subgraphs of $\delta(N_i)$, namely $\delta_{in}(N_i)$ and $\delta_{out}(N_i)$. The final part of the

proof use Item 9 of Definition 4.9, since we now have to show that *the same* condition coloring can be constructed for both subgraphs. For this purpose, we consider a segment (C', E_{in}, E_{out}) in N_i. Since segments correspond to connected components of $\delta(N_i)$, it is sufficient to show that the same condition coloring can be constructed for $\delta_{in}(N_i)$ and $\delta_{out}(N_i)$, restricted to C', which we call $\delta'_{in}(N_i)$ and $\delta'_{out}(N_i)$. From the definition of a segment, it is clear that this restriction does not disturb the structure of $\delta_{in}(N_i)$ and $\delta_{out}(N_i)$, i.e. in both graphs, each connected component is still a complete subgraph. Now consider a possible condition coloring on $\delta'_{in}(N_i)$. Each color given to a condition in that graph refers to a place in the causal net. However, multiple conditions can be mapped onto each place, namely one condition for each token that was produced in that place by an element of E_{in}. The same holds for $\delta'_{out}(N_i)$, i.e. multiple condition can be mapped onto each place, namely one condition for each token that was consumed by a succeeding element of E_{out}. Since Item 9 of Definition 4.9 states that the tokens produced by E_{in} are the tokens consumed by E_{out}, it must be possible to construct the same condition coloring α_i for both $\delta'_{in}(N_i)$ and $\delta'_{out}(N_i)$. Using Lemma 3.5, we then know that this coloring α_i is a condition coloring on $\delta(N_i)'$, i.e. the restriction of $\delta(N_i)$ to C'.

Since we can now provide a condition coloring on each connected component of $\delta(N_i)$, we have shown that we can construct a condition coloring on the entire graph $\delta(N_i)$. □

To clarify the rather complex proof of Lemma 5.8 we use an example. Consider a causal net containing the fragment of a WF-net presented in Figure 14. We numbered the conditions 1 through 8 to be able to distinguish them. Now, assume that the two Petri nets presented in Figure 15 are parts of two alternative system nets appearing in the aggregation class of that causal net.

The proof of Lemma 5.8 depends on the condition graph of a run. Therefore, in Figure 16 we present the condition graph of the run presented in Figure 14. Note that we labeled the edges to show from which event the edge was derived.

In Lemma 5.8, the condition graph of Figure 16 (i.e. $\delta(N_i)$ in the lemma) is split up into two subgraphs, namely one for the input side of events (i.e. $\delta_{in}(N_i)$, see Figure 17) and one for the output sides of events (i.e. $\delta_{out}(N_i)$, see Figure 18).

Then the proof continues, by showing that for each of these two subgraphs it is possible to provide a condition coloring. Figure 19 shows the possible labels for each subgraph and both Petri nets from Figure 15. It is easy to see that this indeed leads to several possible colorings in each graph.

At this point it is proven that it is always possible to construct two coloring functions on the input and output subgraph that give the same label to each condition in both graphs. If we look at Figure 19 and we take the input subgraph shown in Figure 15(a) (i.e. the left-top figure) then it is easy to see that it is possible to label c_4 with p_2 and c_5 with p_1. This however is not possible in the output subgraph, since neighbor c_6 has to be mapped onto p_1. Instead, there is only one mapping that is the same for both subgraphs. The last part of the proof uses the fact that for each segment the input enables the output. This implies that the token that is placed in p_1 has to be consumed from there again.

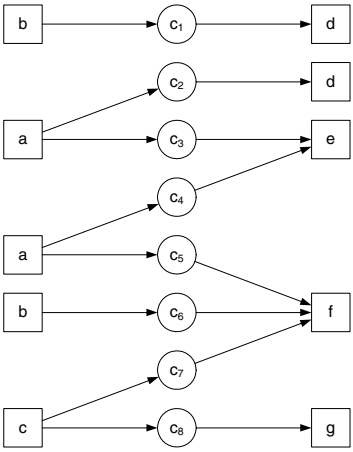


Fig. 14. A part of a run containing two segments

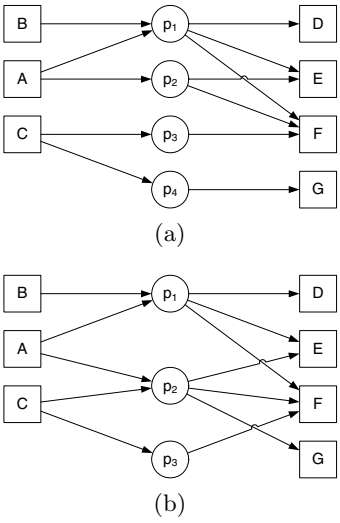


Fig. 15. Two parts of system nets in the aggregation class of Figure 14

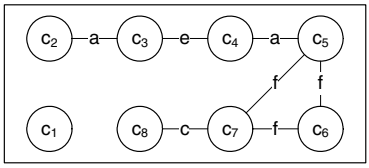


Fig. 16. Part of the condition graph of the run of Figure 14

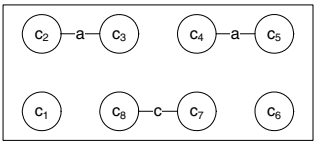


Fig. 17. Input subgraph of Figure 16

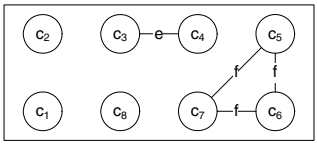


Fig. 18. Output subgraph of Figure 16

Therefore, if we would label c_5 with p_1 then this would be the same as saying that transition A produces a token in p_1 which is consumed by transition F again. However, transition F also consumes another token from p_1, namely the one corresponding to c_6, i.e. coming from transition B. This violates the fact that only one edge can exist between a place and a transition.

Figure 20 shows the only possible condition coloring of the condition graph of Figure 16, using the labels provided by the system net of Figure 15(a) and Figure 21 shows the only possible condition coloring of the condition graph of Figure 16, using the labels provided by the system net 15(b). Note that in general additional condition colorings may be possible.

From Figures 20 and 21, we can conclude that both system nets depicted in Figure 15 are indeed capable of producing the causal net of Figure 14, since we can construct a condition coloring on the condition graphs.

What remains to be shown is that the condition coloring also fulfills the last part of the definition of a run, namely the demand with respect to the initial marking. Furthermore, we conclude that at least three places are needed in the system net and that, for example, the place between C and G could also be p_1.

Lemma 5.9. (Initial marking can be mapped)
Let Φ be a causal set, let $\mathcal{A}_\Phi$ be the aggregation class of Φ and let $(N, \mathcal{B}) \in \mathcal{A}_\Phi$ with $N = (P, T, F, M_0)$. Let $N_i = (C_i, E_i, K_i, S_i) \in \Phi$ be a causal net and $\Delta_{N_i} = (C_i, A)$ be the condition graph of N_i. Let $\alpha_i : C_i \to P$, such that α_i is a condition coloring of Δ_{N_i}. We show that $\alpha_i(S_i) = M_0$.

Proof. From Definition 4.9, we know that $M_0 = [p_{ini}]$. Furthermore, from Definition 4.3, we know that there is exactly one $c \in C_i$ with $S(c) = 1$. Moreover, using Lemma 5.8, we conclude that $\alpha_i(c) = p_{ini}$ and thus $\alpha_i(S_i) = [p_{ini}] = M_0$. □

Finally, we can combine everything and state that each WF-net in an aggregation class is indeed a system net of a causal set.

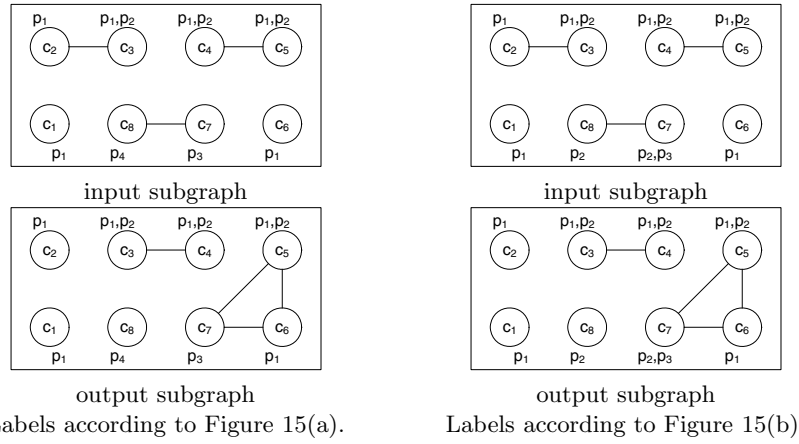


Fig. 19. Possible condition colorings for the subgraphs of figures 17 and 18

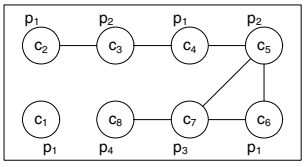


Fig. 20. The condition coloring of Figure 16 according to Figure 15(a)

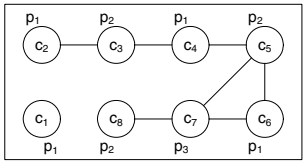


Fig. 21. The condition coloring of Figure 16 according to Figure 15(b)

Property 5.10. (Aggregation class only contains system nets) Let Φ be a causal set, let $\mathcal{A}_\Phi$ be the aggregation class of Φ and let $(N, \mathcal{B}) \in \mathcal{A}_\Phi$ with $N = (P, T, F, M_0)$. Let $N_i = (C_i, E_i, K_i, S_i) \in \Phi$ be a causal net with event labeling function $\beta_i \in \mathcal{B}$, condition graph $\Delta_{N_i} = (C_i, A)$ and $\alpha_i : C_i \to P$ a condition coloring of Δ_{N_i}. Then (N_i, α_i, β_i) is a run of N.

Proof. This result combines Lemma 5.7, which shows that for all $e \in E_i$, α_i induces a bijection from ${}^{N_i}\bullet e$ to ${}^{N}\bullet\beta_i(e)$ and from $e{}^{N_i}\bullet$ to $\beta_i(e){}^{N}\bullet$ and Lemma 5.9 which shows that $\alpha_i(S_i) = M_0$. □

We have shown that it is possible to take a set of causal nets and construct a system net such that each causal net is a run of that system net, as long as the causal nets have one initially marked condition. What we did not show are the conditions under which the aggregation class is not empty. These conditions however, cannot be given based on a set of causal nets. Even if these causal nets belong to one causal set, this is still not enough. What we *can* show however, is that if we start from a *sound* WF-net as a system net, generate a set of runs and remove the labels of places, the original WF-net is in the aggregation class. For the full definition of soundness, we refer to [1, 4].

Property 5.11. (A system net is in the aggregation class of its runs) Let $N = (P, T, F, M_0)$ be a sound WF-net. We consider N to be a system net. Let $B = \{(N_i, \alpha_i, \beta_i) \mid 0 \leq i < n\}$ be the causal behavior of that system net, such that each (N_i, α_i, β_i) is a run of that system net, with $N_i = (C_i, E_i, K_i, S_i)$. Let $\mathcal{B} = \{\beta_i \mid 0 \leq i < n\}$ and $\Phi = \{N_i \mid 0 \leq i < n\}$ be a causal set. We show that $(N, \mathcal{B}) \in \mathcal{A}_\Phi$.

Proof. We show that all conditions of Definition 4.9 are satisfied.

1. $T = \bigcup_{0 \leq i < n} \mathrm{rng}(\beta_i)$ is the set of transitions. Since the WF-net is sound, there are no dead transitions thus implying that in its causal set each transition appears as an event at least once.
2. For all $p \in P$ holds that ${}^{N}\bullet p \cup p{}^{N}\bullet \neq \emptyset$. Since every sound WF-net is connected, this condition is satisfied,
3. $M_0 = [p_{ini}]$ and ${}^{N}\bullet p_{ini} = \emptyset$. Since N is a WF-net, there is exactly one place $p_{ini} \in P$, such that ${}^{N}\bullet p_{ini} = \emptyset$ and $M_0 = [p_{ini}]$,
4. $\mathcal{B}$ is the set of all labeling functions, i.e. $\mathcal{B} = \{\beta_i \mid 0 \leq i < n\}$.
5. For each causal net N_i, with $e \in E_i$ and $\beta_i(e) = t$ and ${}^{N_i}\bullet e = \{c\}$, holds that if $S_i(c) = 1$ then $p_{ini} \in {}^{N}\bullet t$. Since $S_i(c) = 1$, we know that ${}^{N_i}\bullet c = \emptyset$. Now assume $\alpha_i(c) = p$. The fact that for all $e' \in E_i$, α_i induces local bijections from $e'{}^{N_i}\bullet$ to $\beta_i(e'){}^{N}\bullet$ implies that ${}^{N}\bullet p = \emptyset$ and since N is a workflow net, this implies that $p = p_{ini}$. Moreover, the fact that for all α_i induces local bijections from ${}^{N_i}\bullet e$ to ${}^{N}\bullet t$ implies that $p_{ini} \in {}^{N}\bullet t$,
6. For each causal net N_i, with $e \in E_i$ and $\beta_i(e) = t$ holds that $|t{}^{N}\bullet| = |e{}^{N_i}\bullet|$ and $|{}^{N}\bullet t| = |{}^{N_i}\bullet e|$. Since α_i induces bijections from $e{}^{N_i}\bullet$ to $t{}^{N}\bullet$ and from ${}^{N_i}\bullet e$ to ${}^{N}\bullet t$, this condition is satisfied,

7. For each causal net N_i, with $e \in E_i$, $\beta_i(e) = t$ and $T' \subseteq T$ holds that $|t^{N}_{\bullet} \cap \bigcup_{t' \in T'} ({}^{N}_{\bullet}t')| \geq \sum_{e' \in E_i, \beta_i(e') \in T'} |e^{N_i}_{\bullet} \cap {}^{N_i}_{\bullet}e'|$. Let $e \in E_i$ with $\beta_i(e) = t$ and let $T' \subseteq T$. Assume that there $|t^{N}_{\bullet} \cap \bigcup_{t' \in T'} ({}^{N}_{\bullet}t')| = m$, i.e. there are m places between t and T'. Furthermore, assume that $\sum_{e' \in E_i, \beta_i(e') \in T'} |e^{N_i}_{\bullet} \cap {}^{N_i}_{\bullet} e'| < m$. Since for all $e' \in E_i$ with $beta_i(e_i) = t_i$, α_i induces local bijections from ${}^{N_i}_{\bullet}e_i$ to ${}^{N}_{\bullet}t_i$, we know that there are at least two $c_1, c_2 \in e^{N_i}_{\bullet}$ that are mapped onto the same $p \in P$. However, since $p \in t^{N}_{\bullet}$ this violates the local bijection property of α_i,
8. For each causal net N_i, with $e \in E_i$, $\beta_i(e) = t$ and $T' \subseteq T$ holds that $|\bigcup_{t' \in T'} (t'^{N}_{\bullet}) \cap {}^{N}_{\bullet}t| \geq \sum_{e' \in E_i, \beta_i(e') \in T'} |e'^{N_i}_{\bullet} \cap {}^{N_i}_{\bullet}e|$. The proof for this property is similar to the previous one.
9. For each causal net N_i and any segment (C'_i, E_{in}, E_{out}) of N_i holds that $\biguplus_{e \in E_{in}} (\beta_i(e)^{N}_{\bullet}) = \biguplus_{e \in E_{out}} ({}^{N}_{\bullet}\beta_i(e))$. This property relates to soundness. If one set of transitions produces tokens then these tokens will be consumed by another set of transitions (i.e. no tokens are "left behind" in the execution of a sound WF-net). The only exception is the transition that produces a token in the output place, but that transition cannot produce any tokens in any other place. Therefore, in each run, the input events of a segment will enable the output events of that segment.

□

The NLC algorithm takes a set of causal nets without condition labels as a starting point. From these nets, an aggregation class of WF-nets is defined. In this section, we have formally proven that every element of the aggregation class indeed is capable of constructing the causal nets used as input. Furthermore, if the runs were generated from some sound WF-net, then the WF-net itself is in that aggregation class.

6 Conclusion

In this paper, we looked at process mining from a new perspective. Instead of starting with a set of traces, we started with runs which constitute partial orders on events. We presented three algorithms to generate a Petri net from these runs. The first algorithm assumes that, for each run, all labels of both conditions and events are known. The second algorithm relaxes this by assuming that some transitions can have the same label (i.e. duplicate labels are allowed in the system net). This algorithm can also be used if only condition/place-labels were recorded. Finally, we provided an algorithm that does not require condition labels, i.e. the event/transition labels are known, the condition/place labels are unknown and duplicate transition labels are not allowed.

The results presented in this paper hold for a subclass of Petri nets, the so-called WF-nets. However, the first two algorithms presented here can easily be generalized to be applicable to any Petri net. For the third algorithm this can also be done, however, explicit knowledge about how the initial markings of

various runs relate is needed. When taking a set of runs as a starting point, this knowledge is not present in the general case.

References

1. van der Aalst, W.M.P.: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000)
2. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
3. van der Aalst, W.M.P., Alves de Medeiros, A.K., Weijters, A.J.M.M.T.: Genetic Process Mining. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 48–69. Springer, Heidelberg (2005)
4. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of Workflow Nets: Classification, Decidability, and Analysis. Formal Aspects of Computing 23(3), 333–363 (2011)
5. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., Alves de Medeiros, A.K., Song, M., Verbeek, H.M.W.: Business Process Mining: An Industrial Application. Information Systems 32(5), 713–732 (2007)
6. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering 47(2), 237–267 (2003)
7. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)
8. Agrawal, R., Gunopulos, D., Leymann, F.: Mining Process Models from Workflow Logs. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 469–483. Springer, Heidelberg (1998)
9. Badouel, E., Darondeau, P.: Theory of Regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
10. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. Fundamenta Informatica 88(1), 437–468 (2008)
11. Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Synthesis of Petri Nets from Term Based Representations of Infinite Partial Languages. Fundamenta Informatica 95(1), 187–217 (2009)
12. Cook, J.E., Wolf, A.L.: Discovering Models of Software Processes from Event-Based Data. ACM Transactions on Software Engineering and Methodology 7(3), 215–249 (1998)
13. Datta, A.: Automating the Discovery of As-Is Business Process Models: Probabilistic and Algorithmic Approaches. Information Systems Research 9(3), 275–301 (1998)
14. Desel, J.: Validation of Process Models by Construction of Process Nets. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 110–128. Springer, Heidelberg (2000)
15. Desel, J., Erwin, T.: Hybrid specifications: looking at workflows from a run-time perspective. Computer Systems Science and Engineering 5, 291–302 (2000)
16. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Informatica 33, 297–315 (1996)

17. van Dongen, B.F., van der Aalst, W.M.P.: Multi-phase Process Mining: Building Instance Graphs. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 362–376. Springer, Heidelberg (2004)
18. van Dongen, B.F., van der Aalst, W.M.P.: Multi-Phase Process Mining: Aggregating Instance Graphs into EPCs and Petri Nets. In: PNCWB 2005 Workshop, pp. 35–58 (2005)
19. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures - Part 1 and Part 2. Acta Informatica 27(4), 315–368 (1989)
20. Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Discovering Expressive Process Models by Clustering Log Traces. IEEE Transaction on Knowledge and Data Engineering 18(8), 1010–1027 (2006)
21. Harel, D., Kugler, H.-J., Pnueli, A.: Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements. In: Kreowski, H.-J., Montanari, U., Yu, Y., Rozenberg, G., Taentzer, G. (eds.) Formal Methods (Ehrig Festschrift). LNCS, vol. 3393, pp. 309–324. Springer, Heidelberg (2005)
22. Herbst, J.: A Machine Learning Approach to Workflow Management. In: Lopez de Mantaras, R., Plaza, E. (eds.) ECML 2000. LNCS (LNAI), vol. 1810, pp. 183–194. Springer, Heidelberg (2000)
23. Herbst, J., Karagiannis, D.: Workflow mining with InWoLvE. Computers in Industry 53(3), 245–264 (2004)
24. Lorenz, R., Juhás, G.: Towards Synthesis of Petri Nets from Scenarios. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 302–321. Springer, Heidelberg (2006)
25. Alves de Medeiros, A.K., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic Process Mining: An Experimental Evaluation. Data Mining and Knowledge Discovery 14(2), 245–304 (2007)
26. Roychoudhury, A., Thiagarajan, P.S.: Communicating Transaction Processes. In: Lilius, J., Balarin, F., Machado, R. (eds.) Proceedings of Third International Conference on Application of Concurrency to System Design (ACSD 2003), pp. 157–166. IEEE Computer Society (2003)
27. Smith, E.: On Net Systems Generated by Process Foldings. In: Rozenberg, G. (ed.) APN 1991. LNCS, vol. 524, pp. 253–276. Springer, Heidelberg (1991)
28. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering Workflow Models from Event-Based Data using Little Thumb. Integrated Computer-Aided Engineering 10(2), 151–162 (2003)
29. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process Discovery using Integer Linear Programming. Fundamenta Informaticae 94, 387–412 (2010)