Fundamenta Informaticae 168 (2019) 1–27
DOI 10.3233/FI-2019-1800
IOS Press

Lucent Process Models and Translucent Event Logs

Wil M.P. van der Aalst[*]
Process and Data Science (PADS)
RWTH Aachen University, Germany
wvdaalst@pads.rwth-aachen.de

Abstract. A process model is *lucent* if no two reachable states are enabling the same set of activities. An event log is *translucent* if each event carries information about the set of activities enabled when the event occurred (normally one only sees the activity performed). Both lucency and translucency focus on the set of *enabled activities* and are therefore related. Surprisingly, these notions have not been investigated before. This paper aims to (1) characterize process models that are lucent, (2) provide a discovery approach to learn process models from translucent event logs, and (3) relate lucency and translucency. Lucency is defined both in terms of automata and Petri nets. A marked Petri net is *lucent* if there are no two different reachable markings enabling the same set of transitions, i.e., states are fully characterized by the transitions they enable. We will also provide a novel *process discovery technique* starting from a translucent event log. It turns out that information about the set of activities is extremely valuable for process discovery. We will provide sufficient conditions to ensure that the discovered model is lucent and show that a translucent event log sampled from a lucent process model can be used to rediscover the original model. We anticipate new analysis techniques exploiting lucency. Moreover, as shown in this paper, translucent event logs provide valuable information that can be exploited by a new breed to process mining techniques.

Keywords: Process mining, Petri nets, lucent process models, translucent event logs

1. Introduction

First, we informally introduce the lucency and translucency notions. Then, we briefly discuss related work and provide an outline of the paper.

[*]Address for correspondence: Process and Data Science (PADS), RWTH Aachen University, Germany

1.1. Lucent process models

A process model is *lucent* if and only if there do not exist two states that enable the same set of activities. Consider, for example, the marked Petri net AN_1 and automaton AM_1 in Figure 1. Both describe the same process that can generate traces such as $\langle a, b, c, e\rangle$, $\langle a, c, b, e\rangle$, $\langle a, b, c, d, c, b, e\rangle$, and $\langle a, c, b, d, b, c, d, c, b, e\rangle$. There are six reachable states (called markings in the context of Petri net). Clearly, there are no two states enabling the same set of activities (represented by the five transitions in the Petri net). For example, state s_1 in AM_1 (corresponding to marking $[p2, p3]$ in AN_1) enables b and c (corresponding to transitions $t2$ and $t3$ in AN_1) There is no other reachable state enabling the activity set $\{b, c\}$. Therefore, both AN_1 and AM_1 are lucent.

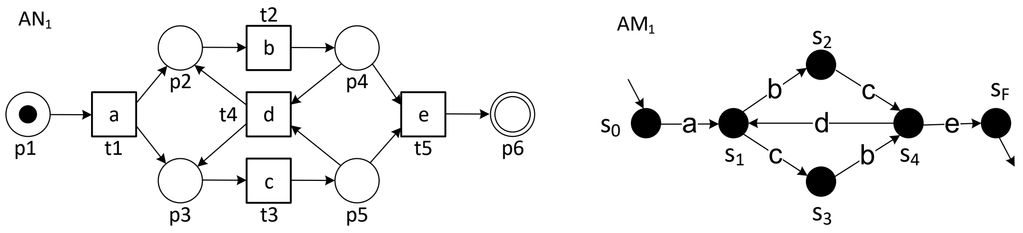


Figure 1. An accepting Petri net $AN_1 = (N_1, [p1], [p6])$ and accepting automaton AM_1. Both are lucent.

Now consider the marked Petri net AN_2 and automaton AM_2 in Figure 2. These are not lucent. For example, state s_1 in AM_2 (corresponding to marking $[p2, p5]$ in AN_2) enables c. However, also state s_2 (corresponding to marking $[p2, p6]$) enables c.

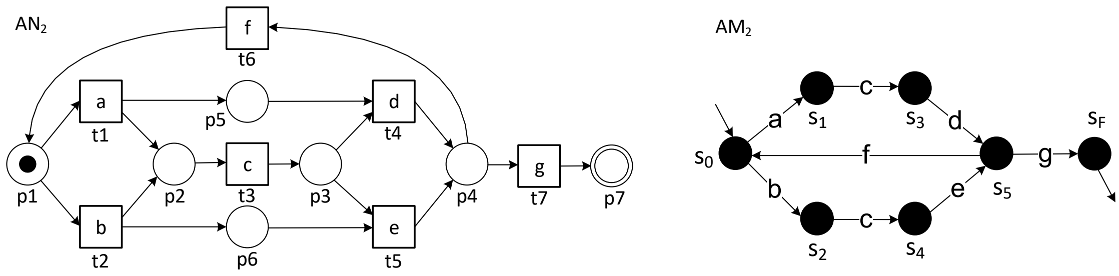


Figure 2. Another accepting Petri net $AN_2 = (N_2, [p1], [p7])$ and accepting automaton AM_2. Both are not lucent because there are two markings/states enabling the same set of activities $\{c\}$.

Lucency is a general notion that is independent of the modeling language used. Here, we only consider automata and Petri nets. However, most process modeling languages (statecharts, activity diagrams, EPCs, BPMN, process calculi, etc.) have a state notion where one can reason about the set of enabled activities (also called actions of labels) when the model is in a particular state. Luceny triggers interesting questions such as: How to exploit lucency during analysis? and How to check lucency efficiently? It also triggers the question: What is the class of Petri nets for which each marking is uniquely identified by the set of enabled transitions? A lucent marked Petri net cannot have two different reachable markings that enable the same set of transitions.

Table 1. A fragment of an event log corresponding to AN_1 and AM_1 in Figure 1.

event	case	activity	time	enabled
e_1	1	a	09:22	$\{a\}$
e_2	1	b	09:34	$\{b,c\}$
e_3	2	a	09:45	$\{a\}$
e_4	2	c	10:12	$\{b,c\}$
e_5	1	c	10:17	$\{c\}$
e_6	1	e	11:06	$\{d,e\}$
e_7	2	b	11:22	$\{b\}$
e_8	3	a	11:55	$\{a\}$
e_9	3	c	12:13	$\{b,c\}$
e_{10}	2	d	12:18	$\{d,e\}$
e_{11}	2	b	13:32	$\{b,c\}$
e_{12}	2	c	13:43	$\{c\}$
e_{13}	3	b	13:52	$\{b\}$
e_{14}	2	e	14:17	$\{d,e\}$
e_{15}	3	e	14:20	$\{d,e\}$
...	...	...	...	...

Table 2. An event log corresponding to AN_2 and AM_2 in Figure 2.

event	case	activity	time	enabled
e_1	1	a	11:07	$\{a,b\}$
e_2	1	c	11:34	$\{c\}$
e_3	2	b	11:45	$\{a,b\}$
e_4	2	c	12:12	$\{c\}$
e_5	2	e	12:17	$\{e\}$
e_6	1	d	13:06	$\{d\}$
e_7	1	g	13:22	$\{f,g\}$
e_8	2	f	13:55	$\{f,g\}$
e_9	2	a	14:13	$\{a,b\}$
e_{10}	3	b	14:18	$\{a,b\}$
e_{11}	3	c	15:32	$\{c\}$
e_{12}	2	c	15:43	$\{c\}$
e_{13}	2	d	15:52	$\{d\}$
e_{14}	2	g	16:17	$\{f,g\}$
e_{15}	3	e	16:20	$\{e\}$
e_{16}	3	g	16:25	$\{f,g\}$

In [1] it was shown that *perpetual marked free-choice nets* are always lucent. These nets are live and bounded and also have a so-called *regeneration marking*. A regeneration marking serves as a "regeneration point", i.e., a state where all tokens mark a single cluster.

1.2. Translucent event logs

An event log is *translucent* if each event carries information about the set of activities possible. In traditional event logs, each event has at least three attributes: *case* (the process instance to which the event belongs, e.g., an order, customer, or patient), *activity* (the action performed, e.g., "register request" or "make payment"), and *time* (the time at which the event occurred, e.g., 2019-01-29T19:20+01:00). In a translucent event log, each event has an additional attribute listing the set of activities enabled.

Consider for example Table 1 showing an event log generated from the process described in Figure 1. Each row corresponds to an event. The last column shows the set of activities enabled. Consider for example event e_4 which represents the occurrence of activity c for case 2 at time 10:12. When e_4 occurred activities b and c were enabled. Note that Table 1 is consistent with both AN_1 and AM_1 in Figure 1. Case 1 corresponds to the trace $\langle a, b, c, e \rangle$. We can also list all activities and make the one selected bold: $\langle \underline{a}, \underline{b}c, \underline{c}, d\underline{e} \rangle$. Cases 2 and 3 can also be represented in this way: $\langle \underline{a}, b\underline{c}, \underline{b}, \underline{d}e, \underline{b}c, \underline{c}, d\underline{e} \rangle$ and $\langle \underline{a}, b\underline{c}, \underline{b}, d\underline{e} \rangle$.

Table 2 shows an event log generated from the process described in Figure 2. The three cases can be compactly represented using the notation introduced before: $\langle \underline{a}b, \underline{c}, \underline{d}, f\underline{g} \rangle$, $\langle a\underline{b}, \underline{c}, \underline{e}, \underline{f}g, \underline{a}b, \underline{c}, \underline{d}, f\underline{g} \rangle$, and $\langle a\underline{b}, \underline{c}, \underline{e}, f\underline{g} \rangle$.

This paper presents a discovery algorithm tailored toward translucent event logs. Based on Table 1, our algorithm is able to rediscover both AN_1 and AM_1 in Figure 1 (modulo the renaming of states and places). The information in Table 2 is not sufficient to discover the models in Figure 2. Our discovery algorithm will provide an underfitting process model because it is unable to distinguish the two states enabling c. This illustrates that lucency and translucency are related. Therefore, this paper investigates this relationship in detail.

1.3. Related work

This paper extends the work presented [1] which only considers lucency in a Petri net setting.[1] This was the first paper that defined and characterized lucency. The results for perpetual marked free-choice nets presented later, build on "structure theory", a branch in Petri nets [2, 3, 4, 5, 6] that asks what behavioral properties can be derived from its structural properties [7, 8, 9]. Many different subclasses have been studied. Examples include state machines, marked graphs, free-choice nets, asymmetric choice nets, and nets without TP and PT handles. Structure theory also studies local structures such as traps and siphons that may reveal information about the behavior of the Petri net and includes linear algebraic characterizations of behavior involving the matrix equation or invariants [8, 9, 3].

Free-choice nets are well studied [10, 8, 11, 12]. The definite book on the structure theory of free-choice nets is [9]. Also, see [8] for pointers to literature. Therefore, it is surprising that the question whether markings are uniquely identified by the set of enabled transitions (i.e., lucency) has not been explored in literature. Lucency is unrelated to the so-called "frozen tokens" [13]. A Petri net has a frozen token if there exists an infinite occurrence sequence never using the token. It is possible to construct live and bounded free-choice nets that are lucent while having frozen tokens. Conversely, there are live and bounded free-choice nets that do not have frozen tokens and are not lucent. Most related to the results presented in this paper is the work on the so-called *blocking theorem* [14, 15]. Blocking markings are reachable markings which enable transitions from only a single cluster. Removing the cluster yields a dead marking. The blocking theorem states that in a bounded and live free-choice net each cluster has a unique blocking marking.

In [1], we did not cover the notion of translucency, but mentioned that lucency is interesting from a process mining point of view. The field of process mining [16] studies problems such a process discovery (learning process models from event logs) [17, 18, 19, 20, 21, 22, 23] and conformance checking (analyzing discrepancies between observed and modeled behavior) [24, 25]. Unlike traditional synthesis approaches [26, 27, 28], the input (event log) is known to be a sample of the possible behavior (like in data mining and machine learning). As far as we know, there are no process mining techniques based on translucent event logs. Therefore, it is impossible to compare our process discovery algorithm to existing approaches not using enabling information. For an overview of process mining techniques, we refer to [16].

1.4. Outline

The remainder of this paper is organized as follows. Section 2 introduces preliminaries (automata, Petri nets, soundness, subclasses, clusters, components, short-circuiting, etc.) and known results (e.g.,

[1]The paper won the best paper award at Petri Nets 2018 and the author was invited to provide this substantially extended version.

the blocking theorem). Accepting Petri nets with a predefined initial and final marking are related to accepting automata. Section 3 defines lucency as a (desirable) behavioral property for both marked Petri nets and automata. Translucent event logs are introduced in Section 4. In Section 5, we present a novel discovery algorithm and show the value of translucency. The section also explores the relation between lucency and translucency. Section 6 discusses techniques to discover Petri nets from translucent event data. Section 7 concludes the paper.

2. Preliminaries

This section introduces basic concepts related to Petri nets, subclasses of nets (e.g., free-choice nets and workflow nets), and blocking markings.

2.1. Multisets, sequences, and functions

$\mathcal{P}(A) = \{X \mid X \subseteq A\}$ is the *powerset* of A, i.e., all subsets of A. $\mathcal{B}(A)$ is the set of all *multisets* over some set A. For some multiset $b \in \mathcal{B}(A)$, $b(a)$ denotes the number of times element $a \in A$ appears in b. Some examples: $b_1 = [\,]$, $b_2 = [x, x, y]$, $b_3 = [x, y, z]$, $b_4 = [x, x, y, x, y, z]$, and $b_5 = [x^3, y^2, z]$ are multisets over $A = \{x, y, z\}$. b_1 is the empty multiset, b_2 and b_3 both consist of three elements, and $b_4 = b_5$, i.e., the ordering of elements is irrelevant and a more compact notation may be used for repeating elements. The standard set operators can be extended to multisets, e.g., $x \in b_2$, $b_2 \uplus b_3 = b_4$, $b_5 \setminus b_2 = b_3$, $|b_5| = 6$, etc. $\{a \in b\}$ denotes the set with all elements a for which $b(a) \geq 1$. $b \leq b'$ if $b(a) \leq b'(a)$ for all $a \in A$. Hence, $b_3 \leq b_4$ and $b_2 \not\leq b_3$ (because b_2 has two x's). $b < b'$ if $b \leq b'$ and $b \neq b'$. Hence, $b_3 < b_4$ and $b_4 \not< b_5$ (because $b_4 = b_5$).

$\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in X^*$ denotes a *sequence* over X of length $|\sigma| = n$. $\sigma_i = a_i$ for $1 \leq i \leq |\sigma|$. $\langle\,\rangle$ is the empty sequence. Sequences can be concatenated using "$\cdot$", e.g., $\langle a, b \rangle \cdot \langle b, a \rangle = \langle a, b, b, a \rangle$. It is also possible to project sequences: $\langle a, b, b, a, c, d \rangle\!\upharpoonright_{\{a,c\}} = \langle a, a, c \rangle$.

Definition 2.1. (Functions Applied to Sets, Sequences, and Multisets)
Let $f \in X \to Y$, $Z \in \mathcal{P}(X)$, $\sigma \in X^*$, and $b \in \mathcal{B}(X)$. $f(Z) = \{f(x) \mid x \in Z\}$, $f(\sigma) = \langle f(\sigma_1), f(\sigma_2), \ldots, f(\sigma_{|\sigma|)}) \rangle$, $f(b) = [f(x) \mid x \in b]$, i.e., the multiset where element $f(x)$ appears $\sum_{y \in b \mid f(x)=f(y)} b(y)$ times. If f is a partial function, i.e., $f \in X \not\to Y$, we drop the elements not in $dom(f)$.

Hence, functions can be applied to sets, sequences, and multisets. For example, consider the function f with $dom(f) = \{a, b\}$ and $f(a) = x$ and $f(b) = y$. $f(\{a, b, c, d\}) = \{x, y\}$, $f(\langle a, b, c, d, b, a \rangle) = \langle x, y, y, x \rangle$, and $f([a^2, b^2, c, d]) = [x^2, y^2]$. Functions can also be applied to nested structures like multisets of sequences, e.g., $f([\langle a, b, c, d, b, a \rangle^2, \langle c, b, a \rangle^3]) = [\langle x, y, y, x \rangle^2, \langle y, x \rangle^3]$.

2.2. Automata

We use *automata* to describe behavior and use *accepting automata* when we are interested in the traces that lead from the initial state to the final state (see figures 1 and 2 for two example automata).

Definition 2.2. (Automaton)
An *automaton* is formally represented by the four-tuple $AM = (S, A, \delta, s_0)$. S is a non-empty set of states, A is a non-empty alphabet (also referred to as actions or activities), $\delta \subseteq S \times A \times S$ is the transition relation, and $s_0 \in S$ is the initial state. An *accepting* automaton is a five-tuple $AM = (S, A, \delta, s_0, s_F)$ adding a final state s_F. An (accepting) automaton is *deterministic* if for any $\{(s_1, a, s_2), (s_1, a, s_3)\} \subseteq \delta$: $s_2 = s_3$. An (accepting) automaton is *finite* if S is finite.

In the remainder, we will refer to $\mathcal{A}$ as the *universe of activities* and $A \subseteq \mathcal{A}$ as a concrete set of activities. Note that in the context of automata, terms like *label* and *action* are more common. However, since we relate process models to event logs, we use the term *activity*.

Definition 2.3. (Enabling, Path, and Reachability)
Let AM be an (accepting) automaton with states S, alphabet A, and transition relation $\delta \subseteq S \times A \times S$. Activity $a \in A$ is *enabled* in state $s \in S$, denoted as $(AM, s)[a\rangle$, if there exists a state $s' \in S$ such that $(s, a, s') \in \delta$. $en(AM, s) = \{a \in A \mid (AM, s)[a\rangle\}$ is the set of enabled activities in s.

$s \xrightarrow{(\rho,\sigma)}_{AM} s'$ with $s, s' \in S$, $\rho \in S^*$, and $\sigma \in A^*$ denotes that there is an $n \geq 0$ such that $|\rho| = |\sigma| + 1 = n + 1$ such that $\rho_1 = s$, $\rho_{n+1} = s'$, and for all $1 \leq i \leq n$: $(s_i, a_i, s_{i+1}) \in \delta$. When AM is clear from the context, we simply write $s \xrightarrow{(\rho,\sigma)} s'$. (ρ, σ) is called a *path* leading from state s to state s'. $(AM, s)[\sigma\rangle(AM, s')$ if any only if there is a ρ such that $s \xrightarrow{(\rho,\sigma)}_{AM} s'$. $R(AM, s) = \{s' \in S \mid \exists_{(\rho,\sigma)}\ s \xrightarrow{(\rho,\sigma)}_{AM} s'\}$ is the set of all states reachable from state s.

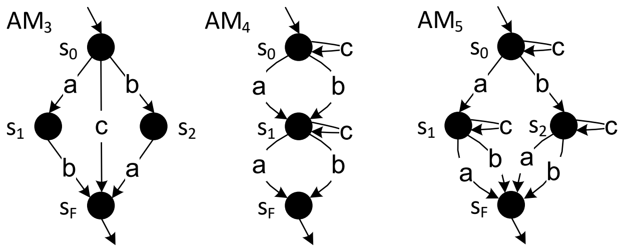


Figure 3. Three accepting automata AM_3, AM_4, and AM_5 (the initial state is s_0 and the final state is s_F).

Figure 3 shows three accepting automata. $en(AM_3, s_0)) = en(AM_4, s_0)) = en(AM_5, s_0)) = \{a, b, c\}$. $R(AM_3, s_0) = R(AM_5, s_0) = \{s_0, s_1, s_2, s_F\}$. $R(AM_4, s_0) = \{s_0, s_1, s_F\}$.

In this paper, we assume that automata only contain reachable states, i.e., $S = R(AM, s_0)$. However, we sometimes need a stronger property: *soundness*.

Definition 2.4. (Sound Accepting Automaton)
Let $AM = (S, A, \delta, s_0, s_F)$ be an accepting automaton. AM is *sound* if each state $s \in S$ is reachable (i.e., $S = R(AM, s_0)$), each activity $a \in A$ can occur (i.e., there exists $s \in R(AM, s_0)$ such that $a \in en(AM, s)$), the final state is reachable from any reachable state (i.e., for any $s \in R(AM, s_0)$: $s_F \in R(AM, s)$), and the final state is dead (i.e., $en(AM, s_F) = \emptyset$).

The three automata in Figure 3 are sound. In the remainder, we focus on sound accepting automata. A non-sound automaton can be made sound by removing states and activities not on a path from s_0 to s_F and adding a "stop" activity.

Definition 2.5. (Complete Paths)
Let $AM = (S, A, \delta, s_0, s_F)$ be an accepting automaton. $(\rho, \sigma) \in S^* \times A^*$ is a *complete path* if $s_0 \xrightarrow{(\rho,\sigma)}_{AM} s_F$. $\Phi(AM) = \{(\rho, \sigma) \in S^* \times A^* \mid s_0 \xrightarrow{(\rho,\sigma)}_{AM} s_F\}$ is the set of all complete paths.

Complete paths start in the initial state s_0 and end in the final state s_F. Consider AM_3, AM_4, and AM_5 in Figure 3. $\Phi(AM_3) = \{(\langle s_0, s_1, s_F\rangle, \langle a, b\rangle), (\langle s_0, s_2, s_F\rangle, \langle b, a\rangle), (\langle s_0, s_F\rangle, \langle c\rangle)\}$. $\Phi(AM_4)$ and $\Phi(AM_5)$ are infinite due to the loops involving activity c. Some examples of complete paths: $(\langle s_0, s_1, s_F\rangle, \langle a, a\rangle) \in \Phi(AM_4)$, $(\langle s_0, s_0, s_1, s_1, s_F\rangle, \langle c, a, c, a\rangle) \in \Phi(AM_4)$, and $(\langle s_0, s_2, s_2, s_2, s_F\rangle, \langle b, c, c, b\rangle) \in \Phi(AM_5)$. A set or multiset of complete paths is *transition-complete* if all transitions $(s, a, s') \in \delta$ occur at least once.

Definition 2.6. (Transition-Complete)
Let $AM = (S, A, \delta, s_0, s_F)$ be an accepting automaton and $X \subseteq \Phi(AM)$ a set of complete paths. X is *transition-complete* if $\delta = \{(s, a, s') \mid \exists_{(\rho,\sigma)\in X} \exists_{1\leq i\leq|\sigma|} (s, a, s') = (\rho_i, \sigma_i, \rho_{i+1})\}$. Multiset $X \in \mathcal{B}(\Phi(AM))$ is *transition-complete* if the corresponding set $\{\sigma \in X\}$ is transition-complete.

$X = \{(\langle s_0, s_0, s_1, s_1, s_F\rangle, \langle c, a, c, a\rangle), (\langle s_0, s_1, s_F\rangle, \langle b, b\rangle)\} \subseteq \Phi(AM_4)$ is transition-complete for AM_4. $Y = \{(\langle s_0, s_1, s_F\rangle, \langle a, b\rangle), (\langle s_0, s_F\rangle, \langle c\rangle)\} \subseteq \Phi(AM_3)$ is not transition-complete for AM_3 since (s_0, b, s_2) and (s_2, a, s_F) are not covered.

Later we will use the following two standard equivalence notions (isomorphic and bisimilar).

Definition 2.7. (Isomorphic and Bisimilar)
Let $AM_1 = (S^1, A^1, \delta^1, s_0^1, s_F^1)$ and $AM_2 = (S^2, A^2, \delta^2, s_0^2, s_F^2)$ be two accepting automata. AM_1 and AM_2 are *isomorphic* if there is a bijective mapping $h \in S^1 \to S^2$ such that $h(s_0^1) = s_0^2$, $h(s_F^1) = s_F^2$ and $(s, a, s') \in \delta^1$ if and only if $(h(s), a, h(s')) \in \delta^2$. AM_1 and AM_2 are *bisimilar* if there is a relation $R \subseteq S^1 \times S^2$ such that (1) $(s_0^1, s_0^2) \in R$, (2) $(s_F^1, s_F^2) \in R$, (3) for any $(s_1, a, s_1') \in \delta^1$ and $s_2 \in S^2$ such that $(s_1, s_2) \in R$, there is an $s_2' \in S^2$ such that $(s_1', s_2') \in R$ and $(s_2, a, s_2') \in \delta^2$, and (4) for any $(s_2, a, s_2') \in \delta^2$ and $s_1 \in S^1$ such that $(s_1, s_2) \in R$, there is an $s_1' \in S^1$ such that $(s_1', s_2') \in R$ and $(s_1, a, s_1') \in \delta^1$.

When two accepting automata are isomorphic, they are also bisimilar. The reverse does not hold, e.g., AM_4 and AM_5 in Figure 3 are bisimilar but not isomorphic.

2.3. Petri nets

Next to automata, we use Petri nets and introduce a few basic concepts and standard notations.

Definition 2.8. (Petri Net)
A Petri net is a tuple $N = (P, T, F)$ with P the non-empty set of places, T the non-empty set of transitions such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ the flow relation such that the graph $(P \cup T, F)$ is connected.

Definition 2.9. (Marking)
Let $N = (P, T, F)$ be a Petri net. A marking M is a multiset of places, i.e., $M \in \mathcal{B}(P)$. (N, M) is a marked net.

For a subset of places $X \subseteq P$: $M\!\upharpoonright_X = [p \in M \mid p \in X]$ is the marking *projected* on this subset.

A Petri net $N = (P, T, F)$ defines a directed graph with nodes $P \cup T$ and edges F. For any $x \in P \cup T$, $\bullet x = \{y \mid (y, x) \in F\}$ denotes the set of input nodes and $x\bullet = \{y \mid (x, y) \in F\}$ denotes the set of output nodes. The notation can be generalized to sets: $\bullet X = \{y \mid \exists_{x\in X}\ (y, x) \in F\}$ and $X\bullet = \{y \mid \exists_{x\in X}\ (x, y) \in F\}$ for any $X \subseteq P \cup T$. We add the net N as superscript when confusion is possible: $\overset{N}{\bullet}$. A *path* in a Petri net $N = (P, T, F)$ is a sequence of nodes $\rho = \langle x_1, x_2, \ldots, x_n\rangle$ such that $(x_i, x_{i+1}) \in F$ for $1 \leq i < n$. Hence, $\bullet x_i = x_{i-1}$ for $1 < i \leq n$ and $x_i\bullet = x_{i+1}$ for $1 \leq i < n$. ρ is an *elementary path* if $x_i \neq x_j$ for $1 \leq i < j \leq n$.

A transition $t \in T$ is *enabled* in marking M of net N, denoted as $(N, M)[t\rangle$, if each of its input places $\bullet t$ contains at least one token. $en(N, M) = \{t \in T \mid (N, M)[t\rangle\}$ is the set of enabled transitions.

An enabled transition t may *fire*, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the output places $t\bullet$. Formally: $M' = (M \setminus \bullet t) \uplus t\bullet$ is the marking resulting from firing enabled transition t in marking M of Petri net N. $(N, M)[t\rangle(N, M')$ denotes that t is enabled in M and firing t results in marking M'.

Let $\sigma = \langle t_1, t_2, \ldots, t_n\rangle \in T^*$ be a sequence of transitions. $(N, M)[\sigma\rangle(N, M')$ denotes that there is a set of markings $M_1, M_2, \ldots, M_{n+1}$ $(n \geq 0)$ such that $M_1 = M$, $M_{n+1} = M'$, and $(N, M_i)[t_i\rangle(N, M_{i+1})$ for $1 \leq i \leq n$. A marking M' is *reachable* from M if there exists a *firing sequence* σ such that $(N, M)[\sigma\rangle(N, M')$. $R(N, M) = \{M' \in \mathcal{B}(P) \mid \exists_{\sigma\in T^*}\ (N, M)[\sigma\rangle(N, M')\}$ is the set of all reachable markings.

Figure 1 shows a marked Petri net having 6 places and 5 transitions. Transitions $t2$ and $t3$ are enabled in marking $M = [p2, p3]$. $R(N, [p1]) = \{[p1], [p2, p3], [p3, p4], [p2, p5], [p4, p5], [p6]\}$. The firing sequence $\langle t1, t2, t3, t4\rangle$ leads from marking $[p1]$ to marking $[p2, p3]$, i.e., $(N, [p1])[\langle t1, t2, t3, t4\rangle\rangle$ $(N, [p2, p3])$.

2.4. Accepting and/or labeled Petri nets

The counterpart of an accepting automaton is an accepting Petri net having a predefined *final marking*.

Definition 2.10. (Accepting Petri Net)
An *accepting* Petri net $AN = (N, M, M_F)$ is a Petri net $N = (P, T, F)$ with an initial marking $M \in \mathcal{B}(P)$ and a final marking $M_F \in \mathcal{B}(P)$. AN accepts $\sigma = \langle a_1, a_2, \ldots, a_n\rangle \in T^*$ if $(AN, M)[\sigma\rangle(AN, M_F)$.

Figures 1 and 2 show two accepting Petri nets AN_1 and AN_2. The tokens indicate the initial markings ($[p1]$ in both models). The places involved in the final marking have a double border, i.e., $[p6]$ is the final marking of AN_1 and $[p7]$ is the final marking of AN_2.

When analyzing accepting Petri nets we often "short-circuit" the model to make it cyclic, i.e., when reaching the final marking the net can return to the initial state. Figure 4 shows how accepting Petri net AN_3 is short-circuited resulting in $\overline{AN_3}$ which includes transition t^*.

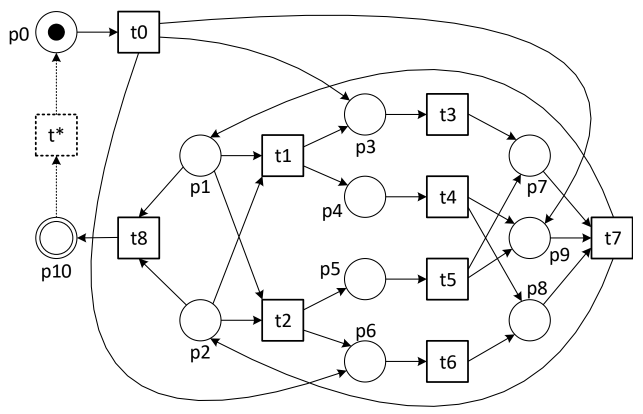


Figure 4. The Petri net without transition t^* is an accepting Petri net $AN_3 = (N_3, [p0], [p10])$ with initial marking $[p0]$ and final marking $[p10]$. The short-circuited net $\overline{AN_3} = (\overline{N_3}, [p0])$ adds transition t^* to make the net cyclic.

Definition 2.11. (Short-Circuited Net)
Let $AN = (N, M, M_F)$ be an accepting Petri net. The corresponding short-circuited marked Petri net is $\overline{AN} = (\overline{N}, M)$ with $\overline{N} = (P, T \cup \{t^*\}, F \cup (\{t^*\} \times \{p \in M\}) \cup (\{p \in M_F\} \times \{t^*\})$ and $t^* \notin T$.

In a similar way, we can short-circuit the two accepting Petri nets AN_1 and AN_2 in figures 1 and 2 and obtain $\overline{AN_1}$ and $\overline{AN_2}$. In these examples, the initial and final markings have only one token. However, the definition allows for any initial and final marking. Hence, this extends the notion of short-circuiting used in the context of *workflow nets* [29, 30]. Just like for workflow nets and accepting automata, we define a *soundness* notion for accepting Petri nets.

Definition 2.12. (Sound Accepting Petri net)
Let $AN = (N, M, M_F)$ be an accepting Petri net. AN is *sound* if (1) for each $t \in T$, there exists a $M' \in R(N, M)$ such that $t \in en(N, M')$, (2) for all $M' \in R(N, M)$, $M_F \in R(N, M')$, (3) for all $M' \in R(N, M)$, $M' \not> M_F$, (4) for all $p \in P$, $M(p) \leq 1$ and $M_F(p) \leq 1$, and (5) for all $p \in M_F$, $p\bullet = \emptyset$.

The first requirement states that there should be no dead transitions. The second requirement ensures that it is always possible to reach the final marking. The third requirement ensures that there are no markings "dominating" M_F. The fourth requirement states that the initial and final markings are "safe" (at most one token in a place), The fifth requirement states that places in the final marking have no output arcs (to ensure that the final marking is dead). The three accepting Petri nets AN_1, AN_2, and AN_3 shown thus far, are all sound.

Transitions can be labeled. Figures 1 and 2 already showed transition labels $\{a, b, \ldots\}$ next to the transition identifiers $\{t1, t2, \ldots\}$. For example, $l(t1) = a$, $l(t2) = b$, etc.

Definition 2.13. (Labeling function)
Let $N = (P, T, F)$ be a Petri net and A a set of activities. Labeling function $l \in T \to A$ is a mapping

relating each transition to an activity. N_l is the corresponding labeled Petri net. AN_l is a labeled accepting Petri net.

Note that labeling function l does not need to be injective, i.e., there can be multiple transitions mapped onto the same activity. Using Definition 2.1, we can apply function l to sets, multisets, and sequences of transitions. For $X = \{t_1, t_2, \ldots, t_n\}$, $l(X) = \{l(t_1), l(t_2), \ldots, l(t_n)\}$. For $\sigma = \langle t_1, t_2, \ldots, t_n \rangle$, $l(\sigma) = \langle l(t_1), l(t_2), \ldots, l(t_n) \rangle$. We extend this also to sets and multisets of sequences. For $Y \subseteq T^*$, $l(Y) = \{l(\sigma) \mid \sigma \in Y\}$. For $Z \in \mathcal{B}(T^*)$, $l(Z) = [l(\sigma) \mid \sigma \in Z]$. This provides a general way to convert transition names into activity names in different contexts, e.g., $en(N_l, M) = en_l(N, M) = l(en(N, M))$.

Figures 1 and 2 already illustrated that accepting Petri nets can be transformed into accepting automata. The corresponds to the standard notion of a *reachability graph*.

Definition 2.14. (Mapping Petri Nets onto Automata)
A marked Petri net (N, M) defines an automaton $aut(N, M) = (S, A, \delta, s_0)$ with $S = R(N, M)$, $A = T$, $\delta = \{(s, a, s') \in S \times A \times S \mid (N, s)[a\rangle(N, s')\}$, and $s_0 = M$. A accepting Petri net (N, M, M_F) defines an accepting automaton $aut(N, M, M_F) = (S, A, \delta, s_0, s_F)$ with additionally $s_F = M_F$. If there is a labeling function $l \in T \to A$, then $aut(N_l, M) = aut_l(N, M) = (S, A, \delta, s_0)$ with $S = R(N, M)$, $A = l(T)$, $\delta = \{(s, l(t), s') \in S \times A \times S \mid (N, s)[t\rangle(N, s')\}$, and $s_0 = M$. $aut(N_l, M, M_F) = aut_l(N, M, M_F) = (S, A, \delta, s_0, s_F)$ adds s_F.

The accepting Petri nets in figures 1 and 2 are sound. Therefore, the corresponding automata depicted are also sound. It is easy to verify that this is always the case.

Observation 2.15. Let (N, M) marked Petri net and $aut(N, M)$ the corresponding accepting automaton. If (N, M) is sound (Definition 2.12), then $aut(N, M)$ is sound (Definition 2.4).

The concept of lucency defined later is related to the notion of a *blocking marking*. To explain this notion we need to introduce the notion of a cluster and free-choice nets.

The *cluster* of node x, denoted $[x]_c$ is the smallest set such that (1) $x \in [x]_c$, (2) if $p \in [x]_c \cap P$, then $p\bullet \subseteq [x]_c$, and (3) if $t \in [x]_c \cap T$, then $\bullet t \subseteq [x]_c$. $[N]_c = \{[x]_c \mid x \in P \cup T\}$ is the set of clusters of N.

$N = (P, T, F)$ is a *free-choice net* if for any for any $t_1, t_2 \in T$: $\bullet t_1 = \bullet t_2$ or $\bullet t_1 \cap \bullet t_2 = \emptyset$. Free-choice nets are probably the best studied subclass of Petri nets [10, 9, 8, 11, 12].

In a blocking marking, the transitions in a particular cluster are enabled while all transitions outside the cluster are disabled. Formally, a blocking marking for cluster C is a marking $M_B \in R(N, M)$ such that $en(N, M_B) = T(C)$, i.e., all transitions in the cluster are enabled, but no other transitions.

In [31] Genrich and Thiagarajan showed that unique blocking markings exist for all clusters in live and safe marked graphs. This was generalized by Gaujal, Haar, and Mairesse in [14] where they showed that blocking markings exist and are unique in live and bounded free-choice nets. Note that in a free-choice net all transitions in the cluster are enabled simultaneously (or all are disabled). There is one unique marking in which precisely one cluster is enabled. Moreover, one can reach this marking without firing transitions from the cluster that needs to become enabled. A simplified proof was given in [15] and another proof sketch can be found in [8].

3. Lucent process models

This paper focuses on process models whose states are uniquely identified based on the activities they enable. Hence, there cannot be two different states enabling the same set of activities. Such process models are called *lucent*.

Definition 3.1. (Lucent Automata)
Let $AM = (S, A, \delta, s_0)$ be an automaton. AM is *lucent* if and only if for any $s_1, s_2 \in R(AM, s_0)$: $en(AM, s_1) = en(AM, s_2)$ implies $s_1 = s_2$.

Automaton AM_1 in Figure 1 is lucent. Automaton AM_2 in Figure 2 is not lucent because $en(AM_2, s_1) = en(AM_2, s_2) = \{c\}$. Automaton AM_3 in Figure 3 is lucent, but the other two automata AM_4 and AM_5 are not (there are multiple states enabling a, b, and c).

Theorem 3.2. (Bisimilar Lucent Automata are Isomorphic)
Let $AM_1 = (S^1, A^1, \delta^1, s_0^1)$ and $AM_2 = (S^2, A^2, \delta^2, s_0^2)$ be two bisimilar sound lucent accepting automata. AM_1 and AM_2 are isomorphic.

Proof:
Assume there is a relation $R \subseteq S^1 \times S^2$ such that (1) $(s_0^1, s_0^2) \in R$, (2) $(s_F^1, s_F^2) \in R$, (3) for any $(s_1, a, s_1') \in \delta^1$ and $s_2 \in S^2$ such that $(s_1, s_2) \in R$, there is an $s_2' \in S^2$ such that $(s_1', s_2') \in R$ and $(s_2, a, s_2') \in \delta^2$, and (4) for any $(s_2, a, s_2') \in \delta^2$ and $s_1 \in S^1$ such that $(s_1, s_2) \in R$, there is an $s_1' \in S^1$ such that $(s_1', s_2') \in R$ and $(s_1, a, s_1') \in \delta^1$. It suffices to show that R is bijective.

Note that due to bismilarity, for any $(s_1, s_2) \in R$: $\{a \mid \exists_{s_1'} (s_1, a, s_1') \in \delta^1\} = \{a \mid \exists_{s_2'} (s_2, a, s_2') \in \delta^2\}$. Hence, $(s_1, s_2) \in R$ implies $en(AM_1, s_1) = en(AM_2, s_2)$. If $(s_1, s_2) \in R$ and $(s_1, s_3) \in R$, then $en(AM_1, s_1) = en(AM_2, s_2) = en(AM_2, s_3)$. Hence, $s_2 = s_3$ due to lucency. If $(s_1, s_2) \in R$ and $(s_3, s_2) \in R$, then $en(AM_1, s_1) = en(AM_1, s_3) = en(AM_2, s_2)$ and $s_1 = s_3$ due to lucency. Hence, R is functional and injective. Due to soundness all states are reachable and it is easy to see that R is also total and surjective, thus proving that AM_1 and AM_2 are isomorphic. □

Lucency is a general property and can also be formulated in the context of (labeled) Petri nets. Given a marked Petri net we would like to know whether each reachable marking has a unique "footprint" in terms of the transitions it enables. If this is the case, then the Petri net is *lucent*.

Definition 3.3. (Lucent Petri Nets)
Let (N, M) be a marked Petri net. (N, M) is *lucent* if and only if for any $M_1, M_2 \in R(N, M)$: $en(N, M_1) = en(N, M_2)$ implies $M_1 = M_2$. Given a labeling function $l \in T \rightarrow A$, labeled Petri net (N_l, M) is lucent if and only if for any $M_1, M_2 \in R(N, M)$: $en_l(N, M_1) = en_l(N, M_2)$ implies $M_1 = M_2$.

Petri net $(N_1, [p1])$ in Figure 1 is lucent. Petri net $(N_2, [p1])$ in Figure 2 is not lucent (two enabling just c). Both $(N_3, [p0])$ (without t^*) and $(\overline{N_3}, [p0])$ (the short-circuited net with t^*) in Figure 4 are lucent.

Every marked Petri net (labeled or not) defines a corresponding automaton (Definition 2.14). From the definitions, we can derive that a Petri net is lucent if and only if its corresponding automaton is lucent.

Observation 3.4. The unlabeled marked Petri net (N, M) is lucent if and only if $aut(N, M)$ is lucent. The labeled marked Petri net (N_l, M) is lucent if and only if $aut(N_l, M)$ is lucent.

It is not easy to characterize the class of lucent Petri nets. However, it is easy to show that unbounded Petri nets or Petri net with non-unique blocking markings are not lucent.

Lemma 3.5. Let (N, M) be a lucent unlabeled marked Petri net. (N, M) is bounded and each cluster has at most one blocking marking.

Proof:
Assume that (N, M) is both lucent and unbounded. We will show that this leads to a contradiction. Since (N, M) is unbounded, we can find markings M_1 and M_2 and sequences σ_0 and σ such that $(N, M)[\sigma_0\rangle(N, M_1)[\sigma\rangle(N, M_2)$ and M_2 is strictly larger than M_1. This implies that we can repeatedly execute σ getting increasingly larger markings: $(N, M_2)[\sigma\rangle(N, M_3)[\sigma\rangle(N, M_4)[\sigma\rangle(N, M_5)\ldots$. At some stage, say at M_k, the set of places that is marked stabilizes. However, the number of tokens in some places continues to increase in M_{k+1}, M_{k+2}, etc. Hence, we find markings that enable the same set of transitions but that are not the same. For example, $M_{k+1} \neq M_{k+2}$ and $en(N, M_{k+1}) = en(N, M_{k+2})$. Hence, the net cannot be lucent.

Take any cluster C and assume that (N, M) has two different reachable blocking markings M_1 and M_2. This means that $en(N, M_1) = en(N, M_2) = C \cap T$. Hence, (N, M) could not be lucent, yielding again a contradiction. □

Lemma 3.5 shows that unbounded Petri nets cannot be lucent. Moreover, it is easy to construct non-free-choice Petri nets that are not lucent. See, for example, Figure 2 where the token in $p5$ or $p6$ is invisible when $t3$ is enabled. Another example is shown in Figure 5.

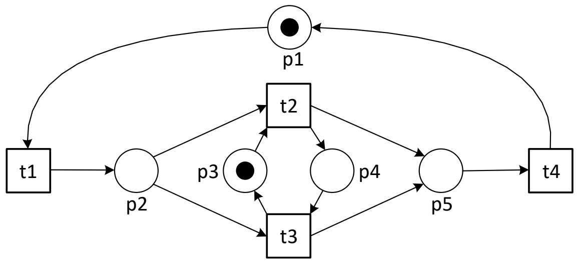


Figure 5. A non-free-choice Petri net that is not lucent. Cluster $\{p1, t1\}$ has two reachable blocking markings $M_1 = [p1, p3]$ and $M_2 = [p1, p4]$. Also cluster $\{p5, t4\}$ has two reachable blocking markings $M_3 = [p3, p5]$ and $M_4 = [p4, p5]$. Note that markings M_1 and M_2 and markings M_3 and M_4 enable the same sets of transitions.

The examples in the introduction suggest that lucency depends on the Petri net being free-choice. However, this is not the case. It is easy to construct a non-free-choice Petri net that is lucent (e.g., add unique self-loop transitions to make places visible). There are also free-choice Petri nets that are not lucent. When multiple transitions have the same label, different markings may enable the same set of activities. However, even when the Petri net is free-choice and unlabeled (i.e., the transitions are distinguishable), it may still no be lucent. Figure 6 shows such a non-lucent free-choice net.

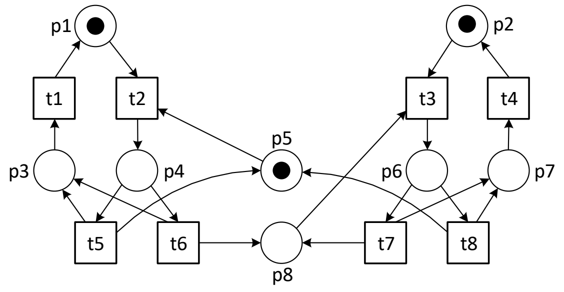


Figure 6. A live and safe free-choice net that is not lucent because reachable markings $[p3, p7, p8]$ and $[p3, p5, p7]$ both enable $t1$ and $t4$.

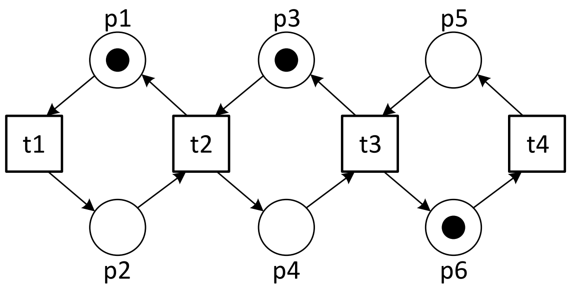


Figure 7. A live and safe marked graph that is not perpetual. Note that the net is a marked graph (all places have one input and one output arc). The model is not lucent since there are two reachable markings $M_1 = [p1, p3, p6]$ and $M_2 = [p1, p4, p6]$ that both enable $t1$ and $t4$.

Hence, just requiring that the Petri net is free-choice is not enough. Figure 7 even shows a marked graph (i.e., all places have one input and one output arc) that is not lucent.

In [1] it was shown that perpetual marked free-choice nets are lucent. These nets are live, bounded, and have a regeneration cluster, i.e., the initial marking is a home marking and marks a single cluster. The Petri net in Figure 7 does not have such a regeneration cluster.

4. Translucent event logs

Intuitively, it seems that lucent process models are more easy to discover. When the process has many states that are different, but that enable the same set of activities, then it is not easy to learn these "hidden" states. However, even when the underlying process model is lucent, traditional event logs do not reveal this. If we remove the "enabled" column in Table 1, we cannot know that c can be followed by d (this never happens in E_1). However, when looking at events e_5 and e_6 and taking the "enabled" column into account, we know that activity c in case 1 could have been followed by activity d.

In this paper, we consider *event logs that also reveal the enabled set of activities* and connect this to the notion of lucency. Event logs where every event lists the set of enabled activities is called a *translucent event log*. This section introduces such logs. Later, we will provide a discovery algorithm and relate translucency of event logs to lucency of process models. However, we first define classical event logs and introduce some notations.

Definition 4.1. (Event Log)
$\mathcal{C}$ is the universe of case identifiers, $\mathcal{A}$ is the universe of activity names, and $\mathcal{T}$ is the universe of timestamps. An event log is a non-empty set of events E such that for any $e \in E$: $\pi_{case}(e) \in \mathcal{C}$, $\pi_{act}(e) \in \mathcal{A}$, $\pi_{time}(e) \in \mathcal{T}$. There could be additional optional attributes, e.g., $\pi_x(e)$ is the x attribute of e (e.g., cost, resource, etc.). For simplicity, we assume that events in E are totally ordered such that $e_1 < e_2$ implies $\pi_{time}(e_1) \leq \pi_{time}(e_2)$.

Tables 1 and 2 show two event logs. For example, in Table 1, $\pi_{case}(e_1) = 1$, $\pi_{act}(e_1) = a$, and $\pi_{time}(e_1) =$ 09:22.

Definition 4.2. (Notation)
Let E be an event log.

- $\pi_{case}(E) = \{\pi_{case}(e) \mid e \in E\} \subseteq \mathcal{C}$ and $\pi_{act}(E) = \{\pi_{act}(e) \mid e \in E\} \subseteq \mathcal{A}$.
- For $c \in \pi_{case}(E)$, $\sigma^{E,c} = \langle e_1, e_2, \ldots, e_n \rangle$ such that $\{e_1, e_2, \ldots, e_n\} = \{e \in E \mid \pi_{case}(e) = c\}$ and $e_1 < e_2 < \ldots < e_n$. $\sigma_i^{E,c} = e_i$ is the i-th event of case c in E and $|c| = n$ is number of events related to c.
- $\overline{E} = \{e \in E \mid \exists_{c \in \pi_{case}(E)}\ e = \sigma_{|c|}^{E,c}\}$ is the set of *end events*.
- Function $next_E \in (E \setminus \overline{E}) \rightarrow E$ gives the next event in the same case (not defined for end events). Hence, $next_E(\sigma_i^{E,c}) = \sigma_{i+1}^{E,c}$ for $c \in \pi_{case}(E)$ and $1 \leq i < |c|$.
- $\pi_x(\sigma^{E,c}) = \langle \pi_x(e_1), \pi_x(e_2), \ldots, \pi_x(e_n) \rangle$ projects $\sigma^{E,c} = \langle e_1, e_2, \ldots, e_n \rangle$ onto the corresponding attribute for a given case c, e.g., $\pi_{act}(\sigma^{E,c})$ is the sequence of activities executed for c.
- $L_{act}^E = [\pi_{act}(\sigma^{E,c}) \mid c \in \pi_{case}(E)] \in \mathcal{B}(\pi_{act}(E))$ is the simple event log of E.

Let $E_1 = \{e_1, e_2, \ldots, e_{15}\}$ be the event log shown in Table 1 (ignoring the "enabled" column). $\pi_{case}(E_1) = \{1, 2, 3\}$, $\pi_{act}(E_1) = \{a, b, c, d, e\}$, and $\overline{E_1} = \{e_6, e_{14}, e_{15}\}$. $\sigma^{E_1,1} = \langle e_1, e_2, e_5, e_6 \rangle$. $next_{E_1}(e_1) = e_2$, $next_{E_1}(e_2) = e_5$, etc. $L_{act}^{E_1} = [\langle a, b, c, e \rangle, \langle a, c, b, d, b, c, e \rangle, \langle a, c, b, e \rangle]$.

A translucent event log also provides an additional attribute $\pi_{en}(e)$ indicating the set of enabled activities when event e occurred. Many information systems provide such information implicitly when providing users with a set of possible actions. In a Workflow Management (WFM) or Business Process Management (BPM) system such information is directly available (the so-called "worklists" are based on this). We also anticipate that $\pi_{en}(e)$ can be predicted based on historic information. For now, we simply assume this information to be present. We realize that this is a strong assumption, but we will show that having this information simplifies discovery dramatically.

Definition 4.3. (Translucent Event Log)
A translucent event log is an event log E such that for any $e \in E$: $\pi_{en}(e) \subseteq \mathcal{A}$ such that $\pi_{act}(e) \in \pi_{en}(e)$. The additional attribute $\pi_{en}(e)$ denotes the set of enabled activities when e occurred. A translucent event log E is *rooted* if and only if there is an $A_{init}^E \subseteq \mathcal{A}$ such that for all $c \in \pi_{case}(E)$: $\pi_{en}(\sigma_1^{E,c}) = A_{init}^E$, i.e., all cases start with the same set of enabled activities. A translucent event log E is *complete* if for any $e_1 \in E$ and $a \in \pi_{en}(e_1)$ there exists $e_2 \in E$ with $\pi_{en}(e_1) = \pi_{en}(e_2)$ and $a = \pi_{act}(e_2)$.

In Table 1 also $\pi_{en}(e)$ is given for all $e \in E_1 = \{e_1, e_2, \ldots, e_{15}\}$, e.g., $\pi_{en}(e_2) = \{b, c\}$ (see the "enabled" column). The event logs in Tables 1 and 2 are both rooted and complete.

If a translucent event log is not rooted, it can be turned into a rooted translucent event log by adding an additional start event to each case. For each case c, add an event e_0^c such that $\pi_{case}(e_0^c) = c$, $\pi_{en}(e_0^c) = \{\blacktriangleright\}$, $\pi_{act}(e_0^c) = \blacktriangleright$, and $\pi_{time}(e_0^c) = \pi_{time}(\sigma_1^{E,c})$. As a result: $A_{init}^E = \{\blacktriangleright\}$. In the remainder, we assume that all translucent event logs are rooted.

One could go one step further and define the notion of "super-translucent event logs" that directly define the state in which the event occurs. Note that for event logs generated from lucent process models, translucent event logs are "super-translucent" because the set of enabled activities identifies a unique state. However, in real-life applications, one cannot inspect the internal state. However, many systems expose the set of enabled activities (e.g., the worklist in a workflow management system or the user interface of an interactive tool).

5. Discovering lucent process models from translucent event logs

We now provide a discovery algorithm for translucent event logs. It exploits the additional enabling information provided. After introducing the technique, we discuss its properties.

5.1. Basic discovery algorithm

If we assume that the process model is lucent and the event log is translucent, process discovery becomes trivial. The state in which an event e occurs is determined by $\pi_{en}(e)$. This immediately yields a process discovery algorithm that transforms a rooted translucent event log into an accepting automaton.

Definition 5.1. (Discovery Algorithm)
Let E be a rooted translucent event log. $disc(E) = (S, A, \delta, s_0, s_F)$ is an accepting automaton such that: $S = \{\pi_{en}(e) \mid e \in E\} \cup \{\emptyset\}$, $A = \{\pi_{act}(e) \mid e \in E\}$, $s_0 = A_{init}^E$, $s_F = \emptyset$, $\delta = \{(s_1, a, s_2) \in S \times A \times S \mid \exists_{e \in E \setminus \overline{E}}\ s_1 = \pi_{en}(e) \wedge a = \pi_{act}(e) \wedge s_2 = \pi_{en}(next_E(e))\} \cup \{(s, a, \emptyset) \in S \times A \times S \mid \exists_{e \in \overline{E}}\ s = \pi_{en}(e) \wedge a = \pi_{act}(e)\}$.

The discovery algorithm adds a final state $\emptyset$ to the set of states. All other states are linked to the sets of enabled activities of events. The last event of each case corresponds to a transition $(s, a, \emptyset) \in \delta$ where a is the activity and s is the set of enabled activities of this event. The log needs to be rooted to pick the initial state.

The algorithm is easy to implement. It is even possible to use the existing "Mine Transition System" plug-in provided in ProM (based on [17]) with the proper configuration and some preprocessing. Figure 8 shows four accepting automata generated by ProM: automata (a) and (c) were obtained using our simple discovery algorithm (Definition 5.1). Automaton (a), based on event log E_1 in Table 1, is identical to AM_1 in Figure 1. Automaton (c), based on event log E_2 in Table 2, differs from AM_2 in Figure 2 because AM_2 is not lucent. In the discovered automaton the states s_1 and s_2 in AM_2 are merged, because both enable the same set of activities $\{c\}$. We also tried to apply the "Mine Transition System" plug-in without using $\pi_{en}(e)$ (i.e., assuming that E_1 and E_2 are conventional event logs). Automata (b) and (d) show the results using a horizon of 1 (i.e., the state is determined by the last activity). The two accepting automata are able to reproduce the E_1 and E_2, but are very different from AM_1 and AM_2. Note that automaton (b) does not allow for the traces like $\langle a, b, c, d, b, c, e\rangle$ (possible in the original model), but allows for traces like $\langle a, b, d, b, d, b, d, e\rangle$ (not possible in the original model). Other configurations of the "Mine Transition System" plug-in (see [17] for the different abstractions possible) result in similar problems.

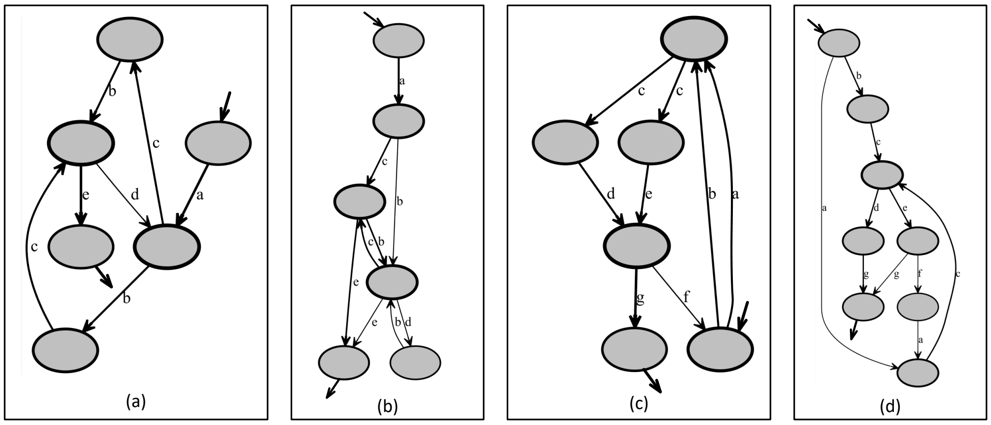


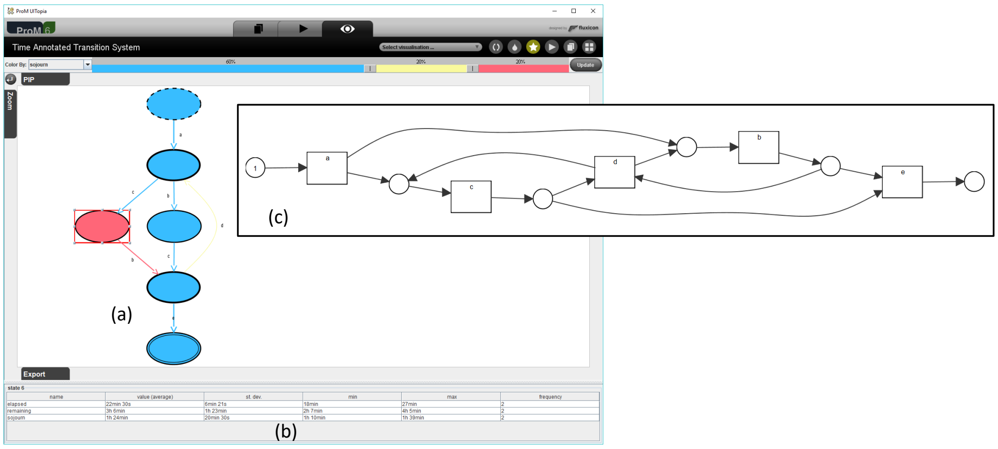
Figure 8. Four accepting automata created using ProM: automaton (a) was obtained by applying our discovery algorithm to the event log E_1 in Table 1, automaton (b) was obtained by applying the "Mine Transition System" plug-in to the same event log without using the set of enabled activities, automaton (c) was obtained by applying our discovery algorithm to the event log E_2 in Table 2, and automaton (d) was obtained without exploiting the set of enabled activities. The initial and final states are highlighted for clarity.

These examples demonstrate that the enabling information in translucent event logs is highly valuable. When the underlying process is lucent, simple algorithms can already provide powerful correctness guarantees. Since the event log can always be replayed on the models created using Definition 5.1, we can easily add frequency and performance information.

Definition 5.2. (Annotated Discovered Automaton)
Let E be a rooted translucent event log and $disc(E) = (S, A, \delta, s_0, s_F)$ the discovered automaton. $(disc(E), FS, FA)$ is an *annotated* discovered automaton adding two collections of functions $FS \subseteq S \to \mathbb{R}$ and $FS \subseteq (S \times A \times S) \to \mathbb{R}$ with $\{sfreq, stsum, stavg\} \subseteq FS$ and $\{afreq, atsum, atavg\} \subseteq FA$ such that:

- $sfreq(s) = |\{e \in E \mid s = \pi_{en}(e)\}|$ if $s \in S \setminus \{\emptyset\}$ and $sfreq(\emptyset) = |\pi_{case}(E)|$.
- $afreq((s_1, a, s_2)) = |events((s_1, a, s_2))|$ with $events((s_1, a, s_2)) = \{e \in E \setminus \overline{E} \mid s_1 = \pi_{en}(e) \ \wedge \ a = \pi_{act}(e) \ \wedge \ s_2 = \pi_{en}(next_E(e))\} \ \cup \ \{e \in \overline{E} \mid s_1 = \pi_{en}(e) \ \wedge \ a = \pi_{act}(e) \ \wedge \ s_2 = \emptyset\}$, for $(s_1, a, s_2) \in \delta$.
- $time(e) = \pi_{time}(next_E(e)) - \pi_{time}(e)$ if $e \in E \setminus \overline{E}$ and $time(e) = 0$ if $e \in \overline{E}$.
- $stsum(s) = \sum_{(s_1,a,s_2)\in\delta|s_1=s} \sum_{e\in events((s_1,a,s_2))} time(e)$, for $s \in S$.
- $atsum((s_1, a, s_2)) = \sum_{e\in events((s_1,a,s_2))} time(e)$, for $(s_1, a, s_2) \in \delta$.
- $stavg(s) = stsum(s)/sfreq(s)$, for $s \in S$.
- $atavg((s_1, a, s_2)) = atsum((s_1, a, s_2))/afreq((s_1, a, s_2))$, for $(s_1, a, s_2) \in \delta$.

As Definition 5.2 shows, we can compute frequencies and durations for both states and transitions. $sfreq(s)$ is the number of times a state s was visited and $afreq((s_1, a, s_2))$ is the number of times activity a occurred in state s_1 leading to s_2. $stavg(s)$ is the average time spent in state s and $atavg((s_1, a, s_2))$ is the average time to transition from s_1 to s_2 by performing a. We can use existing plug-ins provided in ProM to show such replay results. For example, Figure 9 shows the "Analyze Transition System" plug-in providing detailed statistics for states and transitions using the model our discovery algorithm produced for the event log in Table 1.



state 6

name	value (average)	st. dev.	min	max	frequency
elapsed	22min 30s	6min 21s	18min	27min	2
remaining	3h 6min	1h 23min	2h 7min	4h 5min	2
sojourn	1h 24min	20min 30s	1h 10min	1h 39min	2

Figure 9. The accepting automaton can be used for subsequent analysis in ProM, e.g., (a) coloring states and transitions based on sojourn times ($stavg(s)$) and delays ($atavg((s_1, a, s_2))$), (b) providing detailed replay statistics, and (c) converting the accepting automaton into an accepting Petri net using state-based regions [17, 27, 22].

5.2. Properties

The discovery algorithm described in Definition 5.1 provides several guarantees. For example, the process model can reproduce the event log used to learn it.

Theorem 5.3. (Discovered Model Can Reproduce Event Log)
Let E be a rooted translucent event log and $disc(E)$ the corresponding accepting automaton. $disc(E)$ can reproduce the event log, i.e., $\{\sigma \in L^E_{act}\} \subseteq \{\sigma \mid (\rho, \sigma) \in \Phi(disc(E))\}$.

Proof:
Let $\sigma \in L^E_{act}$ and $\sigma' = \sigma^{E,c} = \langle e_1, e_2, \ldots, e_n \rangle$ be a corresponding sequence of events such that $\sigma = \pi_{en}(\sigma')$ (there is at least one such case c). Let $\rho = \pi_{en}(\sigma') \cdot \langle \emptyset \rangle$. (ρ, σ) provides the corresponding path in the automaton starting in s_0 and ending in s_F. Hence, $(\rho, \sigma) \in \Phi(disc(E))$. $\square$

The outgoing arcs of state s in the discovered automaton have activity labels from s ($s \subseteq A$ and is only empty when it is the final state). In case of a complete log, there is at least one arc for each $a \in s$.

Lemma 5.4. Let E be a rooted translucent event log and $disc(E) = (S, A, \delta, s_0, s_F)$ the corresponding accepting automaton. For any $s \in S$: $en(disc(E), s) = \{\pi_{act}(e) \mid e \in E \ \wedge\ \pi_{en}(e) = s\} \subseteq s$. Moreover, if E is complete, then $en(disc(E), s) = s$ for all $s \in S$.

Proof:
Clearly, $en(disc(E), \emptyset) = \emptyset$, because of the way δ is constructed ($\emptyset$ has only ingoing arcs). Hence, both statements hold if $s = \emptyset$. Assume that $s \neq \emptyset$, then there is a non-empty set of events $E_s = \{e \in E \mid \pi_{en}(e) = s\}$ responsible for the outgoing arcs $\{\pi_{act}(e) \mid e \in E_s\} \subseteq s$. If E is complete, all arcs are present. □

If the log is complete, then the discovered model is lucent and sound.

Theorem 5.5. (Discovered Model Is Lucent and Sound When Log Is Complete)
For any rooted complete translucent event log E, $disc(E)$ is a lucent and sound accepting automaton.

Proof:
By construction $AM = disc(E) = (S, A, \delta, s_0, s_F)$ is an accepting automaton.

Since E is complete, $s = en(AM, s)$ for any $s \in S$ (see Lemma 5.4). Hence, for any $s_1, s_2 \in S$: $en(AM, s_1) = en(AM, s_2)$ implies $s_1 = s_2$, i.e., $disc(E)$ is a lucent.

Next, we prove soundness. For each state $s \in S \setminus \{\emptyset\}$ there is an event $e \in E$ such that $s = \pi_{en}(e)$. Consider the corresponding trace c to which e belongs. This trace defines a path starting in s_0, visiting s and ending in $\emptyset$. Similarly, a path can be constructed for each activity $a \in A$. Hence, (1) all states are reachable, (2) all activities can occur, and (3) it is always possible to reach the final state, and (4) the final state is dead ($\emptyset$ has only incoming transitions). □

The *matching* notion will be used to relate complete paths of the model to cases in the event log.

Definition 5.6. (Matching)
Let $AM = (S, A, \delta, s_0, s_F)$ be an accepting automaton, $X \in \mathcal{B}(\Phi(AM))$ a multiset of complete paths, and E a translucent event log. E *weakly matches* X, notation $E \approx_w X$, if there exists a mapping $h \in S \to \mathcal{P}(A)$, such that $[(h(\rho), \sigma) \mid (\rho, \sigma) \in X] = [(\pi_{en}(\sigma^{E,c}) \cdot \langle \emptyset \rangle, \pi_{act}(\sigma^{E,c})) \mid c \in \pi_{case}(E)]$. For a given h, we write $E \approx_w^h X$. E *strongly matches* X, notation $E \approx X$, if $E \approx_w^h X$ with $h(s) = en(AM, s)$ for $s \in S$.

We are able to rediscover a sound lucent accepting automaton based on a translucent event log that "covers" all transitions in the automaton (i.e., there exists a transition-complete multiset of complete paths strongly-matching the event log).

Theorem 5.7. (Rediscovery of a Sound Lucent Accepting Automaton)
Let AM be a sound lucent accepting automaton and E a translucent event log such that there exists a strongly-matching transition-complete multiset of complete paths. AM and $disc(E)$ are isomorphic.

Proof:
Let X be the required strongly-matching transition-complete multiset of complete paths, i.e., $X \in \mathcal{B}(\Phi(AM))$, X is transition-complete, and $E \approx X$. $AM = (S, A, \delta, s_0, s_F)$ is the sound lucent accepting automaton that needs to be rediscovered using E. Rename the states in S such that $s \in S$ is mapped onto $s' = en(AM, s)$. The resulting automaton is $AM' = (S', A', \delta', s'_0, s'_F)$ with $S' = \{en(AM, s) \mid s \in S\}$, $A' = A$, $\delta' = \{(en(AM, s_1), a, en(AM, s_2)) \mid (s_1, a, s_2) \in \delta\}$, $s'_0 = en(AM, s_0)$, $s'_F = en(AM, s_0) = \emptyset$. Since AM is lucent, AM and AM' are isomorphic (both are identical modulo the renaming of states), because there are no two states enabling the same set of activities (the mapping $s' = en(AM, s)$ is a bijection). Let X' be the same as X modulo the renaming of states. Let $disc(E) = (S'', A'', \delta'', s''_0, s''_F)$.

E is complete because $E \approx X'$ and X' is a transition-complete multiset of complete paths. Hence, according to Lemma 5.4, $en(disc(E), s) = s$ for all $s \in S''$. It is easy to see that $S'' = S'$, $A'' = A'$, $\delta'' = \delta'$, $s''_0 = s'_0$, and $s''_F = s'_F$. Therefore, $disc(E) = AM'$ and thus isomorphic with AM. $\square$

Consider the sound lucent accepting automaton AM_1 in Figure 1 and the translucent event log E_1 in Table 1. $X = [(\langle s_0, s_1, s_2, s_4, s_F\rangle, \langle a, b, c, e\rangle), (\langle s_0, s_1, s_3, s_4, s_1, s_2, s_4, s_F\rangle, \langle a, c, b, d, b, c, e\rangle), (\langle s_0, s_1, s_3, s_4, s_F\rangle, \langle a, c, b, e\rangle)]$ is a strongly-matching transition-complete multiset of complete paths. Hence, AM_1 and $disc(E_1)$ are guaranteed to be isomorphic. Figure 1 (showing AM_1) and Figure 8(a) (showing $disc(E_1)$) illustrate that this is indeed the case.

Theorem 5.7 shows that any event log that covers all the transitions in the unknown lucent model can be used to rediscover the model (up to isomorphism). Moreover, if the model is not lucent and only a weakly matching event log is used as input, we still find a model covering all behavior in the original model.

Theorem 5.8. (Reproducing the Behavior of a Non-Lucent Automaton)
Let AM be a sound (possibly non-lucent) accepting automaton and E a translucent event log such that there exists a weakly-matching transition-complete multiset of complete paths. $disc(E)$ is guaranteed to be lucent and sound accepting automaton able to reproduce the behavior of AM, i.e., $\{\sigma \mid (\rho, \sigma) \in \Phi(AM)\} \subseteq \{\sigma \mid (\rho, \sigma) \in \Phi(disc(E))\}$.

Proof:
Compared to Theorem 5.7 there are two differences: AM may not be lucent and the event log only needs to be weakly-matching. Let X be the corresponding weakly-matching transition-complete multiset of complete paths, i.e., $X \in \mathcal{B}(\Phi(AM))$, X is transition-complete, and $E \approx_w X$. Let $h \in S \to \mathcal{P}(A)$ be the corresponding mapping, i.e., $E \approx_w^h X$. Moreover, $AM = (S, A, \delta, s_0, s_F)$ and $disc(E) = (S', A', \delta', s'_0, s'_F)$.

First, we prove that $\{\sigma \mid (\rho, \sigma) \in \Phi(AM)\} \subseteq \{\sigma \mid (\rho, \sigma) \in \Phi(disc(E))\}$ by showing that for any $(\rho, \sigma) \in \Phi(AM)$, $(h(\rho), \sigma) \in \Phi(disc(E))$. This follows from $s'_0 = h(s_0)$ (all complete paths start in s_0 and correspond to initial events having $\pi_{en}(e) = h(s_0)$), $s'_F = h(s_F) = \emptyset$ (all complete paths end in a state mapped onto $\emptyset$ due to $E \approx_w^h X$), and for any $(s_1, a, s_2) \in \delta$, also $(h(s_1), a, h(s_2)) \in \delta'$ (X is transition-complete and $E \approx_w^h X$).

Remains to prove that $disc(E)$ is a lucent and sound. E is rooted because AM has one initial state s_0 that is mapped onto $s_0' = h(s_0)$. Hence, all cases start with an event having $\pi_{en}(e) = s_0'$. E is complete because X is transition-complete, $E \approx_w^h X$, and states can only be merged (not split). Therefore, we can apply Theorem 5.5 showing that $disc(E)$ is a lucent and sound. □

Consider the sound non-lucent accepting automaton AM_2 in Fig. 2 and the translucent event log E_2 in Table 2. $X = [(\langle s_0, s_1, s_3, s_5, s_F\rangle, \langle a, c, d, g\rangle), (\langle s_0, s_2, s_4, s_5, s_0, s_1, s_3, s_5, s_F\rangle, \langle b, c, e, f, a, c, d, g\rangle), (\langle s_0, s_2, s_4, s_5, s_F\rangle, \langle b, c, e, g\rangle)]$ is a transition-complete multiset of complete paths. $E_2 \approx_w^h X$ for $h(s_0) = \{a, b\}$, $h(s_1) = \{c\}$, $h(s_2) = \{c\}$, $h(s_3) = \{d\}$, $h(s_4) = \{e\}$, $h(s_5) = \{f, g\}$, and $h(s_F) = \emptyset$. Therefore, $disc(E_2)$ should be able to reproduce the behavior of AM_2. The discovered model $disc(E_2)$ in Figure8(c) shows that this is indeed the case.

Let us now consider the scenario where $\pi_{en}(e)$ is not recorded correctly, i.e., the logging mechanism changes the set of enabled activities into a different set. We call such event logs *inexact* translucent event logs.

Definition 5.9. (Inexact Translucent Event Logs)
A translucent event log E is inexact when the $\pi_{en}(e)$ does not reflect the real set of enabled activities. For any $e \in E$: $\pi_{en}(e) \subseteq \mathcal{A}$ is the reported set of enabled activities and $\pi_{real}(e) \subseteq \mathcal{A}$ is the real set of enabled activities. Both include the activity that occurred, i.e., $\pi_{act}(e) \in \pi_{en}(e) \cap \pi_{real}(e)$ for any $e \in E$. E_{real} is the translucent event log where $\pi_{en}(e)$ is replaced by the value $\pi_{real}(e)$. Inexact translucent event log E is called *stable* if there is a function $\mathit{fuzz} \in \mathcal{P}(A) \to \mathcal{P}(A)$ such that $\pi_{en}(e) = \mathit{fuzz}(\pi_{real}(e))$ for any $e \in E$. Inexact translucent event log E is *weakly-stable* if there is a function $\mathit{fuzz} \in (\mathcal{P}(A) \times A) \to \mathcal{P}(A)$ such that $\pi_{en}(e) = \mathit{fuzz}(\pi_{real}(e), \pi_{act}(e))$ for any $e \in E$. Inexact translucent event log E *overshoots* if $\pi_{real}(e) \subseteq \pi_{en}(e)$ for all $e \in E$. Inexact translucent event log E *undershoots* if $\pi_{en}(e) \subseteq \pi_{real}(e)$ for all $e \in E$.

When an inexact translucent event log is unstable (i.e., $\pi_{en}(e)$ may have a random value unrelated to the actual state of the system), then the additional enabling information is useless. However, the knowledge that the event log is (weakly) stable, overshooting, or undershooting, may be exploited during discovery.

It is interesting to consider different types of weakly-stable inexact translucent event logs characterized by a function $\mathit{fuzz} \in (\mathcal{P}(A) \times A) \to \mathcal{P}(A)$. For example, $\mathit{fuzz}(s, a) = \{a\}$ (state is fully determined by the next activity, i.e., undershooting) and $\mathit{fuzz}(s, a) = A$ (resulting in a "flower model" allowing for any behavior involving the activities A, i.e., overshooting). It is possible to formulate additional properties depending on the type of inexactness (undershooting, overshooting, etc.) and the level of completeness. Since this is beyond the scope of this paper, we only provide the following corollary and show that in general weak stability is not enough.

Corollary 5.10. (Handling Inexact Stable Translucent Event Logs)
Let AM be a sound accepting automaton and E an inexact stable translucent event log such that for E_{real} there exists a weakly-matching transition-complete multiset of complete paths. $disc(E)$ is guaranteed to be a lucent and sound accepting automaton able to reproduce the behavior of AM.

Proof:
We need to show that $\{\sigma \mid (\rho, \sigma) \in \Phi(AM)\} \subseteq \{\sigma \mid (\rho, \sigma) \in \Phi(disc(E))\}$. Let X be the corresponding weakly-matching transition-complete multiset of complete paths for E_{real}. Let $h \in S \rightarrow \mathcal{P}(A)$ be such that $E_{real} \approx_w^h X$. Since E is stable, there is a function *fuzz* such that $\pi_{en}(e) = \mathit{fuzz}(\pi_{real}(e))$ for any $e \in E$. We can compose h and *fuzz* into a function g such that $g(s) = \mathit{fuzz}(h(s))$. It is easy to verify that $E \approx_w^g X$. Hence, we can apply Theorem 5.8. □

Corollary 5.10 does not hold for *weakly-stable* logs due to the fact that *fuzz* may be diverging, i.e., there may be a state s and activities a and b such that $\mathit{fuzz}(s, a) \neq \mathit{fuzz}(s, b)$. Consider, for example AM_1 in Figure 1 and E_1 in Table 1, in conjunction with the function $\mathit{fuzz}(s, x) = \{x\}$. The resulting discovered model does not allow activity c to followed by activity d. This problem can only be addressed by using a stronger completeness notation than transition-completeness (e.g., completeness with respect to two subsequent transitions).

6. Discovering Petri nets

The basic algorithm described in Definition 5.1 produces an accepting automaton and not an accepting Petri net. However, whenever activities do not happen in a fixed other, automata tend to be too complex or underfitting. Therefore, we discuss the need for discovering Petri nets (either directly or indirectly).

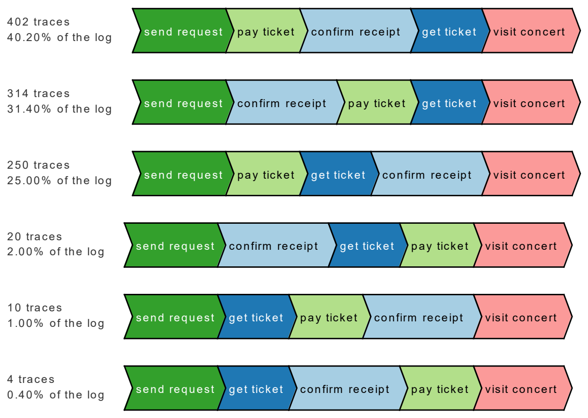
6.1. Concurrency matters

Consider the event log shown in Figure 10(a). The event log contains information about 1000 cases and each case has 5 events. Figure 10(b) shows the so-called directly follows graph produced by Disco, i.e., the process mining tool from Fluxicon (`www.fluxicon.com`). Figure 10(c) shows the same model discovered by Celonis (`www.celonis.com`). The directly follows graphs produced by these and many other commercial systems can be very misleading. The state of the process is determined by the last activity. As a result, concurrent activities automatically result in loops. Note that in the event log each activity occurs precisely once for each case. However, the directly follows graphs in Figure 10 do no capture this and allow for behaviors very different from what was discovered.

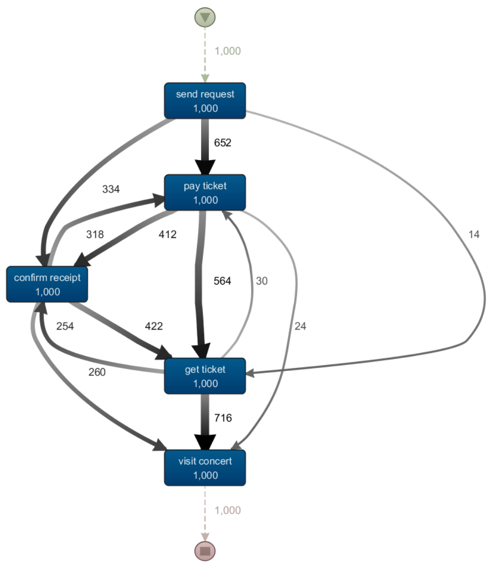
Process discovery techniques that discover Petri nets from event logs will discover concurrency and do not introduce loops. Figure 11(a) shows the model that will be discovered by techniques like the inductive mining approaches [20, 21], the α-algorithm [18], and the ILP miner [23]. Clearly, such models are desirable.

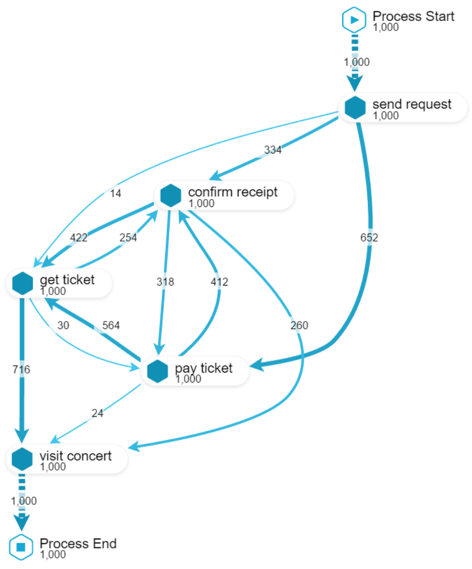
Although the Petri net in Figure 11(a) can be discovered using classical event logs, let us assume that we have a translucent log. $E = [\langle \underline{s}, \underline{p}cg, \underline{c}g, \underline{g}, \underline{v}\rangle^{402}, \langle \underline{s}, \underline{c}pg, \underline{p}g, \underline{g}, \underline{v}\rangle^{314}, \langle \underline{s}, \underline{p}gc, \underline{g}c, \underline{c}, \underline{v}\rangle^{250}, \langle \underline{s}, \underline{c}gp, \underline{g}p, \underline{p}, \underline{v}\rangle^{20}, \langle \underline{s}, \underline{g}pc, \underline{p}c, \underline{c}, \underline{v}\rangle^{10}, \langle \underline{s}, \underline{g}cp, \underline{c}p, \underline{p}, \underline{v}\rangle^{4}]$ describes the translucent event log using the shorthand notation introduced in Section 1.2. Figure 11(b) shows the automaton $AM = disc(E)$ using the discovery algorithm from Definition 5.1. Clearly, the discovered automaton captures the process correctly. However, since it shows all interleavings explicitly, the model is not as compact and less readable.

As mentioned, the Petri net in Figure 11(a) could have been discovered from a non-translucent event log. However, as shown before the additional enabling information can be exploited.

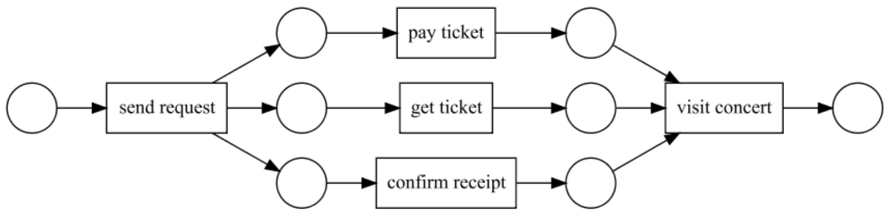


(a) event log with 1000 cases and 5000 events





(b) process model discovered by Disco (c) process model discovered by Celonis



(d) process model discovered by ProM

Figure 10. An event log and the models discovered by Disco, Celonis, and ProM.

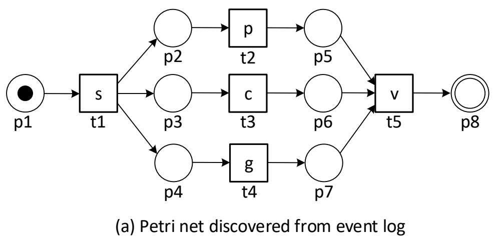

p1
s
t1
p2
p3
p4
p
t2
c
t3
g
t4
p5
p6
p7
v
t5
p8
(a) Petri net discovered from event log
s
pcg
cg
pg
pc
g
p
c
v
(b) automaton discovered from event log

Figure 11. Process models discovered from the event log $E = [\langle \underline{s}, \underline{p}cg, \underline{c}g, \underline{g}, \underline{v}\rangle^{402}, \langle \underline{s}, \underline{c}pg, \underline{p}g, \underline{g}, \underline{v}\rangle^{314}, \langle \underline{s}, \underline{p}gc, \underline{g}c, \underline{c}, \underline{v}\rangle^{250}, \langle \underline{s}, \underline{c}gp, \underline{g}p, \underline{p}, \underline{v}\rangle^{20}, \langle \underline{s}, \underline{g}pc, \underline{p}c, \underline{c}, \underline{v}\rangle^{10}, \langle \underline{s}, \underline{g}cp, \underline{c}p, \underline{p}, \underline{v}\rangle^{4}]$.

6.2. Indirect techniques

A straightforward approach to obtain a Petri net from the automaton $AM = disc(E)$ discovered using Definition 5.1 is to use *region theory* [32]. *State-based regions* were introduced by Ehrenfeucht and Rozenberg in 1989 [26] and generalized by Cortadella et al. [27] and Badouel, Bernardinello, and Darondeau [32]. In [17] it was shown how an event log can be converted into different automata using a range of abstractions (Step 1) and how these automata can be converted into Petri nets to uncover concurrency (Step 2). The work presented in this paper can be used to refine the two-step approach in [17].

As demonstrated in [28, 27], after some preprocessing, any automaton can be converted into a labeled Petri net that is bisimilar. This may require label splitting when the automaton does not satisfy the basic synthesis conditions [28, 27]. Figure 9 already showed the conversion of the automaton discovered using Definition 5.1 into a bisimilar Petri net using ProM.

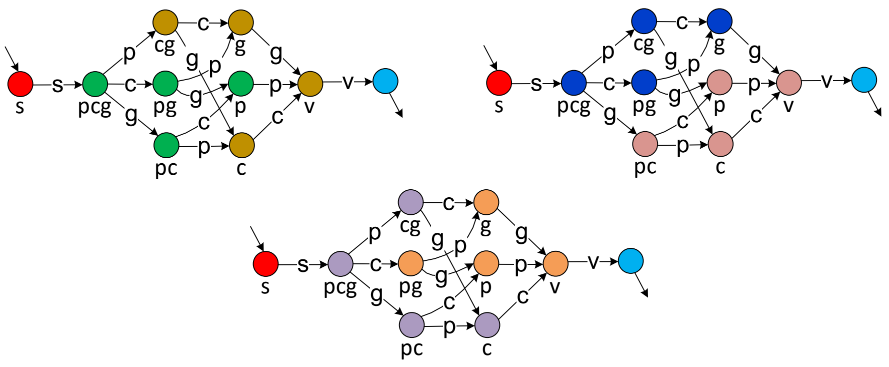


Figure 12. The six minimal regions discovered from the automaton in Figure 11(b). These regions correspond to the places of the Petri net in Figure 11(a).

Figure 12 shows the regions in the automaton discovered for the translucent log $E = [\langle \underline{s}, \underline{p}cg, \underline{c}g, \underline{g}, \underline{v}\rangle^{402}, \langle \underline{s}, \underline{c}pg, \underline{p}g, \underline{g}, \underline{v}\rangle^{314}, \langle \underline{s}, \underline{p}gc, \underline{g}c, \underline{c}, \underline{v}\rangle^{250}, \langle \underline{s}, \underline{c}gp, \underline{g}p, \underline{p}, \underline{v}\rangle^{20}, \langle \underline{s}, \underline{g}pc, \underline{p}c, \underline{c}, \underline{v}\rangle^{10}, \langle \underline{s}, \underline{g}cp, \underline{c}p, \underline{p}, \underline{v}\rangle^{4}]$.

Recall that Theorem 3.2 showed that any two bisimilar sound lucent accepting automata are isomorphic. This makes lucency an interesting property in the context of net synthesis (e.g., its relation to state separation and forward closure).

6.3. Direct techniques

Next to the indirect techniques using the discovery algorithm from Definition 5.1, we also envision a range of *new discovery techniques* directly using translucent event logs to uncover concurrency. It is fairly straightforward to improve existing discovery algorithms using the additional enabling information in translucent event logs. To illustrate this consider the α-algorithm [18] which is based on learning relations such as $a \rightarrow b$ (causality), $a \mid\mid b$ (concurrency), and $a\#b$ (choice). In [18] the algorithm was shown to be correct for a subclass of lucent Petri nets. In [1] it was shown that all perpetual marked free-choice nets are lucent. The class of nets for which the α-algorithm was proven to be correct (Theorem 4.10 in [18]) is actually subclass of perpetual marked free-choice nets. Using the enabling information attached to events it is possible to improve the quality of the relations ($a \rightarrow b$, $a \mid\mid b$, and $a\#b$) and add new ones. Consider, for example, the event log $[\langle \underline{a}, \underline{b}c, \underline{c}, d\underline{e} \rangle, \langle \underline{a}, b\underline{c}, \underline{b}, \underline{d}e, \underline{b}c, \underline{c}, d\underline{e} \rangle, \langle \underline{a}, b\underline{c}, \underline{b}, d\underline{e} \rangle]$ using the shorthand notation used in Section 1.2. Although c was never directly followed by d the pattern $\langle \ldots \underline{c}, d\underline{e} \ldots \rangle$ shows that this is possible. The pattern $\langle \ldots \underline{b}c, \underline{c} \ldots \rangle$ shows that c was not disabled by doing b, i.e., both are concurrent. The α-algorithm cannot distinguish between concurrency and the alternation of two activities a and b (e.g., loops of length 2). However, by using enabling information we can distinguish between $a \rightarrow b \rightarrow a$ and $a \mid\mid b$. These simple examples show how the α-algorithm can exploit translucency.

Another opportunity to employ enabling information is provided by the *inductive mining* approaches [20, 21]. These recursively partition the set of activities and logs based on finding so-called "cuts" in directly-follows graphs. Using enabling information one can find better cuts, even when the event log is incomplete. Also, discovery techniques using *language-based regions* can be easily extended to exploit translucency [19, 23]. For example, in the ILP miner [23] one can add an additional inequality per activity in the enabling set of an event. This will help to avoid overfitting models (a common problem when applying regions).

In case the event log is not translucent, the assumption of lucency still provides a new angle on process discovery, putting more emphasis on finding states. To illustrate this consider a conventional event log where each event in the event log is characterized by $e = (\sigma_{pref}, a, \sigma_{post})$ where σ_{pref} is the prefix (activities that happened before e), a is the activity executed, and σ_{post} is the postfix (activities that happened after e). The result of applying a process discovery algorithm can be seen as a function $state()$ which maps any event e onto a state $state(e)$, i.e., the state in which e occurred (see [24, 17] for explanations). Hence, events e_1 and e_2 satisfying $state(e_1) = state(e_2)$ occurred in the same state and can be viewed as "equivalent". This way discovery is reduced to finding an *equivalence relation* on the set of events in the log. Given such an equivalence relation one can use a variant of the discovery algorithm described in Definition 5.1. Viewing process discovery as "finding an equivalence relation on events" provides an original angle on this challenging and highly relevant learning task.

7. Conclusion and implications

Lucency of process models and translucency of event logs were introduced as two new notions related to explicit “enabling information”. In a lucent process model, there cannot be two states enabling the same set of activities. In a translucent event log, each event carries information about the set of activities enabled when it occurred. This paper explored these notions and related them.

We provided a novel discovery algorithm for translucent event logs. Using the additional enabling information, our algorithm can easily outperform traditional algorithms, especially when models are large and event logs are incomplete (i.e., only a small fraction of the set of possible traces is observed). However, the main point is not to present a particular algorithm. Instead, we want to stress the value of additional enabling information. In many information systems, the system’s interface reveals the set of possible activities. Consider, for example, the worklists provided by WFM/BPM systems or the buttons on a user interface. Hence, it is not unrealistic to obtain translucent event logs. Even when event logs are semi-translucent (i.e., the reported set of enabled activities is too large or too small), the information may be valuable. Moreover, a range of heuristics is possible to enhance enabling information and add it to events.

The discovery algorithm for translucent event logs can also be extended with frequency-based filtering (this is fairly straightforward, cf. Definition 5.2). As described in Section 6, we also envision a range of discovery techniques based on existing approaches while exploiting translucency.

As demonstrated, lucency and translucency strengthen each other. Given a rooted complete translucent event log, our discovery algorithm returns a lucent process model able to reproduce the event log (Theorem 5.3 and Theorem 5.5). It is also relatively easy to rediscover lucent process models using translucent event logs (Theorem 5.7). Moreover, the ability to link the enabling of activities to states (i.e., lucency) is also useful when dealing with conventional event logs. Process mining studies the more general relationship between modeled behavior and observed behavior [16]. Having process models where multiple states enable the same set of activities complicates most of the process mining tasks. For example, Petri nets where different transitions have the same activity label are notoriously difficult to reconstruct based on event data. In general, the search space can be reduced considerably by assuming lucent process models. Hence, these novel insights may lead to new process mining algorithms or help to prove the correctness and/or guarantees of existing algorithms.

Acknowledgements

We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

[1] van der Aalst W. Markings in Perpetual Free-Choice Nets Are Fully Characterized by Their Enabled Transitions. In: Khomenko V, Roux O (eds.), Applications and Theory of Petri Nets 2018, volume 10877 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2018 pp. 315–336. doi:10.1007/978-3-319-91268-4_16.

[2] van der Aalst W, Stahl C. Modeling Business Processes: A Petri Net Oriented Approach. MIT Press, Cambridge, MA, 2011. URL `https://www.jstor.org/stable/j.ctt5vjqff`.

[3] Murata T. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 1989. **77**(4):541–580. doi:10.1109/5.24143.

[4] Reisig W. Petri Nets: Modeling Techniques, Analysis, Methods, Case Studies. Springer-Verlag, Berlin, 2013. doi:10.1007/978-3-642-33278-4.

[5] Reisig W, Rozenberg G (eds.). Lectures on Petri Nets I: Basic Models, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998. doi:10.1007/3-540-65306-6.

[6] Reisig W, Rozenberg G (eds.). Lectures on Petri Nets II: Applications, volume 1492 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998. doi:10.1007/3-540-65307-4.

[7] Best E. Structure Theory of Petri Nets: the Free Choice Hiatus. In: Brauer W, Reisig W, Rozenberg G (eds.), Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties, volume 254 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1987 pp. 168–206. doi:10.1007/978-3-540-47919-2_8.

[8] Best E, Wimmel H. Structure Theory of Petri Nets. In: Jensen K, van der Aalst W, Balbo G, Koutny M, Wolf K (eds.), Transactions on Petri Nets and Other Models of Concurrency (ToPNoC VII), volume 7480 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2013 pp. 162–224. doi:10.1007/978-3-642-38143-0_5.

[9] Desel J, Esparza J. Free Choice Petri Nets, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995. ISBN:0-521-46519-2.

[10] Best E, Desel J, Esparza J. Traps Characterize Home States in Free-Choice Systems. *Theoretical Computer Science*, 1992. **101**:161–176. doi:10.1016/0304-3975(92)90048-K.

[11] Esparza J. Reachability in Live and Safe Free-Choice Petri Nets is NP-Complete. *Theoretical Computer Science*, 1998. **198**(1-2):211–224. URL `https://doi.org/10.1016/S0304-3975(97)00235-1`.

[12] Thiagarajan P, Voss K. A fresh look at free choice nets. *Information and Control*, 1984. **61**(2):85–113. URL `https://doi.org/10.1016/S0019-9958(84)80052-2`.

[13] Wehler J. Free-Choice Petri Nets without Frozen Tokens, and Bipolar Synchronization Systems. *Fundamenta Informaticae*, 2010. **98**(2-3):283–320. doi:10.3233/FI-2010-228.

[14] Gaujal B, Haar S, Mairesse J. Blocking a Transition in a Free Choice Net and What it Tells About its Throughput. *Journal of Computer and System Science*, 2003. **66**(3):515–548. URL `https://doi.org/10.1016/S0022-0000(03)00039-4`.

[15] Wehler J. Simplified Proof of the Blocking Theorem for Free-Choice Petri Nets. *Journal of Computer and System Science*, 2010. **76**(7):532–537. URL `https://doi.org/10.1016/j.jcss.2009.10.001`.

[16] van der Aalst W. Process Mining: Data Science in Action. Springer-Verlag, Berlin, 2016. doi:10.1007/978-3-662-49851-4.

[17] van der Aalst W, Rubin V, Verbeek H, van Dongen B, Kindler E, Günther C. Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling*, 2010. **9**(1):87–111. doi:10.1007/s10270-008-0106-z.

[18] van der Aalst W, Weijters A, Maruster L. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 2004. **16**(9):1128–1142. doi:10.1109/TKDE.2004.47.

[19] Bergenthum R, Desel J, Lorenz R, Mauser S. Process Mining Based on Regions of Languages. In: Alonso G, Dadam P, Rosemann M (eds.), International Conference on Business Process Management (BPM 2007), volume 4714 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2007 pp. 375–383. doi:10.1007/978-3-540-75183-0_27.

[20] Leemans S, Fahland D, van der Aalst W. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In: Lohmann N, Song M, Wohed P (eds.), Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2013), volume 171 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2014 pp. 66–78. doi:10.1007/978-3-319-06257-0_6.

[21] Leemans S, Fahland D, van der Aalst W. Scalable Process Discovery and Conformance Checking. *Software and Systems Modeling*, 2018. **17**(2):599–631. doi:10.1007/s10270-016-0545-x.

[22] Solé M, Carmona J. Process Mining from a Basis of State Regions. In: Applications and Theory of Petri Nets (Petri Nets 2010), volume 6128 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2010 pp. 226–245. doi:10.1007/978-3-642-13675-7_14.

[23] van der Werf J, van Dongen B, Hurkens C, Serebrenik A. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 2010. **94**:387–412. doi:10.3233/FI-2009-136.

[24] van der Aalst W, Adriansyah A, van Dongen B. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery*, 2012. **2**(2):182–192. URL `https://doi.org/10.1002/widm.1045`.

[25] Carmona J, van Dongen B, Solti A, Weidlich M. Conformance Checking: Relating Processes and Models. Springer-Verlag, Berlin, 2018. doi:10.1007/978-3-319-99414-7.

[26] Ehrenfeucht A, Rozenberg G. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 1989. **27**(4):315–368.

[27] Cortadella J, Kishinevsky M, Lavagno L, Yakovlev A. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 1998. **47**(8):859–882. doi:10.1109/12.707587.

[28] Carmona J. The Label Splitting Problem. In: Jensen K, van der Aalst W, Marsan MA, Franceschinis G, Kleijn J, Kristensen L (eds.), Transactions on Petri Nets and Other Models of Concurrency (ToPNoC VI), volume 7400 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2012 pp. 1–23. doi:10.1007/978-3-642-35179-2_1.

[29] van der Aalst W. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 1998. **8**(1):21–66. URL `https://doi.org/10.1142/S0218126698000043`.

[30] van der Aalst W, van Hee K, ter Hofstede A, Sidorova N, Verbeek H, Voorhoeve M, Wynn M. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing*, 2011. **23**(3):333–363. doi:10.1007/s00165-010-0161-4.

[31] Genrich HJ, Thiagarajan PS. A Theory of Bipolar Synchronization Schemes. *Theoretical Computer Science*, 1984. **30**(3):241–318. URL `https://doi.org/10.1016/0304-3975(84)90137-3¿`

[32] Badouel E, Bernardinello L, Darondeau P. Petri Net Synthesis. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2015. doi:10.1007/978-3-662-47967-4.