Workshop Proceedings

Workshop on

Algorithms & Theories for the Analysis of Event Data (ATAED'2019)

Aachen, Germany, June 25, 2019

Satellite event of the conferences

19th International Conference on Application of Concurrency to System Design (ACSD 2019)

40th International Conference on Application and Theory of Petri Nets and Concurrency (PN 2019)

Edited by
Wil van der Aalst, Robin Bergenthum, and Josep Carmona

.

Preface

Ehrenfeucht and Rozenberg defined regions more than 25 years ago as sets of nodes of a finite transition system. Every region relates to potential conditions that enable or disable transition occurrences in an associated elementary net system. Later, similar concepts were used to define regions for Petri nets from languages as well. Both *state-based* and *language-based* approaches aim to constrain a Petri net by adding places deduced from the set of *regions*. By now, many variations have been proposed, e.g., approaches dealing with multiple tokens in a place, region definitions for Petri nets with inhibitor arcs, extensions to partial languages, regions for infinite languages, etc.

Initially, region theory focused on *synthesis*. We require the input and the behavior of the resulting Petri net to be equivalent. Recently, region-based research started to focus on *process mining* as well where the goal is *not* to create an equivalent model but to *infer* new knowledge from the input. Process mining examines observed behavior rather than assuming a complete description in terms of a transition system or prefix-closed language. For this reason, one needs to deal with new problems such as noise and incompleteness. Equivalence notions are replaced by trade-offs between fitness, simplicity, precision, and generalization. A model with good *fitness* allows for most of the behavior seen in the event log. A model that does not *generalize* is "overfitting". Overfitting is the problem that a very specific model is generated whereas it is obvious that the log only holds example behavior. A model that allows for "too much behavior" lacks *precision*. Simplicity is related to Occam's Razor which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything". Following this principle, we look for the *simplest* process model that can explain what was observed in the event log. Process discovery from event logs is very challenging because of these and many other trade-offs. Clearly, there are many theoretical process-mining challenges with a high practical relevance that need to be addressed urgently.

All these challenges and opportunities are the motivation for organizing the *Algorithms & Theories for the Analysis of Event Data* (ATAED) workshop. The workshop first took place in Brussels in 2015 as a succession of the *Applications of Region Theory* (ART) workshop series. From there on, the workshop moved to Toruń (2016), Zaragoza (2017) and Bratislava (2018). After the success of these workshops, it is only natural to bring together researchers working on region-based synthesis and process mining again.

The ATAED'2019 workshop took place in Aachen on June 25, 2019 and was a satellite event of both the 40th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2019) and the 19th International Conference on Application of Concurrency to System Design (ACSD 2019). This year, the workshop is also co-located with the 1st International Conference on Process Mining (ICPM 2019).

Papers related to process mining, region theory and other synthesis techniques were presented at ATAED'2019. These techniques have in common that "lower level" behavioral descriptions (event logs, partial languages, transition sys-

tems, etc.) are used to create "higher level" process models (e.g., various classes of Petri nets, BPMN, or UML activity diagrams). In fact, all techniques that aim at learning or checking concurrent behavior from transition systems, runs, or event logs were welcomed. The workshop was supported by the IEEE Task Force on Process Mining (www.win.tue.nl/ieeetfpm/).

After a careful reviewing process, six papers were accepted for the workshop. We thank the reviewers for providing the authors with valuable and constructive feedback. Moreover, we were honored that *Wolfgang Reisig* was willing to give an invited talk on *"How to analyze BIG systems?"*. We thank Wolfgang, the authors, and the presenters for their wonderful contributions.

Enjoy reading the proceedings!

Wil van der Aalst, Robin Bergenthum, and Josep Carmona
June 2019

Program committee of ATAED'2019

Wil van der Aalst, RWTH Aachen, Germany (co-chair)
Abel Armas Cervantes, QUT, Australia
Eric Badouel, INRIA Rennes, France
Robin Bergenthum, FernUni Hagen, Germany (co-chair)
Luca Bernardinello, Universitá degli studi di Milano-Bicocca, Italy
Andrea Burattin, University of Innsbruck, Austria
Josep Carmona, UPC Barcelona, Spain (co-chair)
Paolo Ceravolo, University of Milan, Italy
Claudio Di Ciccio, Vienna University of Economics and Business, Austria
Benoît Depaire, Hasselt University, Belgium
Jörg Desel, FernUni Hagen, Germany
Dirk Fahland, TU Eindhoven, The Netherlands
Chiara Di Francescomarino, FBK-IRST, Italy
Stefan Haar, LSV CNRS & ENS de Cachan, France
Gabriel Juhás, Slovak University of Technology, Slovak Republic
Anna Kalenkova, Higher School of Economics NRU, Russia
Jetty Kleijn, Leiden University, The Netherlands
Robert Lorenz, Uni Augsburg, Germany
Manuel Mucientes, University of Santiago de Compostela, Spain
Marta Pietkiewicz-Koutny, Newcastle University, GB
Uli Schlachter, Uni Oldenburg, Germany
Arik Senderovich, Technion, Israel
Jochen De Weerdt, KU Leuven, Belgium
Lijie Wen, Tsinghua University, China
Moe Wynn, Queensland University of Technology, Australia
Alex Yakovlev, Newcastle University, GB

Table of Contents

Synthesis of Weighted Marked Graphs from Circular Labelled Transition Systems

Raymond Devillers[1], Evgeny Erofeev(✉)*[2], and Thomas Hujsa**[3]

[1] Département d'Informatique, Université Libre de Bruxelles,
B-1050 Brussels, Belgium (rdevil@ulb.ac.be)
[2] Department of Computing Science, Carl von Ossietzky Universität Oldenburg,
D-26111 Oldenburg, Germany (evgeny.erofeev@informatik.uni-oldenburg.de)
[3] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France (thujsa@laas.fr)

Abstract. Several works have proposed methods for the analysis and synthesis of Petri net subclasses from labelled transition systems (LTS). In this paper, we focus on Choice-Free (CF) Petri nets, in which each place has at most one output, and their subclass of Weighted Marked Graphs (WMGs). We provide new conditions for the WMG-synthesis from a circular LTS, i.e. forming a single circuit, and discuss the difficulties in extending these new results to the CF case.

Keywords: Weighted Petri net, choice-free net, synthesis, labelled transition system, cycles, cyclic words, circular solvability.

1 Introduction

Petri nets form a highly expressive and intuitive operational model of discrete event systems, capturing the mechanisms of synchronisation, conflict and concurrency. Many of their fundamental behavioural properties are decidable, allowing to model and analyse numerous artificial and natural systems. However, most interesting model checking problems are worst-case intractable, and the efficiency of synthesis algorithms varies widely depending on the constraints imposed on the desired solution. In this study, we focus on the Petri net synthesis problem from a labelled transition system (LTS), which consists in determining the existence of a Petri net whose reachability graph is isomorphic to the given LTS, and building such a Petri net solution when it exists.

In previous studies on analysis or synthesis, structural restrictions on nets encompassed *plain* nets (each weight equals 1; also called ordinary nets) [25], *homogeneous* nets (i.e. for each place p, all the output weights of p are equal) [28, 23], *free-choice* nets (the net is plain, hence also homogeneous, and any two

* Supported by DFG through grant Be 1267/16-1 ASYST.
** Supported by the STAE foundation/project DAEDALUS, Toulouse, France.

transitions sharing an input place have the same set of input places) [12, 28], join-free nets (each transition has at most one input place) [28, 11, 22, 23], etc.

More recently, another kind of restriction has been considered, limiting the number of different transition labels of the LTS [2, 3, 18, 19].

In this paper, we study the problem of solvability of LTS with weighted marked graphs (each place has at most one output transition and one input transition) and choice-free nets (each place has at most one output transition). Both these classes are important for real-world applications, and they are widely studied in the literature [27, 21, 15, 9, 26, 8, 16, 7]. In this work, we focus mainly on finite *circular LTS*, meaning strongly connected LTS that contain a unique *cycle*[4]. In this context, we investigate the *cyclic solvability* of a word w, meaning the existence of a Petri net solution to the finite circular LTS induced by the infinite *cyclic word* w^∞.

An important purpose of studying such constrained LTS is to better understand the relationship between LTS decompositions and their solvability by Petri nets. Indeed, the unsolvability of simple subgraphs of the given LTS, typically elementary paths (i.e. not containing any node twice) and cycles (i.e. closed paths, whose start and end states are equal), often induces simple conditions of unsolvability for the entire LTS, as highlighted in other works [2, 18, 4]. Moreover, cycles appear systematically in the reachability graph of live and/or reversible Petri nets [27], which are used to model various real-world applications, such as embedded systems [20].

Contributions. In this work, we study further the links between simple LTS structures and the reachability graph of WMGs and CF nets, as follows. First, we show that a binary LTS is CF-solvable if and only if it is WMG-solvable. Then, we provide new general conditions for the WMG-solvability of a cyclic word over an arbitrary alphabet, together with an algorithm synthesizing a cyclical WMG-solution for a given word. We also discuss the difficulties of extending these results to the CF class.

Organisation of the paper. After recalling classical definitions, notations and properties in Section 2, we present the equivalence of CF- and WMG-solvability for 2-letter words in Section 3. Then, in Section 4, we focus on circular LTS: we develop a new characterisation of WMG-solvability and a dedicated synthesis algorithm. We also provide a number of examples, which demonstrate that some of the presented results cannot be applied to the class of CF-nets. Finally, Section 5 presents our conclusions and perspectives.

[4] A set A of k arcs in a LTS G defines a cycle of G if the elements of A can be ordered as a sequence $a_1 \dots a_k$ such that, for each $i \in \{1, \dots, k\}$, $a_i = (n_i, \ell_i, n_{i+1})$ and $n_{k+1} = n_1$, i.e. the i-th arc a_i goes from node n_i to node n_{i+1} until the first node n_1 is reached, closing the path.

2 Classical Definitions, Notations and Properties

LTS, sequences and reachability. A *labelled transition system with initial state*, *LTS* for short, is a quadruple $TS = (S, \rightarrow, T, \iota)$ where S is the set of *states*, T is the set of *labels*, $\rightarrow \subseteq (S \times T \times S)$ is the *transition relation*, and $\iota \in S$ is the *initial state*. A label t is *enabled* at $s \in S$, written $s[t\rangle$, if $\exists s' \in S\colon (s,t,s') \in \rightarrow$, in which case s' is said to be *reachable* from s by the firing of t, and we write $s[t\rangle s'$. Generalising to any (firing) sequences $\sigma \in T^*$, $s[\varepsilon\rangle$ and $s[\varepsilon\rangle s$ are always true, with ε being an empty sequence; and $s[\sigma t\rangle s'$, i.e., σt is *enabled* from state s and leads to s' if there is some s'' with $s[\sigma\rangle s''$ and $s''[t\rangle s'$. For clarity, in case of long formulas we write $|_r\sigma|_s\tau|_q$ instead of $r[\sigma\rangle s[\tau\rangle q$, thus fixing some intermediate states along a firing sequence. A state s' is *reachable* from state s if $\exists \sigma \in T^*\colon s[\sigma\rangle s'$. The set of states reachable from s is noted $[s\rangle$. $TS = (S, \rightarrow, T, \iota)$ is *fully reachable* if $S = [\iota\rangle$.

Petri nets and reachability graphs. A (finite, place-transition) *weighted Petri net*, or *weighted net*, is a tuple $N = (P, T, W)$ where P is a finite set of *places*, T is a finite set of *transitions*, with $P \cap T = \emptyset$ and W is a *weight* function $W\colon ((P \times T) \cup (T \times P)) \rightarrow \mathbb{N}$ giving the weight of each arc. A *Petri net system*, or *system*, is a tuple $\mathcal{S} = (N, M_0)$ where N is a net and M_0 is the *initial marking*, which is a mapping $M_0\colon P \rightarrow \mathbb{N}$ (hence a member of $\mathbb{N}^P$) indicating the initial number of *tokens* in each place. The *incidence matrix* C of the net is the integer $P \times T$-matrix with components $C(p,t) = W(t,p) - W(p,t)$.
A place $p \in P$ is *enabled by* a marking M if $M(p) \geq W(p,t)$ for every output transition t of p. A transition $t \in T$ is *enabled by* a marking M, denoted by $M[t\rangle$, if for all places $p \in P$, $M(p) \geq W(p,t)$. If t is enabled at M, then t can *occur* (or *fire*) in M, leading to the marking M' defined by $M'(p) = M(p) - W(p,t) + W(t,p)$; we note $M[t\rangle M'$. A marking M' is *reachable* from M if there is a sequence of firings leading from M to M'. The set of markings reachable from M is denoted by $[M\rangle$. The *reachability graph of* $\mathcal{S}$ is the labelled transition system $RG(\mathcal{S})$ with the set of vertices $[M_0\rangle$, the set of labels T, initial state M_0 and transitions $\{(M,t,M') \mid M, M' \in [M_0\rangle \wedge M[t\rangle M'\}$. A system $\mathcal{S}$ is *bounded* if $RG(\mathcal{S})$ is finite.

Vectors. The *support* of a vector is the set of the indices of its non-null components. Consider any net $N = (P, T, W)$ with its incidence matrix C. A *T-vector* is an element of $\mathbb{N}^T$; it is called *prime* if the greatest common divisor of its components is one (i.e., its components do not have a common non-unit factor). A *T-semiflow* ν of the net is a non-null T-vector such that $C \cdot \nu = \mathbb{0}$. A T-semiflow is called *minimal* when it is prime and its support is not a proper superset of the support of any other T-semiflow [27].
The *Parikh vector* $\mathbf{P}(\sigma)$ of a finite sequence σ of transitions is a T-vector counting the number of occurrences of each transition in σ, and the *support* of σ is the support of its Parikh vector, i.e., $supp(\sigma) = supp(\mathbf{P}(\sigma)) = \{t \in T \mid \mathbf{P}(\sigma)(t) > 0\}$.

Strong connectedness and cycles in LTS. The LTS is said *reversible* if, $\forall s \in [\iota\rangle$, we have $\iota \in [s\rangle$, i.e., it is always possible to go back to the initial state;

reversibility implies the strong connectedness of the LTS.
A sequence $s[\sigma\rangle s'$ is called a *cycle*, or more precisely a *cycle at (or around) state* s, if $s = s'$. A non-empty cycle $s[\sigma\rangle s$ is called *small* if there is no non-empty cycle $s'[\sigma'\rangle s'$ in TS with $\mathbf{P}(\sigma') \lneqq \mathbf{P}(\sigma)$ (the definition of Parikh vectors extending readily to sequences over the set of labels T of the LTS). A cycle $s[\sigma\rangle s$ is *prime* if $\mathbf{P}(\sigma)$ is prime. TS has the *prime cycle property* if every small cycle has a prime Parikh vector.
A *circular LTS* is a finite, strongly connected LTS that contains a unique cycle; hence, it has the shape of an oriented circle. The circular LTS *induced by* a word $w = w_1 \dots w_k$ is the LTS with initial state s_0 defined as $s_0[w_1\rangle s_1[w_2\rangle s_2 \dots [w_k\rangle s_0$.

All notions defined for labelled transition systems apply to Petri nets through their reachability graphs.

Petri net subclasses. A net N is *plain* if no arc weight exceeds 1; *pure* if $\forall p \in P\colon (p^\bullet \cap {}^\bullet p) = \emptyset$, where $p^\bullet = \{t \in T \mid W(p,t){>}0\}$ and ${}^\bullet p = \{t \in T \mid W(t,p){>}0\}$; CF (*choice-free* [10, 27]) or ON (place-output-nonbranching [4]) if $\forall p \in P\colon |p^\bullet| \leq 1$; a WMG (*weighted marked graph* [26]) if $|p^\bullet| \leq 1$ and $|{}^\bullet p| \leq 1$ for all places $p \in P$. The latter form a subclass of the choice-free nets; other subclasses are *marked graphs* [9], which are plain with $|p^\bullet| = 1$ and $|{}^\bullet p| = 1$ for each place $p \in P$, and *T-systems* [12], which are plain with $|p^\bullet| \leq 1$ and $|{}^\bullet p| \leq 1$ for each place $p \in P$.

Isomorphism and solvability. Two LTS $TS_1 = (S_1, \to_1, T, s_{01})$ and $TS_2 = (S_2, \to_2, T, s_{02})$ are isomorphic if there is a bijection $\zeta\colon S_1 \to S_2$ with $\zeta(s_{01}) = s_{02}$ and $(s,t,s') \in {\to_1} \Leftrightarrow (\zeta(s),t,\zeta(s')) \in {\to_2}$, for all $s, s' \in S_1$.
If an LTS TS is isomorphic to $RG(\mathcal{S})$, where $\mathcal{S}$ is a system, we say that $\mathcal{S}$ *solves* TS. Solving a word $w = \ell_1 \dots \ell_k$ amounts to solve the acyclic LTS defined by the single path $\iota[\ell_1\rangle s_1 \dots [\ell_k\rangle s_k$. A finite word w is *cyclically solvable* if the circular LTS induced by w is solvable. An LTS is WMG-solvable if a WMG solves it.

Separation problems. Let $TS = (S, \to, T, s_0)$ be a given labelled transition system. The theory of regions [1] characterises the solvability of an LTS through the solvability of a set of *separation problems*. In case the LTS is finite, we have to solve $\frac{1}{2}{\cdot}|S|{\cdot}(|S|{-}1)$ states separation problems and up to $|S|{\cdot}|T|$ event/state separation problems, as follows:

- A *region* of $(S, \to, T, s_0)$ is a triple $(\mathbb{R}, \mathbb{B}, \mathbb{F}) \in (S \to \mathbb{N}, T \to \mathbb{N}, T \to \mathbb{N})$ such that for all $s[t\rangle s'$, $\mathbb{R}(s) \geq \mathbb{B}(t)$ and $\mathbb{R}(s') = \mathbb{R}(s) - \mathbb{B}(t) + \mathbb{F}(t)$. A region models a place p, in the sense that $\mathbb{B}(t)$ models $W(p,t)$, $\mathbb{F}(t)$ models $W(t,p)$, and $\mathbb{R}(s)$ models the token count of p at the marking corresponding to s.
- A *states separation problem* (SSP for short) consists of a set of states $\{s, s'\}$ with $s \neq s'$, and it can be solved by a place distinguishing them, i.e., has a different number of tokens in the markings corresponding to the two states.
- An *event/state separation problem* (ESSP for short) consists of a pair $(s,t) \in S{\times}T$ with $\neg s[t\rangle$. For every such problem, one needs a place p such that $M(p) < W(p,t)$ for the marking M corresponding to state s, where W

refers to the arcs of the hoped-for net. On the other hand, for every edge $(s', t, s'') \in \rightarrow$ we must guarantee $M'(p) \geq W(p, t)$, M' being the marking corresponding to state s'.

If the LTS is infinite, also the number of separation problems (of each kind) becomes infinite.

A synthesis procedure does not necessarily lead to a connected solution. However, the technique of decomposition into prime factors described in [13, 14] can always be applied first, so as to handle connected partial solutions and recombine them afterwards. Hence, in the sequel, we focus on connected nets, without loss of generality. In the next section, we consider the synthesis problem of CF nets with exactly two different labels.

3 Reversible Binary CF Synthesis

In this section, we link the CF-solvability of a reversible LTS with 2 letters to the WMG-solvability.

Lemma 1 (Pure CF-solvability).
If a reversible LTS has a CF-solution, it has a pure CF-solution.

Proof. Let $TS = (S, \rightarrow, T, \iota)$ be a reversible LTS. First, we observe that, if $t \in T$ does not occur in $\rightarrow$, TS is solvable iff $TS' = (S, \rightarrow, T \setminus \{t\}, \iota)$ is solvable and a possible solution of TS is obtained by adding to any solution of TS' a transition t and a fresh place p, initially empty, with an arc from p to t (e.g. with weight 1), so that t is pure. We may thus assume that each label of T occurs in $\rightarrow$.

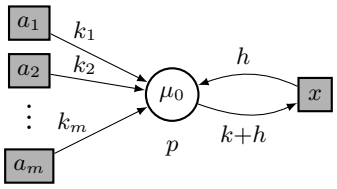


Fig. 1. A general pure ($h = 0$) or non-pure ($h > 0$) choice-free place p with initial marking μ_0. Place p has at most one outgoing transition named x. The set $\{a_1, \ldots, a_m\}$ comprises all other transitions, i.e., $T = \{x, a_1, \ldots, a_m\}$, and k_j denotes the weight of the arc from a_j to p (which could be zero).

The general form of a place in a CF-solution is exhibited in Figure 1. If $h = 0$, we are done, so that we shall assume $h > 0$. If $-h \leq k < 0$, the marking of p cannot decrease, and since x occurs in $\rightarrow$, the system cannot be reversible. If $k = 0$, for the same reason all the k_i's must be null too, $\mu_0 \geq h$, and we may drop p. Hence we assume that $k > 0$ and $\exists i : k_i > 0$.

Once x occurs, the marking of p is at least h, remains so, and since the system is reversible, all the reachable markings have at least h tokens in p. But then, if we replace p by a place p' with initially $\mu_0 - h$ tokens, the same k_i's and $h = 0$, we shall get exactly the same reachability graph, but with h tokens less in p' than in p. This will wipe out the side condition[5] for p, and repeating this for each side condition, we shall get an equivalent pure and choice-free solution. □

Theorem 1 (Reversible binary CF-solvability).
A binary reversible LTS is CF-solvable iff it is WMG-solvable.

Proof. If we have two labels, from Lemma 1, if there is a CF-solution, there will be one with places of the form exhibited in Figure 2, hence a WMG-solution. □

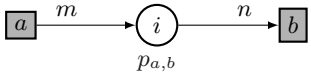


Fig. 2. A generic pure CF-place with two labels.

In the next section, the number of letters is no more restricted.

4 Cyclic WMG- and CF-solvablity

In this section, we recall and extend the conditions for WMG-solvability of some restricted classes of LTS formed by a single circuit, which were suggested in [15]. We gradually study the separation problems (SSPs in Subsection 4.1 and ESSPs in Subsection 4.2) for cyclical solvability with WMGs, and develop a language-theoretical characterisation of WMG-cyclically solvable sequences. The characterisation gives rise to a synthesis algorithm which is presented later. Unfortunately, most of these results cannot be directly extended to the more general class of CF-nets, which is demonstrated by examples in Subsection 4.3.

In the following, two distinct labels a and b are called *(circularly) adjacent* in a word w if $w = (w_1abw_2)$ or $w = (bw_3a)$ for some $w_1, w_2, w_3 \in T^*$. We denote by $p_{a,*}$ any place $p_{a,b}$ where b is adjacent to a. Also, since $|T| > 1$, at least one label is adjacent to t_0, and at least one is adjacent to the ones we exhibited, etc., until we get the whole set T, and we may start from any label t_i instead of t_0.

Theorem 2 (Sufficient condition for cyclic WMG-solvability [15]).
Consider any word w over any finite alphabet T such that $\mathbf{P}(w)$ is prime. Suppose the following: $\forall u = w_{|t_1t_2}$ (i.e., the projection[6] of w on $\{t_1, t_2\}$) for some

[5] A place p is a *side-condition* if ${}^\bullet p \cap p^\bullet \neq \emptyset$.

[6] The projection of a word $w \in A^*$ on a set $A' \subseteq A$ of labels is the maximum subword of w whose labels belong to A', noted $w_{|A'}$. For example, the projection of the word $w = \ell_1\,\ell_2\,\ell_3\,\ell_2$ on the set $\{\ell_1, \ell_2\}$ is the word $\ell_1\,\ell_2\,\ell_2$.

circularly adjacent labels t_1, t_2 in w, $u = v^\ell$ for some positive integer ℓ, $\mathbf{P}(v)$ is prime, and v is cyclically solvable by a circuit (i.e. a circular net as in Fig. 3). Then, w is cyclically solvable with a WMG.

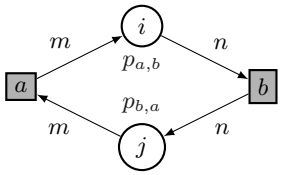


Fig. 3. A generic WMG solving a finite circular LTS induced by a word w over the alphabet $\{a, b\}$.

Theorem 3 (Cyclic WMG-solvability of ternary words [15]).
Consider a ternary word w over the alphabet T with Parikh vector (x, x, y) such that $gcd(x, y) = 1$. Then, w is cyclically solvable with a WMG if and only if $\forall u = w_{|t_1 t_2}$ such that $t_1 \neq t_2 \in T$, and $w = (w_1 t_1 t_2 w_2)$ or $w = (t_2 w_3 t_1)$, $u = v^\ell$ for some positive integer ℓ, $\mathbf{P}(v)$ is prime, and v is cyclically solvable by a circuit.

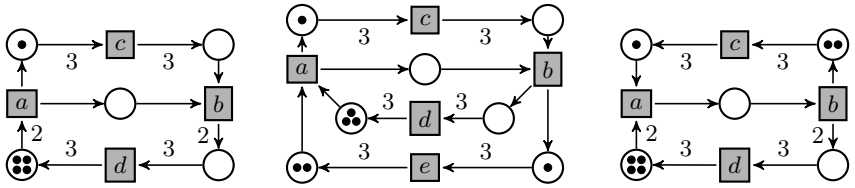


Fig. 4. The WMG on the left solves *aacbbdabd* cyclically, and the WMG in the middle solves *aacbbeabd* cyclically. On the right, the WMG solves *abcabdabd* cyclically.

For a circular LTS, the solvability of its binary projections by circuits is a sufficient condition, as specified by Theorem 2, but it turns out not to be a necessary one. Indeed, for the cyclically solvable sequence $w_1 = aacbbdabd$ (cf. left of Fig. 4), its binary projection on $\{a, b\}$ is $w_{1\,|a,b} = aabbab$ which is not cyclically solvable with a WMG (neither generally solvable). Looking only at the Parikh vector of the sequence is also not enough to establish its cyclical (un)solvability. For instance, sequences $w_2 = abcabdabd$ and $w_3 = abcbadabd$ are Parikh-equivalent: $\mathbf{P}(w_2) = \mathbf{P}(w_3) = (3, 3, 1, 2)$ (and also Parikh-equivalent to w_1), but w_2 is cyclically solvable with a WMG (e.g. with the WMG on the right of Fig. 4) and w_3 is not WMG-cyclically solvable.

All the binary projections of w_1 and w_3 are cyclically WMG-solvable, except $w_{i\,|a,b}$. But only the unsolvability of $w_{3\,|a,b}$ implies the unsolvability of w_3. Since all the w_i are Parikh-equivalent, then so are their binary projections. So, to find the difference we have to look at the sequences themselves, without abstracting

to Parikh-vectors. Since the projections $w_{1\,|a,b}$ and $w_{3\,|a,b}$ are equivalent (up to cyclical rotation and swapping a and b), it is also not enough to look only at the 'problematic' binary projections. We then look at the conditions for solvability of separation problems.

4.1 SSPs for Prime Cycles

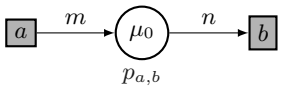


Fig. 5. A place of general form in a WMG.

Lemma 2 (SSPs are solvable for prime cycles). *If for the cyclical transition system $TS = (S, \rightarrow, T, s_0)$ defined by some word $w = t_0 \dots t_k$, where $S = \{s_0, \dots, s_k\}$, $\rightarrow = \{(s_{i-1}, t_{i-1}, s_i) \mid 1 \leq i \leq k\} \cup \{(s_k, t_k, s_0)\}$ with $t_i \in T$, $\mathbf{P}(w)$ is prime, then all the SSPs are solvable.*

Proof. If $|T| = 1$, then $k = 0$ (otherwise $\mathbf{P}(t_0 \dots t_k)$ is not prime) and $|S| = 1$, so that there is no SSP to solve. We may thus assume $|T| > 1$.

For $0 \leq i, j \leq k$ such that $s_i \neq s_j$ (so that $i \neq j$), we note $\mathbf{P}_{ij} = \mathbf{P}(t_i t_{i+1} \dots t_{j-1})$ if $i < j$ and $\mathbf{P}_{ij} = \mathbf{P}(t_i t_{i+1} \dots t_{k-1} t_k t_0 t_1 \dots t_{j-1})$ if $i > j$. For each pair of distinct labels $a, b \in T$ that are adjacent in TS, construct places $p_{a,b}$ (and $p_{b,a}$ since adjacency is commutative) as in Fig. 5 with

$$m = \frac{\mathbf{P}(w)(b)}{\gcd(\mathbf{P}(w)(a), \mathbf{P}(w)(b))},\ n = \frac{\mathbf{P}(w)(a)}{\gcd(\mathbf{P}(w)(a), \mathbf{P}(w)(b))}, \tag{1}$$

and $\mu_0 = n \cdot \mathbf{P}(w)(b)$. Clearly, the markings of $p_{a,b}$ reachable by repeatedly firing $u = w_{\downarrow ab}$ are always non-negative, and the initial marking is reproduced after each repetition of the sequence u. Consider two distinct states $s_i, s_j \in S$ (w.l.o.g. $i < j$). We now demonstrate that there is at least one place of the form $p_{a,b}$ such that $M_i(p_{a,b}) \neq M_j(p_{a,b})$, where M_l denotes the marking corresponding to state s_l for $0 \leq l \leq k$. If $j - i = 1$, then any place of the form $p_{t_i,*}$ distinguishes states s_i and s_j. The same is true if $j - i > 1$ but $\forall l \in [i, j-1] : t_l = t_i$. Otherwise, choose some letter a from $t_i \dots t_{j-1}$ and an adjacent letter b. Then $M_j(p_{a,b}) = M_i(p_{a,b}) + m \cdot \mathbf{P}_{ij}(a) - n \cdot \mathbf{P}_{ij}(b)$. If $M_i(p_{a,b}) \neq M_j(p_{a,b})$, place $p_{a,b}$ distinguishes s_i and s_j. Otherwise we have $m \cdot \mathbf{P}_{ij}(a) = n \cdot \mathbf{P}_{ij}(b)$, hence, due to the choice of m and n:

$$\frac{\mathbf{P}_{ij}(a)}{\mathbf{P}(w)(a)} = \frac{\mathbf{P}_{ij}(b)}{\mathbf{P}(w)(b)}$$

(so that b also belongs to $t_i \dots t_{j-1}$). Consider some other letter c which is adjacent to a or b. If place $p_{a,c}$ distinguishes s_i and s_j, we are done. Otherwise,

due to the choice of the arc weights for these places, we have

$$\frac{\mathbf{P}_{ij}(a)}{\mathbf{P}(w)(a)} = \frac{\mathbf{P}_{ij}(c)}{\mathbf{P}(w)(c)} = \frac{\mathbf{P}_{ij}(b)}{\mathbf{P}(w)(b)}.$$

Since $t_i \dots t_{j-1}$ is finite, by progressing along the adjacency relation, either we find a place which has different markings at s_i and s_j, or for all $a, b \in supp(t_i \dots t_{j-1})$ we have

$$\frac{\mathbf{P}_{ij}(a)}{\mathbf{P}(w)(a)} = \frac{\mathbf{P}_{ij}(b)}{\mathbf{P}(w)(b)}.$$

If $supp(t_i \dots t_{j-1}) = supp(w)$, $\mathbf{P}(w)$ is proportional to $\mathbf{P}(t_i \dots t_{j-1})$, but since $t_i \dots t_{j-1}$ is smaller than w (otherwise $s_i = s_j$) this contradicts the primality of $\mathbf{P}(w)$. Hence, there exist adjacent c and d such that $c \in supp(w) \setminus supp(t_i \dots t_{j-1})$ and $d \in supp(t_i \dots t_{j-1})$. For the place $p_{c,d}$ we have $M_j(p_{c,d}) \neq M_i(p_{c,d})$. □

This property has some similarities with Theorem 4.1 in [17], but the preconditions are different.

The reachability graph of any CF net, hence of any WMG, satisfies the prime cycle property [5, 6]. Thus, primeness of a sequence allows us skip checking of SSPs when looking at solvability within these two classes of Petri nets.

4.2 ESSPs in Cyclical Solvability with WMGs

Now, consider further conditions for the cyclical WMG-solvability of a sequence $w = t_0 \dots t_k$, where $\mathbf{P}(w)$ is prime. Let us assume that the system $((P, T, W), M_0)$ is a WMG solving w cyclically. Due to the definition of WMGs, all the places that we have to consider are of the form schematised in Fig. 6. The arc weights may differ due to the integer parameter $l \geq 1$, but the ratio $\frac{W(a,p_{a,b})}{W(p_{a,b},b)} = \frac{m}{n}$ is determined by the Parikh vector of w and its cyclical solvability (let us notice that the initial marking is to be defined). Moreover, we have to consider only these places, which are connected to the pairs of adjacent transitions in w. Indeed, if $w = u_1 \mid_{s_i} a \mid_{s_{i+1}} b\, u_2$, where s_i is the state reached after performing u_1 and s_{i+1} is the state reached after performing $u_1 a$, then any place that solves the ESSP[7] $\neg M_i[b\rangle$ is an input place for b. On the other hand, any place whose marking at M_{s_i} differs from its marking at $M_{s_{i+1}}$ is connected to a. Hence, a place $p \in P$ solving $\neg M_i[b\rangle$ is of the form $p_{a,b}$. Since p is only affected by a and b, it also disables b at all the states between s_l and s_i in w when it is of the form $w = u_3 \mid_{s_j} t_j \mid_{s_{j+1}} b^+ \mid_{s_l} u_4 \mid_{s_i} abu_2$ with $\mathbf{P}(u_4)(b) = 0$ (in the case there is no b in the prefix between s_0 and abu_2, $s_l = s_0$). Analogously, if $t_j \neq b$, there must be a place $q \in P$ of the form $p_{t_j,b}$ that solves $\neg M_{s_j}[b\rangle$. Doing so, we ascertain that the places of the form schematised in Fig. 6 for the adjacent pairs of transitions are sufficient to handle all the ESSPs.

[7] Assuming $b \neq a$.

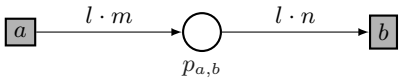


Fig. 6. A general place from a to b of a WMG solution (l may be any multiple of $1/\gcd(m,n)$).

In fact, for every pair of adjacent transitions a and b in w, a single place of the form $p_{a,b}$ is sufficient. Indeed, assume there are $p_1, p_2 \in P$ of the form $p_{a,b}$. If $\frac{M_0(p_1)}{\gcd(W(a,p_1),W(p_1,b))} \geq \frac{M_0(p_2)}{\gcd(W(a,p_2),W(p_2,b))}$ then for any $M \in [M_0\rangle$, $M(p_1) < W(p_1,b)$ implies $M(p_2) < W(p_2,b)$. Hence, p_1 is redundant in the system, and the following is true.

Lemma 3. *If $w \in T^*$ is cyclically solvable by a WMG, there exists a WMG $S = ((P,T,W),M_0)$, where P consists of places $p_{a,b}$, for each pair of distinct circularly adjacent a and b (i.e., either $w = u_1abu_2$ or $w = bu_3a$).*

Let w be cyclically solvable with a WMG $S = ((P,T,W),M_0)$ as in Lemma 3, and place $p \in P$ be of the form $p_{a,b}$ (as in Fig. 6, with $l = 1$) for an adjacent pair ab. Choose two successive ab's in $w = u_1\,|_{s_r}a\,|_{s_{r+1}}\,b\,|_{s_{r+2}}\,\ldots\,|_{s_q}\,a\,|_{s_{q+1}}\,b\,u_2$ with possibly other letters between s_{r+2} and s_q (if there is only one ab, apply the argumentation while wrapping around w circularly, i.e., with $s_r = s_q$). Since p solves ESSPs $\neg s_r[b\rangle$ and $\neg s_q[b\rangle$, the next inequalities hold true, where μ_r denotes the marking of $p_{a,b}$ at state s_r:

$$\begin{array}{rll} \neg s_r[b\rangle: & \mu_r & < n \\ s_{r+1}[b\rangle: & \mu_r + m & \geq n \\ \forall j:\ r \leq j \leq q: & \mu_r + \mathbf{P}_{rj}(a)\cdot m - \mathbf{P}_{rj}(b)\cdot n & \geq 0 \\ \neg s_q[b\rangle: & \mu_r + \mathbf{P}_{rq}(a)\cdot m - \mathbf{P}_{rq}(b)\cdot n & < n \end{array} \tag{2}$$

From the first and the third line of (2) we get $\mathbf{P}_{rj}(a)\cdot m - \mathbf{P}_{rj}(b)\cdot n > -n$. This implies:

$$\frac{\mathbf{P}_{rj}(b)-1}{\mathbf{P}_{rj}(a)} < \frac{m}{n},\ r < j \leq q. \tag{3}$$

From the third and the fourth line of (2) we obtain

$$(\mathbf{P}_{rq}(a) - \mathbf{P}_{rj}(a))\cdot m - (\mathbf{P}_{rq}(b) - \mathbf{P}_{rj}(b))\cdot n < n.$$

For $\mathbf{P}_{jq}(a) \neq 0$, since $\mathbf{P}_{rq} = \mathbf{P}_{rj} + \mathbf{P}_{jq}$ this inequality can be written as

$$\frac{m}{n} < \frac{\mathbf{P}_{jq}(b)+1}{\mathbf{P}_{jq}(a)}. \tag{4}$$

Thus, from (3) and (4) we have a necessary condition for solvability in the following sense.

Lemma 4 (A necessary condition for cyclical solvability with a WMG). *If $w \in T^*$ is cyclically solvable by a WMG, then for any adjacent transitions a and b in w, and any two successive-up-to-rotation occurrences of ab in $w = u_1 \,|_{s_r} a\, b \,\ldots\, |_{s_q} a\, b\, u_2$, the inequality*

$$\frac{\mathbf{P}_{rj}(b) - 1}{\mathbf{P}_{rj}(a)} < \frac{m}{n} < \frac{\mathbf{P}_{jq}(b) + 1}{\mathbf{P}_{jq}(a)} \tag{5}$$

holds true, where m, n are as in (1), $r < j \leq q$, and the right inequality is omitted when $\mathbf{P}_{jq}(a) = 0$.

In particular, Lemma 4 explains the cyclical unsolvability of the word $w_3 = |_{s_r}\, ab\, c\, b\, |_{s_j}\, a\, d\, |_{s_q}\, ab\, d$. Indeed, $\mathbf{P}(w_3)(b) = 3 = \mathbf{P}(w_3)(a)$, so that $m/n = 1$ and $1 \not< \frac{0+1}{1} = \frac{\mathbf{P}_{jq}(b)+1}{\mathbf{P}_{jq}(a)}$.

Lemma 5 (A sufficient condition for cyclical solvability by a WMG). *If $w \in T^*$ has a prime Parikh vector, and for each pair of circularly adjacent ab in $w = \ldots\, |_q\, a\, b\, \ldots$, the inequality*

$$\frac{m}{n} < \frac{\mathbf{P}_{jq}(b) + 1}{\mathbf{P}_{jq}(a)},\ j \neq q \tag{6}$$

holds true, then w is cyclically solvable by a WMG.

Proof: We have earlier proved (Lemma 2) that all SSPs are solvable for prime cycles. Let us now consider the ESSPs at states s as in $w = \ldots |_s ab \ldots$, i.e. $\neg s[b\rangle$. Since we are looking for a WMG-solution, all the sought places are of the form $p_{a,b}$ (see Lemma 3 and Fig. 6) with m, n as in (1). To define the initial marking of $p_{a,b}$, let us put $n \cdot \mathbf{P}(w)(b)$ tokens on it and fire the sequence w once completely. Choose some state s' in $w = \ldots |_{s'}\, a \ldots$ such that the number k of tokens on $p_{a,b}$ at state s' is minimal (it may be the case that such an s' is not unique; we can choose any such state). Define $M_0(p_{a,b}) = n \cdot \mathbf{P}(w)(b) - k$ as the initial marking of $p_{a,b}$. By construction, the firing of w reproduces the markings of $p_{a,b}$ and M_0 guarantees their non-negativity. Let us now demonstrate that the constructed place $p_{a,b}$ solves all the ESSPs $\neg s[b\rangle$, where $w = \ldots |_s ab \ldots$. Consider such a state s in w (w.l.o.g. we assume $s \neq s'$, since s' certainly disables b). From $w = u_1 \,|_{s'} a \,\ldots\, |_s\, a\, b\, u_2$, and from inequality (6) for $s_j = s'$ we get $\mathbf{P}_{s's}(a) \cdot m - \mathbf{P}_{s's}(b) \cdot n < n$. Since $M_{s'}(p_{a,b}) = 0$, $M_s(p_{a,b}) = M_{s'}(p_{a,b}) + \mathbf{P}_{s's}(a) \cdot m - \mathbf{P}_{s's}(b) \cdot n < n$, i.e., $p_{a,b}$ disables b at state s.

Now, we show that places of the form $p_{a,b}$ also solve the other ESSPs against b, i.e., at the states where b is not the subsequent transition. Sequence w (up to rotation) can be written as $w = u_1\, b^{x_1}\, u_2\, b^{x_2} \,\ldots\, u_l\, b^{x_l}$, $1 \leq l \leq \mathbf{P}(w)(b)$, and for $1 \leq i \leq l$: $x_i > 0$, $u_i \in (T \setminus \{b\})^+$. Transition b has to be deactivated at all the states between neighbouring b-blocks. Consider an arbitrary pair of such blocks b^{x_j} and $b^{x_{j+1}}$ in $w = \ldots\, b^{x_j}\, u_j\, b^{x_{j+1}} \ldots = \ldots\, b^{x_j} \,|_s\, u'_j \,|_r\, t\, b^{x_{j+1}} \ldots$, with $u_j = u'_j t$. Place $p_{t,b}$ does not allow b to fire at state r. We have to check

that b is not activated at any state between s and r, i.e., it is not activated 'inside' u'_j. If u'_j is empty, then $s = r$, and we are done. Let $u'_j \neq \varepsilon$. Due to $\mathbf{P}(u'_j)(b) = \mathbf{P}(u_j)(b) = 0$, the marking of place $p_{t,b}$ cannot decrease from s to r, i.e., $M_s(p_{t,b}) \leq M_{s''}(p_{t,b}) \leq M_r(p_{t,b})$ for any s'' 'inside' u'_j. Since $p_{t,b}$ deactivates b at r, it then deactivates b at all states between s and r, inclusively. □

From Lemma 4 and Lemma 5 we can deduce the following characterisation.

Theorem 4 (A characterisation of cyclical solvability with a WMG). *A sequence $w \in T^*$ is cyclically solvable with a WMG iff $\mathbf{P}(w)$ is prime and for any pair of circularly adjacent labels in w, for instance $w = \ldots |_q\, ab \ldots$,*

$$\frac{m}{n} < \frac{\mathbf{P}_{jq}(b) + 1}{\mathbf{P}_{jq}(a)},\ j \neq q$$

holds true with m, n as in (1). A WMG-solution can be found with the places of the form $p_{a,b}$ for every such pair of a and b.

Based on the characterisation from Theorem 4 and the considerations above, Algorithm 1 below synthesizes a cyclical WMG-solution for a given sequence $w \in T^*$, if one exists, with a runtime in $\mathcal{O}(|w|^2)$. For a comparison, the general region-based synthesis typically uses ILP-solvers, and for Karmarkar's algorithm [24] (which is known to be efficient) we may expect a running time of $\mathcal{O}(|w|^3 \cdot L(|w|))$, with a logarithmic factor $L(|w|) = \log(|w|) \cdot \log(\log(|w|))$. Note that, with this general approach, some redundant places may be constructed, but they can be reduced in a post-processing phase.

4.3 CF-solvability vs WMG-solvability of Cycles

The class of WMGs is clearly a proper subset of the class of CF nets. If we are only looking at cyclical solvability of sequences, this inclusion remains strict, i.e., there exist sequences which are cyclically solvable by a CF net but have no cyclical solution in the form of a WMG. E.g., the sequence $w = abcbad$ has a cyclical CF-solution (cf. Fig. 7). On the other hand, for $a\,|_r\, b\, c\,|_q\, b\, a\, d$ we have $\frac{\mathbf{P}(w)(a)}{\mathbf{P}(w)(b)} = \frac{2}{2} \not< \frac{0+1}{1} = \frac{\mathbf{P}_{rq}(a)+1}{\mathbf{P}_{rq}(b)}$ which, by Theorem 3, implies the cyclical unsolvability of w by a WMG.

From Lemma 3, for the cyclical solvability by a WMG it is enough to use only places between adjacent transitions. For the sequence *abcbad* in Fig. 7, transition b follows a and c, and the input place of b in the CF-solution is an output place for both a and c. The situation is similar for transition a, which follows b and d. However, this is not always the case when we are looking for a solution in the class of CF nets. For instance, the sequence *cabdaaabeab* is cyclically solvable by a CF net (see Fig. 8). In this sequence, b always follows a. But in order to solve ESSPs against b, we need a place which is an output place for c and e (in addition to a).

Algorithm 1: WMG-cycles

input : $w \in T^*$, $T = \{t_0, \ldots, t_{n-1}\}$
output: A WMG N cyclically solving w if it exists
var: $T[0\,..\,|T|-1] = (t_0, \ldots, t_{n-1})$, $v[0\,..\,|w|-1]$, a, b, na, nb, ia, ib, M, $Mmin$;
compute the Parikh-vector $\mathbf{P}[0\,..\,|T|-1]$ of w;
if $\mathbf{P}$ *is not prime* **then return** *unsolvable* ; `// Parikh-primeness`
$b \leftarrow w[0]$;
for $j = 0$ **to** $|T|-1$ **do** `// index of` b
  if $b = T[j]$ **then** $ib \leftarrow j$;
for $i = 0$ **to** $|w|-1$ **do**
  $v \leftarrow w[i] \ldots w[|w|-1]w[0] \ldots w[i-1]$; `// rotation of` w
  $a \leftarrow b$, $b \leftarrow v[1]$, $ia \leftarrow ib$; `// fix first adjacent pair`
  for $j = 0$ **to** $|T|-1$ **do**
    if $b = T[j]$ **then** $ib \leftarrow j$;
  $na \leftarrow 1$, $nb \leftarrow 1$;
  if $a \neq b$ **then**
    for $k = 2$ **to** $|w|-1$ **do**
      if $\frac{\mathbf{P}[ib]}{\mathbf{P}[ia]} \geq \frac{\mathbf{P}[ib]-nb+1}{\mathbf{P}[ia]-na}$ **then**
        return *unsolvable* ; `// check solvability condition`
      if $v[k] = T[ia]$ **then** $na \leftarrow na+1$;
      if $v[k] = T[ib]$ **then** $nb \leftarrow nb+1$;
    $M \leftarrow \mathbf{P}[ia] \cdot \mathbf{P}[ib]$, $Mmin \leftarrow M$;
    for $k = 0$ **to** $|w|-1$ **do** `// find initial marking`
      if $w[k] = a$ **then** $M \leftarrow M + \mathbf{P}[ib]$;
      if $w[k] = b$ **then** $M \leftarrow M - \mathbf{P}[ia]$;
      if $M < Mmin$ **then** $Mmin \leftarrow M$;
    add new place $p_{T[ia],T[ib]}$ to N with
    $W(T[ia], p) = \mathbf{P}[ib]$, $W(p, T[ib]) = \mathbf{P}[ia]$, $M_0 = \mathbf{P}[ia] \cdot \mathbf{P}[ib] - Mmin$;
return N

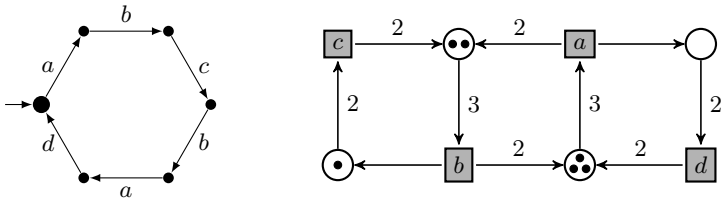


Fig. 7. Sequence *abcbad* is cyclically solved by the CF net on the right.

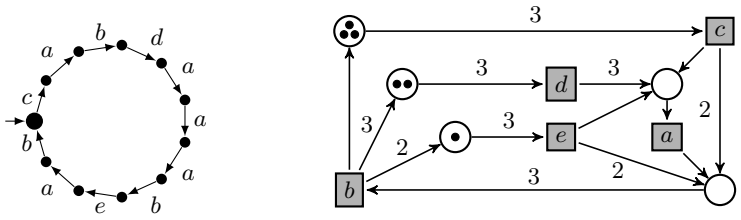


Fig. 8. Sequence *cabdaaabeab* is cyclically solved by the CF-net on the right.

Indeed, if there is a place $p_{a,b}$ as on the left of Fig. 9 which solves ESSPs against b, then for $ca\,|_s\,bdaa\,|_q\,abeab$

$$\begin{array}{lll} s[b\rangle & : \mu_0 + 3 & \geq 5 \\ \neg q[b\rangle & : \mu_0 + 3 \cdot 3 - 5 & < 5 \end{array}$$

Subtracting the first inequality from the second one, we get $6 - 5 < 0$, which is a contradiction. Hence, a place of form $p_{a,b}$ cannot solve all ESSPs against b in *cabdaaabeab*.

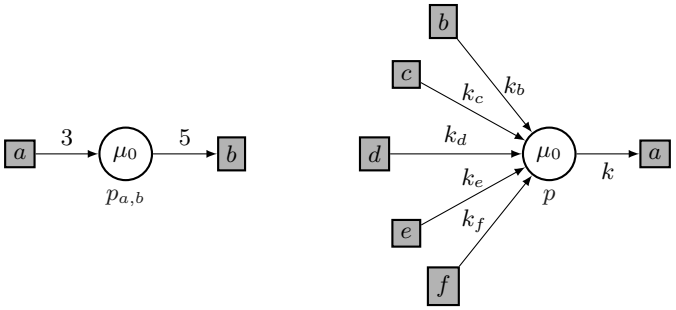


Fig. 9. A place from a to b (left); a place of a CF net with 6 transitions (right).

In a WMG, a place can have at most one input transition. This restriction is relaxed for choice-free nets and multiple inputs are allowed. Nevertheless, this does not imply that a single input place for each transition will always be sufficient. As an instance, consider the sequence $bcafdeaaabcdaafdcaaa$ (see Fig. 10) which is cyclically solvable with a CF net.

Assume that we can solve all ESSPs against transition a with a single place p as on the right of Fig. 9 (we know that we do not need any side-conditions). Then, for p and $w = |_{s_0}\, b\ c\ a\ f\ d\,|_{s_5}\, e\,|_{s_6}\, a\ a\ a\ b\ c\,|_{s_{11}}\, d\,|_{s_{12}}\, a\ a\ f\ d\,|_{s_{16}}\, c\,|_{s_{17}}\, a\ a\ a$, the

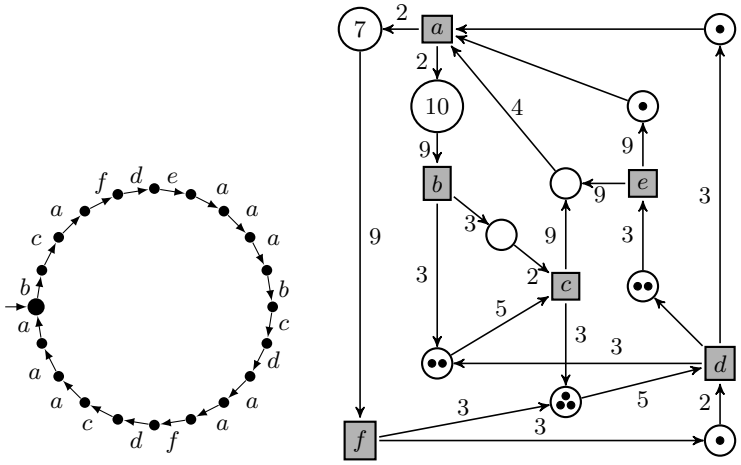


Fig. 10. $w = bcafdeaaabcdaafdcaaa$ is cyclically solved by the CF net on the right.

following system of inequalities must hold true:

$$\begin{array}{lll}
\text{cycle} & : 2 \cdot k_b + 3 \cdot k_c + 3 \cdot k_d + k_e + 2 \cdot k_f & = 9 \cdot k \quad (0) \\
\neg s_5[a\rangle & : \mu_0 + k_b + k_c + k_d + k_f - k & < k \quad (1) \\
s_6[aaa\rangle & : \mu_0 + k_b + k_c + k_d + k_e + k_f - k & \geq 3 \cdot k \quad (2) \\
\neg s_{11}[a\rangle & : \mu_0 + 2 \cdot k_b + 2 \cdot k_c + k_d + k_e + k_f - 4 \cdot k & < k \quad (3) \\
s_{12}[aa\rangle & : \mu_0 + 2 \cdot k_b + 2 \cdot k_c + 2 \cdot k_d + k_e + k_f - 4 \cdot k & \geq 2 \cdot k \quad (4) \\
\neg s_{16}[a\rangle & : \mu_0 + 2 \cdot k_b + 2 \cdot k_c + 3 \cdot k_d + k_e + 2 \cdot k_f - 6 \cdot k & < k \quad (5) \\
s_{17}[aaa\rangle & : \mu_0 + 2 \cdot k_b + 3 \cdot k_c + 3 \cdot k_d + k_e + 2 \cdot k_f - 6 \cdot k & \geq 3 \cdot k \quad (6)
\end{array}$$

From the system above we obtain:

$$\begin{array}{lll}
(2) - (1) & : k_e & > 2 \cdot k \\
(4) - (3) & : k_d & > k \\
(6) - (5) & : k_c & > 2 \cdot k
\end{array}$$

which implies $3 \cdot k_c + 3 \cdot k_d + k_e > 13 \cdot k$, contradicting the equality (0). Hence, the ESSPs against a cannot be solved by a single place.

5 Conclusions and Perspectives

In this work, we specialised previous methods of analysis and synthesis to the CF nets and their WMG subclass, two useful subclasses of weighted Petri nets allowing to model various real-world applications.

We highlighted the correspondance between CF-solvability and WMG-solvability for binary alphabets. We also tackled the case of an LTS formed of a single circuit with an arbitrary number of letters, for which we developed a characterisation of WMG-solvability together with a dedicated and efficient synthesis algorithm. Finally, we discussed the applicability of our conditions to CF synthesis.

As a natural continuation of the work, we expect extensions of our results in two directions: generalising the class of goal-nets (e.g. to choice-free or fork-attribution nets), and relaxing the restrictions for the LTS under consideration.

Acknowledgements

We would like to thank the anonymous referees for their useful suggestions.

References

1. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Springer-Verlag (2015)
2. Barylska, K., Best, E., Erofeev, E., Mikulski, L., Piatkowski, M.: On binary words being Petri net solvable. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015, Brussels, Belgium. pp. 1–15 (2015)
3. Barylska, K., Best, E., Erofeev, E., Mikulski, L., Piatkowski, M.: Conditions for Petri net solvable binary words. T. Petri Nets and Other Models of Concurrency **11**, 137–159 (2016). https://doi.org/10.1007/978-3-662-53401-4_7
4. Best, E., Devillers, R.: Synthesis and reengineering of persistent systems. Acta Inf. **52**(1), 35–60 (2015). https://doi.org/10.1007/s00236-014-0209-7
5. Best, E., Devillers, R.: Characterisation of the state spaces of marked graph Petri nets. Information and Computation **253**(3), 399–410 (2017)
6. Best, E., Devillers, R., Schlachter, U.: Bounded choice-free Petri net synthesis: Algorithmic issues. Acta Informatica (2017)
7. Best, E., Devillers, R., Schlachter, U., Wimmel, H.: Simultaneous petri net synthesis. Sci. Ann. Comp. Sci. **28**(2), 199–236 (2018)
8. Best, E., Hujsa, T., Wimmel, H.: Sufficient conditions for the marked graph realisability of labelled transition systems. Theoretical Computer Science (2017)
9. Commoner, F., Holt, A., Even, S., Pnueli, A.: Marked directed graphs. J. Comput. Syst. Sci **5**(5), 511–523 (1971)
10. Crespi-Reghizzi, S., Mandrioli, D.: A decidability theorem for a class of vector-addition systems. Inf. Process. Lett. **3**(3), 78–80 (1975). https://doi.org/10.1016/0020-0190(75)90020-4
11. Delosme, J.M., Hujsa, T., Munier-Kordon, A.: Polynomial sufficient conditions of well-behavedness for weighted join-free and choice-free systems. In: 13th International Conference on Application of Concurrency to System Design. pp. 90–99 (July 2013). https://doi.org/10.1109/ACSD.2013.12
12. Desel, J., Esparza, J.: Free Choice Petri Nets, Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, New York, USA (1995)

13. Devillers, R.: Products of Transition Systems and Additions of Petri Nets. In: Proc. 16th International Conference on Application of Concurrency to System Design (ACSD 2016) J. Desel and A. Yakovlev (eds). pp. 65–73 (2016)
14. Devillers, R.: Factorisation of transition systems. Acta Informatica (2017)
15. Devillers, R., Erofeev, E., Hujsa, T.: Synthesis of weighted marked graphs from constrained labelled transition systems. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2018 Satellite event of the conferences: Petri Nets 2018 and ACSD 2018, Bratislava, Slovakia, June 25, 2018. pp. 75–90 (2018)
16. Devillers, R., Hujsa, T.: Analysis and synthesis of weighted marked graph Petri nets. In: Khomenko, V., Roux, O.H. (eds.) Application and Theory of Petri Nets and Concurrency: 39th International Conference, PETRI NETS 2018, Bratislava, Slovakia, 2018, Proceedings. pp. 1–21. Springer (2018)
17. Devillers, R., Hujsa, T.: Analysis and synthesis of weighted marked graph Petri nets: Exact and approximate methods. Fundamenta Informaticae (2019)
18. Erofeev, E., Barylska, K., Mikulski, L., Piatkowski, M.: Generating all minimal Petri net unsolvable binary words. In: Proceedings of the Prague Stringology Conference 2016, Prague, Czech Republic. pp. 33–46 (2016)
19. Erofeev, E., Wimmel, H.: Reachability graphs of two-transition Petri nets. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2017, Zaragoza, Spain. pp. 39–54 (2017)
20. Hujsa, T.: Contribution to the study of weighted Petri nets. Ph.D. thesis, Pierre and Marie Curie University, Paris, France (2014)
21. Hujsa, T., Delosme, J.M., Munier-Kordon, A.: On the reversibility of well-behaved weighted choice-free systems. In: Ciardo, G., Kindler, E. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 334–353. Springer (2014)
22. Hujsa, T., Delosme, J.M., Munier-Kordon, A.: Polynomial sufficient conditions of well-behavedness and home markings in subclasses of weighted Petri nets. ACM Trans. Embed. Comput. Syst. **13**(4s), 141:1–141:25 (Jul 2014). https://doi.org/10.1145/2627349
23. Hujsa, T., Devillers, R.: On liveness and deadlockability in subclasses of weighted Petri nets. In: van der Aalst, W., Best, E. (eds.) Application and Theory of Petri Nets and Concurrency: 38th International Conference, PETRI NETS 2017, Zaragoza, Spain, June 25–30, 2017, Proceedings. pp. 267–287. Springer (2017). https://doi.org/10.1007/978-3-319-57861-3_16
24. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**(4), 373–395 (Dec 1984). https://doi.org/10.1007/BF02579150
25. Murata, T.: Petri nets: properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (April 1989)
26. Teruel, E., Chrzastowski-Wachtel, P., Colom, J.M., Silva, M.: On weighted T-systems. In: Jensen, K. (ed.) 13th International Conference on Application and Theory of Petri Nets and Concurrency (ICATPN), LNCS. vol. 616, pp. 348–367. Springer, Berlin, Heidelberg (1992)
27. Teruel, E., Colom, J.M., Silva, M.: Choice-Free Petri Nets: a Model for Deterministic Concurrent Systems with Bulk Services and Arrivals. IEEE Transactions on Systems, Man, and Cybernetics, Part A **27**(1), 73–83 (1997). https://doi.org/10.1109/3468.553226
28. Teruel, E., Silva, M.: Structure theory of Equal Conflict systems. Theoretical Computer Science **153**(1&2), 271–300 (1996)

Can a Single Transition Stop an Entire Petri Net?

Jörg Desel

FernUniversität in Hagen, Germany
joerg.desel@fernuni-hagen.de

Abstract. A transition t eventually stops a place/transition Petri net if each reachable marking of the net enables only finite occurrence sequences without occurrences of t (i.e., every infinite occurrence sequence enabled at this marking contains occurrences of t). Roughly speaking, when t is stopped then all transitions of the net stop eventually. This contribution shows how to identify stopping transitions of bounded nets using the reachability graph and of unbounded nets using the coverability graph.

1 Introduction

We consider the following problem in this paper: Assume a place/transition Petri net and a transition t of this net. Can we eventually stop the behavior of the net by forbidding occurrences of t in occurrence sequences enabled at an arbitrary reachable marking m, or, equivalently, does no reachable marking m enable an infinite occurrence sequence without occurrences of t? If this is the case then we say that *transition t eventually stops the Petri net*. If t does not stop the net eventually, then some reachable marking enables an infinite occurrence sequence without occurrences of t. However, even if t does not stop the net eventually, there might be occurrence sequences (with or without occurrences of t) leading to a deadlock.

Apparently, this question is relevant for several applications of Petri nets. For example, given a robot (or any kind of machine) modeled by a Petri net, can some component modeled by a particular transition be used as an off switch? As we know from our computers, immediate stops are not always desirable, but rather forced shut down processes. A transition t stops a Petri net model eventually if it enforces a shutdown process which will eventually lead to a marking which enables no transition, except possibly transition t.

The problem tackled in this article could be solved by any standard mechanism involving temporal logics, for example the temporal logic LTL. In [5] it is shown that the model checking problem for Petri nets and LTL formulas is decidable, although according algorithms applied to unbounded Petri nets have a huge complexity. Instead, this article provides a solution which is purely based on Petri net analysis techniques. A typical advantage of these techniques is that the user gets more insight to the actual behavior of the net. Often, analysis

methods tailored for Petri nets are more efficient as analysis techniques based on a translation to other languages, at least for certain classes of inputs. This might also be the case for the approach presented in this paper; a detailed study to identify such classes is, however, still missing and a topic for further research.

Throughout this paper we consider place/transition Petri nets without arc weights, capacity restrictions or inhibitor arcs. We call these place/transition Petri nets just *nets*. For definitions and notations, see any textbook on Petri nets, e.g. [7] or [4]. As usual, we assume that the sets of places and transitions of a net are finite. We do, however, consider *unbounded nets*, i.e., nets with unbounded places (a place is *unbouded* if, for any number b, some reachable marking assigns more than b tokens to the place). We assume the concepts of *reachability graph* and *tree* to be known, and also the concept of *coverability graph* for unbounded nets (this concept goes back to [6]). The coverability graph represents aspects of infinite behavior by finite means, and thus abstracts heavily from behavioral details. However, it can be used to identify unbounded places. Notice that often the coverability graph is defined as a result of a non-deterministic algorithm and is hence not unique. The algorithm constructs the finite reachability graph for bounded nets and a finite coverability graph otherwise.

Recall that a (reachability or coverability) graph is a directed graph with initial vertex and arcs labeled by transition names. Vertices of reachability graphs represent *reachable markings* of the considered net, whereas vertices of coverability graphs represent so-called ω*-markings*, which assign to each place either a non-negative integer, representing its actual token count, or the symbol ω, representing arbitrarily many tokens.

Notice that every two vertices of a (reachability or coverability) graph can be connected by two distinct arcs, labeled by two different transition names, whenever both transitions lead from the same source vertex to the same target vertex. Recall also that the source vertex and the target vertex of an arc can be identical.

A *path* of a graph is a finite nonempty sequence of arcs such that the target vertex of each (except the last) arc coincides with the source vertex of its subsequent arc. A path is a *closed path* if the target vertex of its last arc coincides with the source vertex of its first arc. A closed path is a *cycle* if moreover no vertex is source of more than one arc of the path, i.e., the path does not pass through any vertex more than once.

Let us finally recall some important properties of reachability and coverability graphs:

- The reachability graph of a net is finite if and only if the net is bounded.
- A coverability graph of a net is always finite.
- Reachability and coverability graphs are deterministic, i.e., no vertex is source of two distinct arcs with the same label.
- For each finite occurrence sequence of a net enabled at the initial marking, there is a unique path of the reachabilty / coverability graph starting at the initial vertex.

2 Terminating Petri nets

To warm up, we first consider the question whether a net terminates eventually, i.e., whether all its occurrence sequences are finite.

Obviously, a bounded net terminates if and only if its reachability graph has no cycles. In fact, if the reachability graph has a cycle, then each occurrence sequence from the initial marking to any marking represented by a vertex of the cycle can be extended infinitely, following the arcs of the cycle (remember that each vertex of the reachability graph represents a reachable marking). Conversely, a bounded net has only finitely many reachable markings, because the set of places of the net is finite. If the net does not terminate, it has an infinite occurrence sequence and therefore finite occurrence sequences of arbitrary length. Since each finite occurrence sequence corresponds to a directed path of the reachability graph, each occurrence sequence of sufficient length (choose the number of reachable markings) corresponds to a directed path that passes through at least one vertex more than once; thus the reachability graph has a closed path, and therefore it has a cycle.

Unbounded nets do not terminate anyway. To see this, consider the construction of the reachability tree. Since the set of transitions is finite, each vertex of this tree has finitely many immediate successors. By König's Lemma, the tree has an infinite path, corresponding to an infinite occurrence sequence.

Hence, an obvious algorithm to check termination of a net first checks boundedness, for example by the coverability graph construction. In case the considered net is bounded, the algorithm constructs the reachabilty graph and checks whether this graph has a cycle. Actually, this two-step approach is not necessary, because the coverability graph of a bounded net equals its reachability graph, and cyclicity of this graph is implicitly checked during the coverability graph construction. A perhaps more elegant algorithm[1] first adds a place to the net which has all transitions of the net in its pre-set and no transition in its post-set, and then checks boundedness of this place, again by construction of the coverability graph. Obviously, this additional place is bounded if and only if the length of all occurrence sequences is bounded. Since the set of transitions is finite, this is the case if and only if there is no infinite occurrence sequence.

3 Termination After Stopping a Transition – The Bounded Case

We now come back to the question asked initially: Does a transition t of a net stop the net eventually? This is the converse of the question: Is there an infinite occurrence sequence, enabled at some reachable marking, without occurrences of t? An even simpler formulation of the same property is: Is there an initially enabled infinite occurrence sequence with only finitely many occurrences of t? In fact, an infinite occurrence sequence enabled at a reachable marking m is suffix

[1] communicated by Karsten Wolf

of an infinite sequence enabled initially, and the finite prefix up to m can contain only finitely many occurrences of t. Conversely, assume an infinite occurrence sequence containing only finitely many occurrences of t. Then the minimal prefix containing all these t-occurrences leads to a reachable marking which enables the according infinite suffix without occurrences of t.

For bounded nets, there is thus a very simple algorithmic solution to the problem whether a transition t eventually stops its net, based on the following proposition.

Proposition 1. *A transition t of a bounded net eventually stops the net if and only if the reachability graph of the net has no cycle without an arc labeled by t.*

Proof. Assume that the reachability graph has a cycle without a t-labeled arc. Then some initially enabled infinite occurrence sequence starts with a finite sequence leading to some marking represented by a vertex of this cycle (which might include occurrences of t) and then runs along the cycle infinitely. Hence this infinite occurrence sequence has only finitely many occurrences of t.

Conversely, assume that each cycle of the reachability graph has at least one t-labeled arc. Let m be an arbitrary reachable marking. Each sufficiently long occurrence sequence enabled at m passes through some marking at least twice, because the net is bounded. Hence the according path of the reachability graph passes through some vertex at least twice. The subsequence starting and ending with that vertex corresponds to a closed path. Each closed path contains all arcs of at least one cycle, and thus by assumption also an arc labeled by t. Therefore, the subsequence contains an occurrence of t, and so does the infinite occurrence sequence. □

So a very simple algorithm constructs the reachability graph and checks whether every cycle of this graph contains at least one arc labeled by t. A more elegant solution is to first delete all t-labeled arcs of the reachability graph (which does *not* necessarily lead to a connected graph) and then check for cycles.

4 Termination After Stopping a Transition – The Unbounded Case

Now we consider the case that the considered net is unbounded. Does it eventually terminate, provided a given transition t occurs only finitely often? For unbounded nets, the reachability graph is infinite, but the coverability graph is finite. However, unfortunately the coverability graph does not bring immediate help. Consider the simple example of a net with only one initially unmarked place, a single input transition i, and a single output transition o, as shown in Figure 1.

In this example, transition i eventually stops the net, whereas transition o does not. However, both transitions occur in the coverability graph in quite the same way, namely as labels of arcs leading from and to the vertex labeled by $[\omega]$. These are the only cycles of this coverability graph. While the coverability

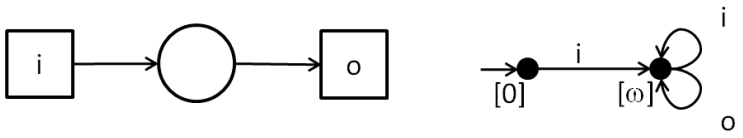


Fig. 1. A simple net and its coverability graph

graph does thus not lead to an algorithmic solution, we can solve the problem considering additional information, as shown below.

Remember that, during the (nondeterministic) construction of the coverability graph, we compare new ω-markings with already constructed ω-markings. When a new vertex of the coverability graph is constructed, the algorithm compares the ω-marking m corresponding to this new vertex with the ω-markings m' corresponding to vertices which are on paths from the initial vertex (representing the initial marking) to the new one, according to the graph constructed so far. If, for all places, the new ω-marking m is identical to m', then the new vertex is identified with the vertex corresponding to m'. Otherwise, if $m(s) \geq m'(s)$ for each place s (where $\omega > n$ for every integer n), then m is modified as follows: For each place s with $m(s) > m'(s)$, we set $m(s) := \omega$, because the sequence from the vertex corresponding to m' to the newly constructed vertex can be repeated arbitrarily often, leading to an unbounded token growth on the place s. For all other places s, $m(s)$ remains unchanged.

In the example of Figure 1, the marking reached by the occurrence of transition i is greater than the initial marking for the only place of the net. Hence, this place receives an ω-entry for the corresponding ω-marking $[\omega]$, represented by the vertex $[\omega]$ of the coverability graph. Further occurrences of transition i are possible, leading to the same ω-marking, because ω already means "arbitrarily many". Observe that transition i of the net can occur infinitely often, no matter if transition o occurs, whereas o cannot occur arbitrarily often without i, and in particular there is no infinite occurrence sequence $o\,o\,o\ldots$ enabled at any marking, a fact which is not reflected by the coverability graph.

In general, we are looking for infinite occurrence sequences, enabled by some reachable marking, without occurrences of t. Since occurrence sequences correspond to paths of the coverability graph and sufficiently long occurrence sequences have to pass through some vertex more than once, we have a closer look to closed paths of coverability graphs in the sequel.

Since ω-entries are only added during the construction of the coverability graph, and are never removed, all ω-markings appearing as vertices in a closed path of the coverability graph agree on the set of ω-marked places, whereas the non-negative integers assigned to the other places still represent the token game. Therefore, cycles and closed paths of coverability graphs do not necessarily correspond to cyclic behavior, because according occurrence sequences might increase or decrease the token count of places which have ω-entries in ω-markings of the path. If the token count of each place is not decreased, then the path can

be repeated arbitrarily, leading to an infinite occurrence sequence. Otherwise, it can not.

Consider a path π of the coverability graph of a net and let σ_π be the sequence of labels of arcs of π. We call the path π *non-decreasing*, if, for each place s, the number of occurrences of transitions in the pre-set of s in σ_π is not smaller than the number of occurrences of transitions in the post-set of s in σ_π, i.e., if at least as many tokens are added as are removed, and hence the *effect* of σ_π is non-negative for each place. Otherwise, the effect of transition occurrences in σ_π is negative for some place, and then we call π *decreasing*.

Proposition 2. *Given an unbounded net N and a coverability graph of N, a transition t stops N eventually if and only if there is no non-decreasing closed path of the coverability graph without an arc labeled by t.*

Proof. Assume that the coverability graph has a non-decreasing closed path π without an arc labeled by t. Let m_π be the ω-marking corresponding to the source vertex of the first arc of π. It is well-known that all ω-marked places of m_π are simultaneously unbounded, i.e., for each number b there is a reachable marking m_b of N satisfying $m_b(s) \geq b$ if $m_\pi(s) = \omega$ and $m_b(s) = m_\pi(s)$ if $m_\pi(s) \neq \omega$. Now, choosing b as the length of π, the sequence of labels of π is an occurrence sequence σ_π enabled at m_b. This follows from the construction rule of the coverability graph, which considers places not marked by ω as in the reachability graph construction. Places marked by ω carry sufficiently many tokens to allow the occurrences of all transitions in the sequence σ_π. Since π is assumed to be non-decreasing, the marking m' reached by σ_π satisfies $m'(s) \geq m_b(s)$ for each place s. Therefore, σ_π can be repeated arbitrarily. Thus, there is an infinite sequence without occurrences of t, enabled at the reachable marking m_b.

For the converse direction, consider an infinite occurrence sequence σ enabled at the initial marking of N with only finitely many occurrences of transition t. Let $\sigma = \sigma_1\,\sigma_2$ such that σ_2 contains no occurrences of t and σ_1 is minimal with this property (i.e., σ_1 is empty or ends with t). Let $\sigma_2 = t_1\,t_2\,t_3\ldots$ and let, for $i \geq 0$, m_i be the marking reached by the sequence $\sigma_1\,t_1\ldots t_i$. By repeated application of Dickson's Lemma, we find indices $k_1, k_2, k_3, \ldots$ such that, for $j > i$, $m_{k_j}(s) \geq m_{k_i}(s)$ for each place s. By the construction of the coverability graph, for each finite occurrence sequence enabled at the inital marking, there is unique path from the initial vertex such that the sequence of labels of its arcs equals the occurrence sequence. Since the coverability graph is finite, the vertices reached by the paths corresponding to the sequences $\sigma_1\,t_1\ldots t_{k_i}$ $(i > 0)$ cannot be pairwise different, whence some vertex is visited at least twice. Assume this is the case for the sequences $\sigma_1\,t_1\ldots t_{k_n}$ and $\sigma_1\,t_1\ldots t_{k_m}$, where $n < m$. Then the subsequence $t_{k_n+1}\ldots t_{k_m}$ corresponds to a closed path of the coverability graph, which does not contain an arc labeled by t. By construction, this path is non-decreasing. □

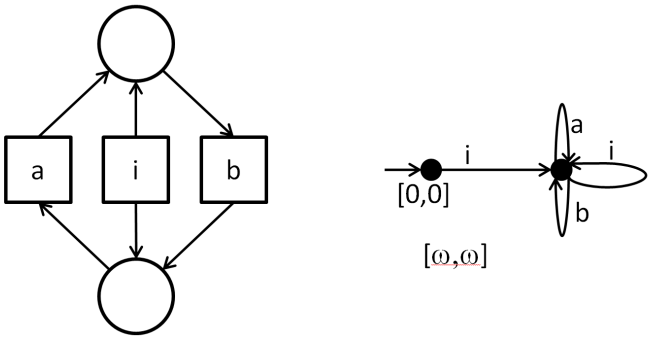


Fig. 2. Another simple net and its coverability graph

Figure 2 illustrates that the proposition does not hold when cycles instead of closed paths are considered.[2] The net in this figure is not stopped eventually by transition i. The only cycles without occurrences of i are the short cycles labeled by a and b, respectively. Both cycles are decreasing paths, whereas the closed path with arc labels a and b is non-decreasing (and so are all closed paths starting at the vertex $[\omega, \omega]$ with the same number of a-occurrences and b-occurrences).

Proposition 2 provides a characterization of stopping transitions based on the coverability graph. Unfortunately, this characterization is based on closed paths of a coverability graph, but there are infinitely many closed paths in general. Therefore, this characterization does not immediately lead to a deciding algorithm.

5 An Algorithm Deciding Whether a Transition Stops an Unbounded Net Eventually

Throughout this section, let N be an unbounded net and let t be a transition of N. We refer to the characterization given in Proposition 2 and collect properties of a non-decreasing closed path π without an arc labeled by t of a coverability graph of N.

(1) The subgraph of the coverability graph constituted by the arcs of π and the vertices occurring as sources or targets in these arcs is strongly connected.

Connectedness of the subgraph is obvious. The subgraph is even strongly connected because π is a closed path.

(2) For each vertex v of the coverability graph, the number of arcs occurring in π which have v as the source vertex equals the number of arcs in π which have v as the target vertex (the same arc can occur more than once in π,

[2] This was pointed out by an anonymous reviewer, thanks a lot!

and is in this case also counted more than once).

A simple observation, because π is a closed path.

(3) All ω-markings appearing in the path π (as sources or targets of arcs) have the same set of places marked by ω. In particular, this holds for the source and target vertices of each arc in π.

If a place is marked by ω in an ω-marking, then this place is also marked by ω in a subsequent marking in the coverability graph, by the construction rule of coverability graphs. The claim follows because π is a closed path.

(4) For each place s marked by ω in the ω-markings of the path π, the number of occurrences of transitions in arcs of π that belong to the pre-set of s is not smaller than the number of occurrences of transitions in arcs of π that belong to the post-set of s.

For places s of N which are not marked by ω, the number of occurrences of transitions in arcs of π that belong to the pre-set of s equals the number of occurrences of transitions in arcs of π that belong to the post-set of s by the construction rule of coverability graphs. For places marked by ω, the claim follows because π is non-decreasing by assumption.

(5) No arc in π is labeled by t.

This is part of the assumption.

So we obtain as an immediate consequence of Proposition 2:

Proposition 3. *If t does not stop the net N eventually, then conditions (1) to (5) are fulfilled for some path π of a coverability graph of N.* □

All the above conditions (1) to (5) can be viewed as properties of a multi-set of arcs of the coverability graph, which tells how often (and if at all) an arc occurs in a suitable path. The following proposition states that the properties are not only necessary but also sufficient for the existence of a non-decreasing closed path without occurrences of t.

Proposition 4. *Assume a coverability graph of N with arcs $\{a_1, \ldots, a_k\}$, and a mapping $f\colon \{a_1, \ldots, a_k\} \to \{0, 1, 2, \ldots\}$ (a multiset of arcs) satisfying $f(a_i) > 0$ for at least one arc and moreover the following conditions:*

(1) The subgraph of the coverability graph constituted by the arcs a_i satisfying $f(a_i) > 0$ and by the vertices occurring as sources or targets in these arcs is strongly connected.

(2) For each vertex v of the coverability graph, the sum of all $f(a_i)$ with v being the source vertex of a_i equals the sum of all $f(a_j)$ with v being the target vertex of a_j.

(3) For each arc a_i with the property that source and target ω-markings of a_i have different sets of ω-marked places, we have $f(a_i) = 0$.

(4) For each place s, the sum of all $f(a_i)$ where a_i is labeled by a transition in the pre-set of s is not smaller than the sum of all $f(a_j)$ where a_j is labeled by a transition in the post-set of s.

(5) If $f(a_i) > 0$ then a_i is not labeled by t.

Then there is a non-decreasing closed path π without arcs labaled by t.

Proof. We show that there exists such a path π such that, for each arc a_i, this arc occurs $f(a_i)$ times in π.

First, we construct the following sub-graph of the considered coverabiliy graph: We delete all arcs a_i of the coverablitiy graph satisfying $f(a_i) = 0$, and then delete all isolated vertices. Since at least one arc a_i satisfies $f(a_i) > 0$, this subgraph has at least one arc and hence at least one vertex. By condition (1), it is strongly connected.

It is a well-known theorem that a directed multigraph has an Euler Circuit (which is a closed path in our terminology), if it is connected and every vertex has the same in- and out-degree. If the multiplicity of arcs of the subgraph is given by the mapping f, we obtain a directed multigraph. Then the in-degree of a vertex v of the subgraph is the sum of all $f(a_i)$ for arcs a_i with target vertex v, and its out-degree is the sum of all $f(a_i)$ for arcs a_i with source vertex v. So, by condition (2), the above theorem can be applied to the subgraph. It proves that there exists a closed path π such that, for each arc a_i, $f(a_i)$ provides the number of occurrences of a_i in π. By conditions (3) and (4), π is non-decreasing. By condition (5), no arc of π is labeled by t. □

Hence, for deciding if transition t eventually stops the net N, it suffices to construct a coverability graph of N and check whether there exists no non-empty multiset of arcs satisfying the above conditions.

All conditions of the previous proposition except the first can be expressed in terms of inequalities. Let again $a_1, \ldots, a_k$ denote the arcs of a coverability graph of N. Given a path π of this coverability graph, the variables $x_1, \ldots, x_k$ indicate how often a particular arc appears in the path π. Using this notation, we rewrite the above conditions (the additional condition (6') just states that all x_i are non-negative):

(2') For a vertex v of the coverability graph, let $in(v)$ be the set of arcs with target v and let $out(v)$ be the set of arcs with source v.
For each vertex v,

$$\sum_{a_i \in in(v)} x_i = \sum_{a_i \in out(v)} x_i \ .$$

(3') For each arc a_i connecting two vertices representing ω-markings with different sets of ω-marked places, we have $x_i = 0$.

(4') For each place s with pre-set ${}^\bullet s$ and post-set $s^\bullet$ satisfying $m(s) = \omega$ in the ω-markings appearing in π, we have

$$\sum_{u \in {}^\bullet s} \sum_{\lambda(a_i)=u} x_i \;\geq\; \sum_{u \in s^\bullet} \sum_{\lambda(a_i)=u} x_i \;\;,$$

where $\lambda(a)$ denotes the label of arc a, i.e., the occurring transition.

(5') For each arc a_i labeled by t, we have $x_i = 0$.

(6') For $i = 1, \ldots, k$, we have $x_i \geq 0$.

It remains to find a solution $x_1, \ldots, x_k$ of the according homogeneous system of linear inequalities such that not all x_i are zero, which additionally satisfies condition (1), i.e., the multiset of arcs a_i constitutes a strongly connected subgraph of the coverability graph. Since the inequality system is homogeneous, we do not have to care about integral solutions, because for any (rational) solution there is a solution in the integers, derived by multiplication with the common denominator. However, since the number of solutions of the system of linear inequalities is potentially infinite, this still does not lead to a feasible algorithm.

Fortunately, all solutions of the inequality system can be represented as linear combinations (with non-negative coefficients) of a finite basis. See e.g. [1] for an algorithm to compute such a basis. Let $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_l\}$ be a basis, with $\mathbf{b}_i = [b_{i,1}, b_{i,2}, \ldots, b_{i,k}]$ for $1 \leq i \leq l$. Then each solution to the system of inequalities can be represented as

$$\mu_1 \cdot \mathbf{b}_1 + \mu_2 \cdot \mathbf{b}_2 + \cdots \mu_l \cdot \mathbf{b}_l \;\;,$$

where all μ_i are non-negative integers.

The following simple proposition shows that we do not have to consider all these (infinitely many) solutions, but may restrict on solutions where all coefficients belong to the set $\{0, 1\}$.

Proposition 5. *Let $\mu_1 \cdot \mathbf{b}_1 + \cdots + \mu_l \cdot \mathbf{b}_l$ be a solution of the system of inequalities (2') to (6'). Define, for $1 \leq i \leq l$, μ'_i by $\mu'_i := 0$ if $\mu_i = 0$, and $\mu'_i := 1$ if $\mu_i > 0$.*

Then $\mu'_1 \cdot + \mathbf{b}_1 + \mu'_2 \cdot \mathbf{b}_2 + \cdots \mu'_l \cdot \mathbf{b}_l$ is a solution of the system of inequalities (2') to (6'), too, and the subgraphs generated by both solutions conincide. In particular, the subgraph generated by the first solution is strongly connected if and only if the subgraph generated by the second solution is strongly connected.

□

Combining Propositions 2,3,4 and 5 yields the main result of this contribution:

Theorem 1. *Transition t does not stop the net N eventually if and only if, for any coverability graph of N and for any basis $\mathbf{B}$ of the solutions of the system of inequalities* (2') *to* (6')*, the sum of all solutions of a nonempty subset of $\mathbf{B}$ generates a connected subgraph of the coverability graph.* □

An algorithm can directly be derived from this theorem. The worst case complexity of this algorithm is apparently quite poor, because it requires the construction of the coverability graph, the construction of a basis of the derived system of inequalities, and finally it requires to consider all (exponentially many) subsets of these basis solutions.

6 A Faster Algorithm for Finding Suitable Subsets of Basis Solutions

Instead of considering all subsets of basis solutions to find a set of solutions generating a strongly connected subgraph of the coverability graph, such a set can be found by means of the following efficient divide-and-conquer algorithm:

We define an algorithmic function which, given a set $\mathbf{S}$ of solutions of the system of inequalities, first constructs the generated subgraph of the coverability graph, i.e., this graph has all arcs a_i of the coverability graph which have positive coefficients in any solution of $\mathbf{S}$. If this subgraph is strongly connected, we are finished and conclude that the considered transition t eventually stops the net. Otherwise this subgraph has more than one strongly connected component. For each strongly connected component, we consider the subset of solutions $\mathbf{S}' \subset \mathbf{S}$ with the property that all its positive coefficients refer to arcs of this component. If this set $\mathbf{S}'$ is not empty for a strongly connected component, it again generates a subgraph of the coverability graph. This subgraph is entirely located within the considered strongly connected component, but it is not necessarily strongly connected itself. We recursively apply this function, for each strongly connected component with nonempty according set $\mathbf{S}'$, to this set $\mathbf{S}'$. If the set $\mathbf{S}'$ is empty for all strongly connected components, the algorithm returns to its calling instance. If the algorithm terminates without finding a strongly connected subgraph generated by a set of solutions, it concludes that no such set exists and that therefore transition t eventually stops the net.

Initially, the function is applied to a basis $\mathbf{B}$ of solutions to the system of inequalities.

Proposition 6. *The algorithm terminates and outputs the correct answer. It runs in linear time with respect to the size of the basis $\mathbf{B}$.*

Proof. The algorithm terminates because the function is only called recursively for a set $\mathbf{S}'$ if the current set $\mathbf{S}$ does not generate a strongly connected graph and $\mathbf{S}'$ generates a smaller strongly connected subgraph.

The algorithm only stops before proper termination if it found a stronly connected subgraph generated by a solution, and hence the output that transition t eventually stops the net is correct.

It remains to show that, if the algorithm reaches its proper end and hence did not find a set generating a strongly connected subgraph, then no such set exists. So assume a set $\mathbf{S} \subseteq \mathbf{B}$ of solutions exists such that the generated subgraph is strongly connected. Now assume that $\mathbf{S} \subset \mathbf{S}'$. Then, obviously the subgraph generated by $\mathbf{S}$ is still in one strongly connected component of the subgraph generated by $\mathbf{S}'$. Therefore, whenever the function is called for some set $\mathbf{S}'$ satisfying $\mathbf{S} \subset \mathbf{S}'$, and from this instance it is called for subsets $\mathbf{S}_1, \mathbf{S}_2, \ldots$, then the set $\mathbf{S}$ is included in one of the sets $\mathbf{S}_i$. Since $\mathbf{S}$ is included in $\mathbf{B}$, it will eventually be found by the algorithm (unless another suitable set of solutions is found before).

Finally, the algorithm runs in linear time with respect to the size of the basis $\mathbf{B}$ because each function call performs a proper split of the set $\mathbf{B}$, and $\mathbf{B}$ can be splitted at most $|\mathbf{B}| - \mathbf{1}$ times. □

7 Conclusion

We have shown how to decide if a single transition is able to stop an entire net eventually, i.e., if no infinite occurrence sequences has only finitely many occurrences of t. The approach can easily be generalized to sets of transitions (if we stop all transitions of this set at some marking, will the net eventually terminate?). Another generalization refers to arc weights; the procedure works for nets with arc weights with only small changes. The usual complement place construction makes the approach also available for nets with capacity restrictions.

Experimental results and comparisons to other approaches, in particular to model checking algorithms for temporal logics, will be topics of further research.

Another tool for identifying transitions that stop a net is given by transition invariants, which are closely related to cyclic occurrence sequences, and by transition sur-invariants, which are related to occurrence sequence with non-negative effect to all places. Both types of invariants can be derived by linear algebraic means, see e.g. [2]. These techniques lead to much more efficient algorithms, but unfortunately provide only sufficient criteria for termination problems.

Yet another approach to solve the problem is to consider cycles in coverability graphs (see [3]), representing cyclic behavior. The calculation of such cycles requires, however, by far more effort than the algorithms suggested in the present contribution.

References

1. Dimitri Chubarov and Andrei Voronkov. Basis of solutions for a system of linear inequalities in integers: Computation and applications. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005, Gdansk, Poland, August 29 -*

September 2, 2005, Proceedings, volume 3618 of *Lecture Notes in Computer Science*, pages 260–270. Springer, 2005.
2. Jörg Desel. Basic linear algebraic techniques for place/transition nets. In Reisig and Rozenberg [8], pages 257–308.
3. Jörg Desel. On cyclic behaviour of unbounded Petri nets. In Josep Carmona, Mihai T. Lazarescu, and Marta Pietkiewicz-Koutny, editors, *13th International Conference on Application of Concurrency to System Design, ACSD 2013, Barcelona, Spain, 8-10 July, 2013*, pages 110–119. IEEE Computer Society, 2013.
4. Jörg Desel and Wolfgang Reisig. Place/transition Petri nets. In Reisig and Rozenberg [8], pages 122–173.
5. Javier Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Inf.*, 34(2):85–107, 1997.
6. Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.
7. Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.
8. Wolfgang Reisig and Grzegorz Rozenberg, editors. *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, based on the Advanced Course on Petri Nets, Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*. Springer, 1998.

Two Operations for Stable Structures of Elementary Regions

Federica Adobbati[1], Carlo Ferigato[3], Stefano Gandelli[1], and Adrián Puerto Aubel[2]

[1] DISCo — Università degli Studi di Milano-Bicocca
[2] INRIA - Rennes Bretagne-Atlantique, IRISA, Université de Rennes I
[3] JRC — Joint Research Centre of the European Commission

Abstract. The set of *regions* of a *condition/event transition system* represents all the possible local states of a *net system* the behaviour of which is specified by the transition system. This set can be endowed with a structure, so as to form an *orthomodular partial order*. Given such a structure, one can then define another condition/event transition system. We study cases in which this second transition system has the same collection of regions as the first one. When it is so, the structure of regions is called stable. We propose, to this aim, a composition operation, and a refinement operation for stable orthomodular partial orders, the results of which are stable.

1 Introduction

This work studies the interrelations between local states, locally observable properties, and events of distributed systems. To this aim, its framework is set in the relation between elementary Petri net systems, and the labelled transition systems expressing their behaviour, their case graphs. Indeed, labelled transition systems are commonly used models for verification of properties of the specified system [8].

We focus on the particular case in which local states act like Boolean variables, forbidding executions by carrying true or false values. Thus, the framework is narrowed down to elementary and condition/event models, and their close relation to the theory of regions. In [10, 11], regions were shown to be the key to solving the synthesis problem: Given a labelled transition system, determine whether it is the case graph of some elementary net system. The extension of a local state of a net system in its case graph is the set of global states at which it holds the value true. In a labelled transition system, a region is a set of global states having a consistent incidence with respect to the actions, so as to be the extension of a local state. Given an elementary, or condition/event transition system, a net system can be constructed with regions as local states, such that its case graph will be precisely the specified transition system. It was further shown [9] that given an elementary (condition/event) system, it suffices to consider the sub-collection of regions that is minimal with respect to set inclusion

[1]. The remaining regions correspond to local states which are redundant in the net system.

Regions, as subsets of global states of a system, carry an algebraic structure [2]. They can be ordered by set inclusion, in a relation that expresses implication. Negation, disjunction, and conjunction can be defined as set operations, only partially in the latter two cases. The resulting structure was shown to be a prime and coherent orthomodular partial order [2], OMP for short.

Orthomodular partial orders typically represent, in an algebraic fashion, systems about which the acquisition of information depends on the point of view, or experiment. They were first introduced by Birkhoff and Von Neumann [6] in order to formalise the logic of properties of quantum systems. Foulis and Randall [12, 18] extended their approach to provide a generalized model.

In the view that OMP's are a suitable specification for the observable properties of a system, we are concerned with a second synthesis problem: Given an OMP determine whether it is the regional structure of some elementary transition system. This problem was first posed in [2], and a solution was pointed at. When an OMP is prime, one can make use of a representation theorem to find a set of states such that the elements of the OMP are subsets of it. By considering a complete graph whose vertices are these states, one can label each edge so that all elements of the OMP can be found as regions of this new transition system. When the structure of regions of this new system is isomorphic to the specified OMP, then the latter is called *stable*.

Not every OMP is stable, and the full characterization of the class of stable OMP's is an open problem which represents a long term objective. It is conjectured that every regional OMP, the structured collection of regions of some elementary transition system, is stable. The present work includes itself in a series of papers [2–5] the scope of which is to study this stability of OMP's, in an attempt to prove the conjecture.

The present work extends [4], in which some classes of stable OMP's were presented. Here, composition and refinement operations are defined, which preserve stability, thus proving the stability of regional OMP's for a wide range of systems.

2 Background

This work implicitly deals with elementary net systems (ENS for short) [15]. Even though these models will not be formalised here, they underlie the main ideas in the presented results. ENS are a class of pure, and simple Petri net systems with a particular firing rule which ensures that the system remains 1-safe in case of contact. *Conditions* of ENS can only be in one of two states, marked or un-marked, and as such they can be seen as Boolean variables. Moreover, out of Petri's extensionality principle, two events having the same incidence on the set of conditions, and viceversa for conditions in relation to events, should be considered the same [17].

Condition/Event net systems (CENS for short) are structurally identical to ENS with the addition of a backward firing rule which provides symmetry to the reachability relation amongst markings; reachability becomes an equivalence relation, and reachability classes substitute the initial marking required for ENS.

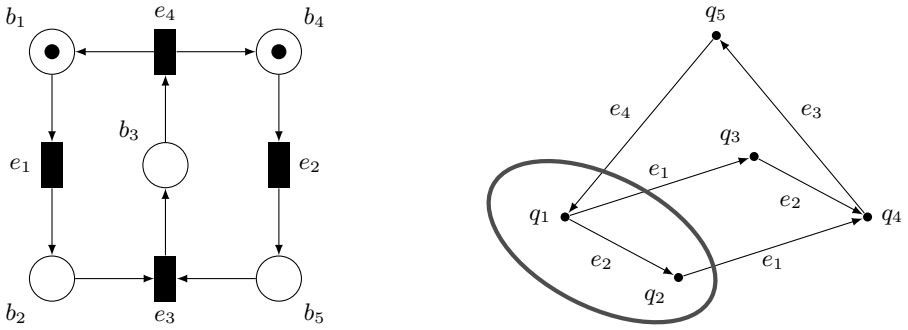


Fig. 1. To the left, a CENS. To the right, its sequential case graph. The state q_1 corresponds to the marking $\{b_1, b_4\}$, at which e_1 and e_2 are concurrently enabled. To the right, the subset of states $\{q_1, q_2\}$ is in evidence; it corresponds to the extension of b_1, namely the set of markings containing b_1.

2.1 Transition Systems and Regions

The behaviour of CENS can be expressed by means of a labelled transition system. In such a model, states correspond to the reachable markings of the net system. A marking enabling an event will correspond to a state from which an arrow, labelled with this event, points to the state which is reached when the event is fired. Such a labelled transition system is the *sequential case graph* of the underlying net system.

Definition 1 (Transition System). *A* transition system *is a structure* $A = (Q, E, T)$*, where* Q *is a set of states,* E *is a set of* events *and* $T \subseteq Q \times E \times Q$ *is a set of* transitions *carrying labels in* E*, such that:*

1. *the underlying graph of the* transition system *is connected;*
2. $\forall (q_1, e, q_2) \in T \quad q_1 \neq q_2$*;*
3. $\forall (q, e_1, q_1)(q, e_2, q_2) \in T \quad q_1 = q_2 \Rightarrow e_1 = e_2$*;*
4. $\forall e \in E \quad \exists\, (q_1, e, q_2) \in T$*.*

All the structures we consider are finite. A fundamental notion used in this contribution is the one of *region*. Regions were introduced in [10, 11] as the fundamental tool for solving the so called *synthesis problem* for Petri Nets.

Definition 2 (Region). *A* region *of a transition system* $A = (Q, E, T)$ *is a subset* r *of* Q *such that:* $\forall e \in E, \forall (q_1, e, q_2), (q_3, e, q_4) \in T$*:*

1. *($q_1 \in r$ and $q_2 \notin r$) implies ($q_3 \in r$ and $q_4 \notin r$)* and
2. *($q_1 \notin r$ and $q_2 \in r$) implies ($q_3 \notin r$ and $q_4 \in r$).*

Regions are subsets of states which have a consistent orientation with the labelling of transitions. As such, they can be assigned an incidence with respect to each event, and interpreted as Boolean conditions which hold at the states composing them.

Example 1. With reference to Figure 1, the extension of b_1 is a region. As a subset of states, all occurrences of e_1 exit it and e_4 enters it. All occurrences of the remaining events do not cross the border of b_1.

The set of regions of a transition system A will be denoted by $\mathcal{R}(A)$ and, given a state q of A, R_q will denote the subset of the regions in $\mathcal{R}(A)$ which contains the state q. Given a transition system A and its set of regions $\mathcal{R}(A)$, the *pre-set* and *post-set* operators can be defined:

Definition 3 (Pre- and Post-sets). *${}^\bullet(\,.\,)$, the* pre-set *operator, and $(\,.\,)^\bullet$, the* post-set *operator, are defined on the events E and on the regions $\mathcal{R}(A)$ of the transition system A as follows. Let r be a region in $\mathcal{R}(A)$*

1. ${}^\bullet r = \{e \in E \mid \exists\,(q_1, e, q_2) \in T \text{ such that } q_1 \notin r \text{ and } q_2 \in r\}$;
2. $r^\bullet = \{e \in E \mid \exists\,(q_1, e, q_2) \in T \text{ such that } q_1 \in r \text{ and } q_2 \notin r\}$;
3. ${}^\bullet e = \{r \in \mathcal{R}(A) \mid e \in r^\bullet\}$;
4. $e^\bullet = \{r \in \mathcal{R}(A) \mid e \in {}^\bullet r\}$.

We are concerned with a specific class of transition systems, the *condition/event transition systems* (CETS for short). CETS can be defined as those labelled transition systems which are the sequential case graph of some CENS. An alternative definition, in terms of regions and the *separation axioms*, follows from the results of [10, 11].

Definition 4 (CETS – Condition/Event Transition System). *A* CETS *is a transition system in which the following conditions hold:*

1. $\forall\, q_1, q_2 \in Q \quad R_{q_1} = R_{q_2} \Rightarrow q_1 = q_2$;
2. $\forall\, q_1 \in Q\ \forall\, e \in E \quad {}^\bullet e \subseteq R_{q_1} \Rightarrow \exists\, q_2 \in Q \text{ such that } (q_1, e, q_2) \in T$;
3. $\forall\, q_1 \in Q\ \forall\, e \in E \quad e^\bullet \subseteq R_{q_1} \Rightarrow \exists\, q_2 \in Q \text{ such that } (q_2, e, q_1) \in T$.

This definition requires of a transition system, that its set of regions is sufficient to fully determine the incidence of each event with respect to each state. Under these conditions, a CENS can be constructed, which has the set of regions for conditions, and the transition labels as events, and such that its case graph is isomorphic to the given transition system. This tight relation between CENS and CETS was thoroughly formalised in [15], in the equivalent framework of elementary systems.

The following basic properties of the regions $\mathcal{R}(A)$ of a CETS A are proven in [1]:

1. $\emptyset, Q \in \mathcal{R}(A)$;
2. $r \in \mathcal{R}(A) \;\Rightarrow\; Q \setminus r \in \mathcal{R}(A)$;
3. $r_1, r_2 \in \mathcal{R}(A) \;\Rightarrow\; (r_1 \cap r_2 \in \mathcal{R}(A) \Leftrightarrow r_1 \cup r_2 \in \mathcal{R}(A))$.

2.2 Orthomodular Partially Ordered Sets

Orthomodular partially ordered sets, indicated as OMP in what follows, are well known algebraic structures in the literature on *Quantum Logics*. The following definition can be found as the finite version of Definition 1.1.1 in [16]. Note that $\wedge$ and $\vee$ are the usual meet and join operations on a partial order, denoting, respectively, the *greatest lower bound*, and the *least upper bound*.

Definition 5 (OMP - Orthomodular Partially Ordered Set). *An* orthomodular partially ordered set $\langle L, \leq, (.)', 0, 1\rangle$ *is a set L endowed with a partial order $\leq$ and a unary operation $(\,.\,)'$ (called orthocomplement), such that the following conditions are satisfied:*

1. *L has a least and a greatest element (respectively 0 and 1) and $0 \neq 1$;*
2. *$\forall\, x, y \in L \quad x \leq y \Rightarrow y' \leq x'$;*
3. *$\forall\, x \in L \quad (x')' = x$;*
4. *for any finite sequence $\{x_i \,|\, i \in I\}$ of elements of L such that $i \neq j \;\Rightarrow x_i \leq x'_j$, then $\bigvee_{i\in I} x_i$ exists in L;*
5. *$\forall\, x, y \in L \quad x \leq y \;\Rightarrow\; y = x \vee (x' \wedge y)$.*

This latter condition is sometimes referred to as orthomodular law*.*

We will often denote $\langle L, \leq, (.)', 0, 1\rangle$ simply by L, and assume that an OMP L_i has underlying structure $\langle L_i, \leq_i, (.)', 0_i, 1_i\rangle$.

In general, $\wedge$ and $\vee$ operations can be undefined for some pairs of elements of L. Two elements x and y in L are said to be *orthogonal*, noted by $x \perp y$, when $x \leq y'$. As a consequence of their basic properties, as listed at the end of Section 2.1 above, the set of regions of a CETS is an OMP. 0 and 1 are, respectively, $\emptyset$ and Q, the order is given by set containment and orthocomplement by set complement; moreover, two regions are *orthogonal* whenever their intersection is empty.

Since all the structures we consider are finite, an OMP can be equivalently specified by its collection of *atoms* $\mathcal{A}(L) := \{x \in L \setminus \{0\} \mid \forall y \in L : (y \leq x) \rightarrow (y = x \text{ or } y = 0)\}$ together with their orthogonality relation. Each element of L can then be retrieved as the join of a collection of pairwise orthogonal atoms. The reader is referred to [5] for the details, and to [12, 18] for a full characterisation of this representation of OMP's. A subset M of an OMP L with the order relation and the orthocomplement operation inherited from L is a sub-OMP of L in case M is an OMP ([16], definition 1.2.2).

Definition 6 (OMP-morphism). *([16], finite case of definition 1.2.7) Let L_1 and L_2 be* OMP*. A mapping $f : L_1 \rightarrow L_2$ is a* morphism of OMP's *if the following conditions are satisfied:*

1. *$f(0) = 0$;*
2. *$\forall x \in L_1\; f(x') = f(x)'$;*
3. *for any finite sequence $\{x_i \,|\, i \in I\}$ of mutually orthogonal elements in L_1,*

$$f(\bigvee_{i\in I} x_i) = \bigvee_{i\in I} f(x_i)$$

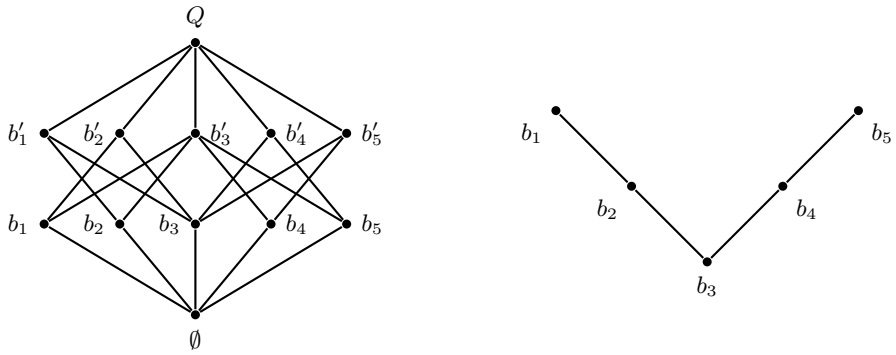


Fig. 2. Two representations of the OMP obtained by ordering the regions of the transition system in Figure 1. To the left, all regions are represented in a *Hasse diagram*. To the right, only the atoms are represented; each of the two solid lines is a maximal clique of orthogonality. This last representation is called *Greechie diagram*.

A morphism $f : L_1 \to L_2$ *is an* isomorphism *if* f *is injective, maps* L_1 *onto* L_2 *and* f^{-1} *is a morphism. An isomorphism* $f : L_1 \to M$*, when* M *is a sub-*OMP *of* L_2*, is called* embedding.

Morphisms of OMP's preserve *compatibility*, order and orthogonality [2]. Two elements x, y of an OMP L are said to be *compatible* if they admit a common orthogonal base. Formally, $x \$ y \Leftrightarrow \exists a, b, c \in L : (a \perp b \perp c \perp a)$ and $(x = a \vee b$ and $y = b \vee c)$ ([16], definition 1.2.1). This orthogonal base generates a Boolean algebra which contains both elements, and is a sub-OMP of L. When considered as conditions of a net system, two compatible elements will induce sequential behaviour of their neighbouring events, imposed by the orthogonal base. Indeed, orthogonality is to be interpreted as mutual exclusion of the corresponding conditions, and in fact, when restricted to atoms, the compatibility relation reduces to orthogonality [5]. We say two elements are incompatible, denoted $x \not\$ y$, when they are not compatible. Compatible elements x and y will be denoted by $x \$ y$. An embedding that preserves and reflects compatibility is called *strong* ([4], Definition 15).

A *two-valued state* of an OMP is a well-known concept in the literature on *Quantum Logics* ([16], Chapter 2). We will use here a specific definition of *two-valued state*, *state* for short, apt to the OMP's of the *regions* of CETS.

Definition 7 (State of an OMP). *([2], theorem 41) Let* $\mathcal{A}(L)$ *be the set of* atoms *of an* OMP L*. Let* E *be the set of the* maximal cliques *of* $\perp$ *in* L *restricted to* $\mathcal{A}(L)$*; let* C *be the set of the maximal cliques of* $\not\$$ *in* L *restricted to* $\mathcal{A}(L)$*; let* $S = \{s \in C \mid \forall e \in E \ |s \cap e| = 1\}$*, then the up-closure of* s*,* $\uparrow(s)$*, in* L *is a* state *in* L*.*

Since *states* of L are uniquely defined by means of up-closures of maximal cliques of $\not\$$, we will frequently refer to these maximal cliques as the *states* of the OMP L when clear from the context.

The collection of all states of an OMP L will be denoted by $S(L)$. It will also be useful to consider the collection of states which contain a given element. Given an element $x \in L$, let us define the notation $S_x := \{s \in S(L) \mid x \in s\}$.

The following section is concerned with the representation of an OMP as a collection of subsets of its states. The order in L will then be set inclusion, and the orthocomplement will simply be given by set complement. When this is the case, the OMP is said to admit a concrete representation.

2.3 Properties of Regional OMP's

As explained in more detail in section 2.4 below, OMPs can be constructed from the regions of a CETS. These OMPs will, from now on, be referred to as *regional* OMP*s*. They have been shown, in [2], to satisfy the following properties. We shall not get in the details, and the reader is referred to [2] for a full explanation of these, and the implications they provide in the interpretation of OMPs for system specification.

- *Regularity:* L is regular whenever every set of pairwise compatible elements is a compatible set [16]
- *Richness:* L is rich whenever it admits a concrete representation [16]

There is some inconsistency in the terminology referring to these concepts in the literature, and it has led to some confusion in the axiomatic definition of these properties. Indeed *regularity* is the term used in [16], whereas [14] refers to this property as *coherence*. The concept of *coherence* is however slightly more general in [12, 18].

Richness is axiomatically defined in [16], and assumed through [2–5] to be equivalent to the notion of *primeness* defined in [14]. The two definitions however, differ enough to raise a doubt, as their equivalence might not seem straightforward. This equivalence is, in what follows, proven, for the sake of mathematical rigour.

An OMP L is called unital when for every element $x \in L \setminus \{0\}$ there is a state $s \in S(L)$ such that $x \in s$.

Definition 8 (Rich OMP). *[16] An* OMP L *is rich iff*
$$\forall x, y \in L : S_x \subseteq S_y \Rightarrow x \leq y$$

In Chapter 2 of [16], a few equivalent characterisations of richness are shown. In particular we are interested in the following:

Theorem 1. *[16] An* OMP L *is rich iff it is unital and*
$$\forall x, y \in L : x \not\perp y \Rightarrow \exists s \in S(L) : x \in s \text{ and } y \in s$$

Rich OMP's where shown in Chapter 2 of [16] to also be unital.

Definition 9 (Prime OMP). *[14] An* OMP L *is prime iff*
$$\forall x, y \in L : x \neq y \Rightarrow \exists s \in S(L) : x \in s \Leftrightarrow y \notin s$$

We now prove two lemmas that will be used in the proof of Theorem 2 below:

Lemma 1. *In an* OMP L *we have that:*
$\forall x, y \in L : x \not\$ y \Rightarrow \exists s \in S(L) : x \notin s$ *and* $y \notin s$ *if and only if:*
$\forall x, y \in L : x \not\$ y \Rightarrow \exists s \in S(L) : x \in s$ *and* $y \in s$

Proof. Suppose that L is an OMP such that $\forall x, y \in L : x \not\$ y \Rightarrow \exists s \in S(L) : x \in s$ and $y \in s$ and consider two elements $x, y \in L : x \not\$ y$; then it is also true that $x' \not\$ y'$. By hypothesis we have that $\exists s \in S(L) : x' \in s$ and $y' \in s$ and neither x nor y belong to s, which means that $\exists s \in S(L) : x \notin s$ and $y \notin s$ for every pair of incompatible elements in L.

Conversely, suppose that L is an OMP such that $\forall x, y \in L : x \not\$ y \Rightarrow \exists s \in S(L) : x \notin s$ and $y \notin s$ and consider two elements $x, y \in L : x \not\$ y$; again it is true that $x' \not\$ y'$. By hypothesis we have that $\exists s \in S(L) : x' \notin s$ and $y' \notin s$ and both x and y belong to s, which means that $\exists s \in S(L) : x \in s$ and $y \in s$ for every pair of incompatible elements in L.

Lemma 2. *A prime* OMP L *is also unital.*

Proof. For every element $x \in L$ distinct from 0 we have that $\exists s \in S(L) : x \in s \Leftrightarrow 0 \notin s$ and since 0 doesn't belong to any state of L then x belongs to s.

Theorem 2. *An* OMP L *is rich iff it is prime.*

Proof. Suppose that L is rich and consider two elements $x, y \in L : x \neq y$. There are two cases:

$x \$ y$: in this case there exists a Boolean subalgebra of L that contains both x and y. For both of them we calculate the set of atoms below them, $\mathcal{A}_{\downarrow}(x) = \{a \in \mathcal{A}(L) \mid a \leq x\}$ and $\mathcal{A}_{\downarrow}(y) = \{a \in \mathcal{A}(L) \mid a \leq y\}$. Such sets differs in at least an element z, because otherwise it would mean, out of Axiom 4 in Definition 5, that $x = y$. If we consider a state $s \in S(L)$ such that $z \in s$ then s will contain only one element between x and y because z belongs to to only one of the two sets $\mathcal{A}_{\downarrow}(x)$ and $\mathcal{A}_{\downarrow}(y)$. The existence of s is ensured by the fact that L is unital, which means that $\exists s \in S(L) : x \in s \Leftrightarrow y \notin s$.

$x \not\$ y$: if $x \not\$ y$ then we also have that $x \not\$ y'$. From the hypothesis we have that $\exists s \in S(L) : x \in s$ and $y' \in s$ which means that $\exists s \in S(L) : x \in s \Leftrightarrow y \notin s$.

Now, suppose that L is prime, for Lemma 2 L is also unital. Consider two elements $x, y \in L : x \not\$ y$. That means that $x \not\$ y'$ and in particular $x \neq y'$. By Definition 9 of prime OMP we have that $\exists s \in S(L) : x \in s \Leftrightarrow y' \notin s$ which means that $\exists s \in S(L) : x \in s \Leftrightarrow y \in s$. Again, we consider two cases:

$x \in s$: then $\exists s \in S(L) : x \in s$ and $y \in s$

$x \notin s$: then $\exists s \in S(L) : x \notin s$ and $y \notin s$

which, as shown in Lemma 1, is equivalent to $\exists s' \in S(L) : x \in s'$ and $y \in s'$. □

2.4 Saturated Transition System and the Stability Problem

As anticipated in section 2.3 above, we know that, given a CETS $A = (S, E, T)$, $H(A) = (\mathcal{R}(A), \subseteq, \emptyset, S, (\,.\,)')$ with $(\,.\,)'$ the *set complement* operation, is an OMP. Moreover, this *regional* OMP is *regular* and *rich*.

We know as well that, given a *regular* and *rich* OMP L, $J(L) = (S(L), E(L), T(L))$ is a CETS; where $E(L)$ and $T(L)$, the sets of, respectively, *events* and *transitions*, are constructed by exploiting symmetric differences between states as defined in [2] and [5]. Formally, a state is a collection of elements of L, and each event $e \in E(L)$ will be of the form $e = [s, s'] = (s \setminus s', s' \setminus s)$, as such, it is characterised by its sets of pre-conditions, and post-conditions seen as elements of L. The underlying graph of $J(L)$ is complete, presenting a transition $(s, [s, s'], s')$ between each ordered pair of distinct states (s, s'), and is therefore called the *saturated transition system* of L.

In [5] it is shown how the construction of $J(L)$ can be done by considering exclusively the *atoms* of L. In particular, states are determined by their atoms, and so are events, as symmetric differences of the atoms of states. A natural question concerns the full axiomatic definition of the *regional* OMP*s*. We know that, in the general case, given a *rich* and *regular* logic L, L embeds in $H(J(L))$. We consequently define as *stable* an OMP L isomorphic to $H(J(L))$.

It is our long-term goal to fully characterise the class of stable OMP's, and it is conjectured that it coincides with the class of regional OMP's. In the next section, we will present a compositional approach which will allow us to prove the stability of a wide class of regional OMPs.

3 Composition of OMPs and their Stability

This section will start introducing a rather general composition operation for OMP's. The result of this composition will not always be an OMP but we show that it is the case in particular instances.

3.1 Composition of OMP's

We here present a composition operation for OMP's, in the fashion of those presented in [7] for modular ortholattices and orthomodular lattices. We focus on the class of orthomodular posets, of which orthomodular lattices are a subclass. Note that, whereas the cases treated in [7] are classes of algebras, our class is not, since in OMP's joins and meets are only partially defined.

Definition 10 (V-formation). *A* V-formation *of* OMP*'s is a tuple* $(I, L_1, L_2, \phi_1, \phi_2)$*, such that* I, L_1*, and* L_2 *are* OMP*'s, and* $\phi_i : I \to L_i (i = 1, 2)$ *are morphisms of* OMP*'s.*

A V-formation serves as specification for composing L_1 and L_2 on the common interface I. In categorical terms, it is simply a diagram in the category of OMP's. Strictly speaking, the interface would only be $\phi_1^{-1}(L_1) \cap \phi_2^{-1}(L_2)$, so in

order to simplify notation, we here consider V-formations in which $\phi_i (i = 1, 2)$ are embeddings, and so for each $i = 1, 2$, $\phi_i(I)$ is a sub-OMP of L_i isomorphic to I.

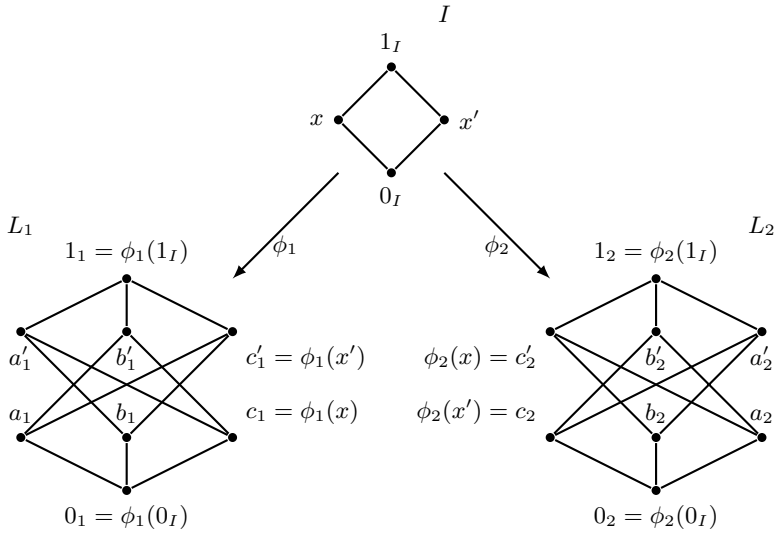


Fig. 3. A V-formation

Given a V-formation of OMP'embeddingss, we propose a straightforward composition operation. This construction is inspired by co-equalisers [13], however, universality of the construction remains an open problem. As a matter of fact, the nature of the proposed composition interprets the interface I as a sub-OMP of each component, so as to identify the two copies element-wise. In this sense, it requires the morphisms of the V-formation to be actual embeddings.

Definition 11 (Equivalence induced by a V-formation).
Let $V = (I, L_1, L_2, \phi_1, \phi_2)$ be a V-formation of OMP*'s. Consider $\tilde{L}$, the disjoint union of L_1 and L_2 such that $\phi_i : I \to \tilde{L}\,(i = 1, 2)$, with $\phi_1(I) \cap \phi_2(I) = \emptyset$. The* equivalence relation induced by *V is the binary relation $\sim_V := \{(x, x) \in \tilde{L}^2 \mid x \in \tilde{L}\} \cup \{(\phi_1(x), \phi_2(x)) \in \tilde{L}^2 \mid x \in I\} \cup \{(\phi_2(x), \phi_1(x)) \in \tilde{L}^2 \mid x \in I\}$.*

It is straightforward to verify that $\sim_V$ is an equivalence relation. It is reflexive, and symmetric by construction. If $x, y, z \in \tilde{L}$ are such that $x \sim_V y$ and $y \sim_V z$ then it must be either $z = x$ or $z = y$, so $\sim_V$ is transitive. We will denote the equivalence class of an element $x \in \tilde{L}$ by $[x]$. This equivalence relation satisfies these additional properties:

Proposition 1. *Let $V = (I, L_1, L_2, \phi_1, \phi_2)$ be a V-formation of* OMP *'s, and $\sim_V$ as in Definition 11. Then:*

1. $\forall i \in \{1,2\} : \forall x \in L_i : [x] \cap L_i = \{x\}$
2. $\forall x, y \in \tilde{L} : x \sim_v y \Leftrightarrow x' \sim_V y'$
3. $[0_1] = [0_2] = \{0_1, 0_2\}$ *and* $[1_1] = [1_2] = \{1_1, 1_2\}$
4. $\forall x, y \in \tilde{L} : (x \leq y) \Rightarrow \neg(\exists \tilde{x} \in [x], \tilde{y} \in [y] : \tilde{y} < \tilde{x})$.

Proof. 1. By construction.

2. Let $x, y \in \tilde{L}$ be two distinct elements such that $x \sim y$. From the definition of $\sim_V$, it follows that $\exists z \in I : \phi_1(z) = x$ and $\phi_2(z) = y$ (up to swapping of x and y). Since ϕ_1 and ϕ_2 are OMP-morphisms, it follows that $\phi_1(z') = x'$ and $\phi_2(z') = y'$, hence $x' \sim y'$.
3. It is a direct consequence of Axioms 1 and 2 in Definition 6.
4. Let $x \leq y$ then $\exists i \in \{1,2\} : x, y \in L_i$, and $x \leq_i y$. Analogously, $\exists j \in \{1,2\} : \tilde{x}, \tilde{y} \in L_j$, and $\tilde{y} <_j \tilde{x}$. Clearly it must be that $i \neq j$. Now, $x \sim_V \tilde{x}$, and $y \sim_V \tilde{y} \Rightarrow \exists a, b \in I : \phi_i(a) = x, \phi_j(a) = \tilde{x}, \phi_i(a) = y$, and $\phi_j(b) = \tilde{y}$. Since ϕ_i is an OMP-embedding, it must reflect the order, yielding $a \leq b$, but ϕ_j preserves the order, and so $\tilde{x} \leq \tilde{y}$.

□

These results allow for endowing the quotient $\tilde{L}/\sim_V$ with a structure.

Definition 12 (I-pasting of OMP's). *Consider the setting of Definition 11, and define:*

1. $L = \tilde{L}/\sim_V$,
2. $0 = [0_1] = [0_2]$ *and* $1 = [1_1] = [1_2]$,
3. $\forall [x] \in L : [x]' = [x']$,
4. $[x] \prec [y] \Leftrightarrow \exists x \in [x] : \exists y \in [y] : x \leq y$ *in* $\tilde{L}$, *and*
5. $\leq \subseteq L \times L$ *as the transitive closure of* $\prec$.

Then the I-pasting *of* L_1 *and* L_2 *induced by* V *is the structure* $L_1|I|L_2 = \langle L, \leq, (\cdot)', 0, 1\rangle$.

It follows immediately from Proposition 1 (4.) that $\leq$ is an order relation. Proposition 1 (2.), states that $\sim_V$ is congruence for complementation, and so $(\cdot)'$ is well-defined on L. It is furthermore trivial to verify that 0 and 1 are respectively the minimal and maximal elements in L. Note that whenever $x \perp y$, then $[x] \perp [y]$. So defined, the composition of two OMP's over an interface is simply obtained by identifying the elements whose pre-images through ϕ_1 and ϕ_2 coincide. In general, such an I-pasting will be an *orthocomplemented partial order* [14]. This is, however, not sufficient to guarantee that it is in fact an OMP. The following example shows that it can fail to be.

Example 2. With reference to Figure 3. Let $I = \{0, 1, x, x'\}$ be an OMP with $0 \leq x, x' \leq 1$. For $i = 1, 2$, let L_i be Boolean algebras with three atoms each $\{a_i, b_i, c_i\}$. Since ϕ_i are OMP-morphisms, $\phi_i(0) = 0_i$, and $\phi_i(1) = 1_i$. Now let $\phi_1(x) = c_1$, so that $\phi_1(x') = c_1' = a_1 \vee b_1$; and $\phi_2(x') = c_2$ so that $\phi_2(x) = c_2' = a_2 \vee b_2$. In this case, $\sim$ is the reflexive and symmetric closure of $\{(0_1, 0_2), (1_1, 1_2), (c_1, c_2'), (c_1', c_2)\}$, let $[x]$ denote its equivalence classes. In

$L_1|I|L_2$, we have that $[a_1] \leq [c_1'] = [c_2] \leq [a_2']$. Hence, $[a_1]$ and $[a_2]$ are orthogonal, but they have no least upper bound. Indeed, $[a_1], [a_2] \leq [b_1'], [b_2']$ and $[b_1'] \wedge [b_2'] = [0]$.

In what follows, we will study cases in which this composition is actually an OMP.

3.2 Extending a system with a sequential component

It is a known result [16] that whenever L_1 and L_2 are OMP's, and $I = \{0, 1\}$ is the trivial Boolean algebra, then $L_1|I|L_2$, as defined in the last subsection, is an OMP. It was further shown in [4] that in this case, if L_1 and L_2 are stable, then so must be $L_1|I|L_2$. This composition operation corresponds to the parallel composition of the corresponding operand systems. Indeed, the two systems are simply considered as a whole, although they remain independent, they do not synchronise or exchange information. Since the result of $L_1|I|L_2$ is stable, it can be itself an operand for a further composition, and so this composition can be iterated, in order to generate a wide class of stable systems. As an operation, it is associative and commutative.

We consider now the case in which the interface is a non-trivial Boolean algebra $I = \{0, 1, y, y'\}$ whose atoms are: $\mathcal{A}(I) = \{y, y'\}$. With such an interface, we impose that the corresponding saturated transition systems synchronise according to the specified embeddings. One of the components will be a finite Boolean algebra, and will be denoted B. Boolean algebras considered as OMP's were shown to be stable in [4]. The proof of stability will require the notion of *free atom*. An atom is *free* in an OMP if it belongs to just one maximal Boolean subalgebra. When seen as the region of a CETS, a free atom is a minimal region belonging to a single regional partition.

Example 3. With reference to Figure 2, b_1, b_2, b_4, b_5 and b_3 are all atoms, however b_1, b_2, b_4, b_5 are free, but b_3 is not. Indeed, b_3 belongs to two maximal Boolean algebras.

The considered embeddings will then identify a free atom of an OMP with an atom of B.

Theorem 3. *Let L be an OMP, and $x \in \mathcal{A}(L)$ be an atom. Let B be a finite Boolean algebra, and $a \in \mathcal{A}(B)$. Let $I = \{0, 1, y, y'\}$, and define $\phi_L : I \to L$, and $\phi_B : I \to B$ such that $\phi_L(y) = x$, and $\phi_B(y) = a$. Then $L|I|B$ induced by the V-formation $(I, L, B, \phi_L, \phi_B)$ is an OMP.*

Proof. After Proposition 1, it suffices to show that orthogonal joins are well defined, and that the orthomodular law holds. First note that in this case, the only identifications are $[0] = \{0_L, 0_B\}, [1] = \{1_L, 1_B\}, [x] = \{x, a\}$ and $[x'] = \{x', a'\}$, all other equivalence classes being singletons. Since both x and a are atoms, we have that $\leq = \prec$, in the setting of Definition 12. Furthermore, for each pair of orthogonal elements $[c] \perp [d]$, there must be $c \in L' \cap [c]$ and $d \in L' \cap [d]$, where $L' \in \{L, B\}$ such that $c \perp d$ in L'. If this holds for both $L' = L$ and

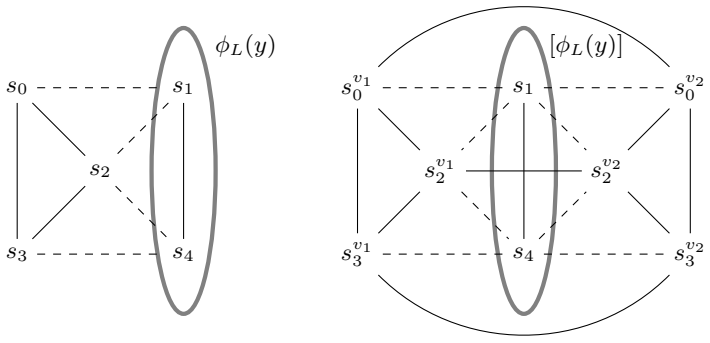
$L' = B$, then the only possibility is $[c] = [x]$ and $[d] = [x']$, for which the join must be [1], and is well defined. Now, from the Definition 12 (4.) of $\prec$, it follows that for every pair of ordered elements $[c] \leq [f]$, there must be one $L' \in \{L, B\}$ such that $c \in L' \cap [c]$ and $f \in L' \cap [f]$, with $c \leq f$. Now, this ensures, on one hand, that orthogonal joins (and meets) are well defined in the $I - pasting$, whenever they are well-defined on L and B. Indeed if $c \in L' \cap [c]$ and $f \in L' \cap [f]$, with $c \leq f$ holds for both $L' = L$ and $L' = B$, ϕ'_L preserving order, it must be either $c = 0_{L'}$ or $f = 1_{L'}$.

On the other hand, since L' is an OMP, then $c \leq f$ implies that $f = c \vee (f \wedge c')$, hence $[f] = [c] \vee ([f] \wedge [c]')$. □

In the following, $L|I|B$ will refer to this particular construction, and L will be assumed to be stable. Furthermore, we will suppose that $\phi_L(y) = x$ is a free atom, and show that $L|I|B$ is stable whenever L is.

We start defining $J'(L) = (Q'_L, E'_L, T'_L)$ in the following way:

$$
\begin{aligned}
Q'_L &= S_y \cup \{s \cup \{v_i\} | s \in S_{y'}, \phi_B(y) \neq v_i \in \mathcal{A}(B)\} \\
E'_L &= \{[s, s'] | s, s' \in Q'_L, s \neq s'\} \\
T'_L &= \{(s, [s, s'], s') | s, s' \in Q'_L, [s, s'] \in E'_L, s \neq s'\}.
\end{aligned}
$$



$J(L)$ $\qquad$ $J'(L) \cong J(L|I|B)$

Fig. 4. Construction of $J'(L)$, where L is the OMP of Figure 2, I is as in Theorem 3, and B is a Boolean algebra with three atoms: $\mathcal{A}(B) = \{\phi_B(y), v_1, v_2\}$. Lines represent transitions in both directions. Dashed lines have an incidence with respect to $\phi_L(y)$ or $[\phi_L(y)]$, whereas solid lines are independent from them.

Lemma 3. *$J'(L)$ is isomorphic to $J(L|I|B)$. Furthermore, for every $v_i \in \mathcal{A}(B)$ such that $\phi_B(y) \neq v_i$, the subgraph of $J'(L)$ induced by $S(y) \cup S(v_i)$ is isomorphic to $J(L)$.*

Proof. Every state $q \in Q'_L$ contains one, and only one, atom of B and since $\phi_L(y)$ is a free atom, it has one, and only one, atom for every Boolean algebra of L. Hence the elements of Q'_L coincide with the elements of $S(L|I|B)$. Since the construction of $J'(L)$ is completely determined by the set of states as in the construction of $J(L|I|B)$, the two transition systems $J(L|I|B)$ and $J'(L)$ are isomorphic.

We also observe that for every atom $v_i \in \mathcal{A}(B)$ distinct from $\phi_B(y)$ the elements in $S(y) \cup S(v_i)$ coincide with the elements of $S(y) \cup S(y')$, which is the set of states of $J(L)$. From this observation it is easy to see that there is an isomorphism between $J(L)$ and any subgraph of $J'(L)$ induced by a set of states in the form $S(y) \cup S(v_i)$. □

This last lemma will permit to consider $J'(L)$ instead of $J(L|I|B)$.

Lemma 4. *If a region $H \in \mathcal{R}(J'(L))$ contains a state $s \cup \{v_i\} \in Q'_L$ and $S_{v_i} \not\subseteq H$ then $\forall s \cup \{v_j\} \in Q'_L : s \cup \{v_j\} \in H$.*

Proof. Since $S_{v_i} \not\subseteq H$ there must be a state $s' \cup \{v_i\} \notin H$, which means that the event $[s, s']$ is an event labeling a transition exiting H. Now suppose that there is a state $s \cup \{v_j\} \in Q'_L$ that doesn't belong to H. This means that the transition from $s \cup \{v_j\}$ to $s' \cup \{v_j\} \in Q'_L$ does not exit H, but such a transition is also labeled $[s, s']$, which is not possible since that would mean that H is not a region. □

Lemma 5. *Every atomic region of $J'(L)$ is in the form S_x for $x \in \mathcal{A}(L|I|B)$.*

Proof. Assume that there is an atomic region $H \notin \{S_x \mid x \in \mathcal{A}(L|I|B)\}$. Consider the subgraphs induced by $S_y \cup S_{v_i}$ for all the $\phi_B(y) \neq v_i \in B$ and call $H_{v_i} \subset H$ the sets $H \cap (S_y \cup S_{v_i})$. All the H_{v_i} are atomic regions in every subgraph of $J'(L)$ induced by $S_y \cup S_{v_i}$, since they are all isomorphic to $J(L)$. The regions H_{v_i} can be atomic or not. If they are atomic, then they must coincide with an atomic region S_x, with $y \neq x \in L_n$. Hence, after Lemma 4, $H \in \{S_x\}_{x \in L_{n+1}}$. If they are not atomic, then there are $H'_{v_1} \subset H_{v_1}, \ldots, H'_{v_k} \subset H_{v_k}$ from which we can make the region $\bigcup_{i \in \{1,..,k\}} H'_{v_i} \subset H$, hence H is not atomic. □

Theorem 4. *Let L be a stable* OMP*, and $x \in \mathcal{A}(L)$ be a free atom. Let B be a finite Boolean algebra, and $a \in \mathcal{A}(B)$. Let $I = \{0, 1, y, y'\}$, and define $\phi_L : I \to L$, and $\phi_B : I \to B$ such that $\phi_L(y) = x$, and $\phi_B(y) = a$. Then $L|I|B$ induced by the V-formation $(I, L, B, \phi_L, \phi_B)$ is stable.*

Proof. We wish to show that $H(J(L|I|B)) \simeq L|I|B$. With Lemma 3, we reduce it to showing that $H(J'(L)) \simeq L|I|B$. Since $H(J'(L))$ is a finite OMP, it is characterised by the orthogonality relation among its atoms. Now, Lemma 5 states that each atom of $H(J'(L))$ corresponds to an atom of $L|I|B$, and it was shown in [4] that every atom of $L|I|B$ must be an atom of $H(J'(L))$.

For each pair of elements $x \in \mathcal{A}(L)$, $y \in \mathcal{A}(B)$, there is a state in $s \in J(L|I|B)$ such that $[x] \in s$ and $[y] \in s$. Hence, the pasting must preserve incompatibility. Since the pasting also preserves orthogonality, we have that $L|I|B$, and

$H(J(L|I|B))$ have same collection of atoms, with identical orthogonality relations. As it was shown in [5], this is sufficient to state that $H(J(L|I|B)) \simeq L|I|B$ □

3.3 Stability of Atom Refinement

The operation of refining an atom of an OMP into two new atoms preserves stability.

Theorem 5. *Let L be a stable* OMP. *Let $x \in \mathcal{A}(L)$. Consider $M_a = (\mathcal{A}(L) \setminus \{x\}) \cup \{y, z\}$, in which all orthogonal atoms of L remain orthogonal in M_a, and all atoms orthogonal to x in L are orthogonal to both y and z in M_a. Then the* OMP *M generated by M_a is stable.*

Proof. We will consider only the atoms of L and M, and states as represented by maximal cliques of $\not\perp$ as in Definition 7. Let $S_{x'}$ be the set of states of L not containing x. By construction of M_a, for each state $s \in S_x$ of L there are two states of M, in S_y and S_z respectively. Furthermore, the states of $S_{x'}$ all contain an atom orthogonal to x in L, and it will be orthogonal to both y and z in M. Thus, $S_{x'}, S_y$ and S_z constitute a partition of the states of M.

Starting from the states of M as partitioned above, it is possible to define the following sets of events: $E_{x'} = \{[s, s'] \,|\, s, s' \in S_{x'}, s \neq s'\}$, $E_{y,z} = \{[s, s'] | s \in S_y, s' \in S_z\}$, $E_y = \{[s, s'] | s, s' \in S_y\}$, $E_z = \{[s, s'] | s, s' \in S_z\}$, $E_{x',y} = \{[s, s'] \,|\, s \in S_{x'}, s' \in S_y\}$ and $E_{x',z} = \{[s, s'] \,|\, s \in S_{x'}, s' \in S_z\}$.

Let A_y be the transition system with the following sets of states and events: $S_{x'} \cup S_y$ and $E_{x'} \cup E_y \cup E_{x',y}$, let, symmetrically, A_z be the transition system whose states are $S_{x'} \cup S_z$ and whose events are $E_{x'} \cup E_z \cup E_{x',z}$. We note that both $\mathcal{R}(A_y)$ and $\mathcal{R}(A_z)$ are isomorphic to the regions of the saturated system $J(L)$ since in both cases of $\mathcal{R}(A_y)$ and $\mathcal{R}(A_z)$, atoms y and z replace uniformly x. Moreover, since states S_y and S_z are disjoint, it is possible to construct the CETS $A = A_y \cup A_z$ endowed by all the new events in $E_{y,z}$. We note that $\mathcal{R}(A)$ must contain $\mathcal{R}(J(M))$ since CETS A, having less events than $J(M)$, can have less constraints in the construction of its regions. We want to show that $\mathcal{R}(A) = \mathcal{R}(J(M))$. Let, by contradiction, r be a region in $\mathcal{R}(A)$ not belonging to $\mathcal{R}(J(M))$.

If $r \subseteq S_{x'}$, then $r \in \mathcal{R}(J(M))$ since all the labels in $E_{x'}$ belong to both CETS A and $J(M)$ and the new events in $E_{x',y}$ and $E_{x',z}$ are distinct copies of the original events $E_{x',x}$ in $J(L)$, so they do not create new regions. If $r \subseteq S_y$ and, symmetrically, for $r \subseteq S_z$ then r must be a region in $\mathcal{R}(J(M))$ since all the labels in E_y and $E_{x',y}$, and symmetrically E_z and $E_{x',z}$ are, by construction, isomorphic to the labels $E_{x',x}$ in $J(L)$ and all the new labels $E_{y,z}$ are exiting from or, respectively, entering in r. The only remaining case could be for r being a minimal region in $\mathcal{R}(A)$ and a non minimal region in $\mathcal{R}(J(M))$ but this would be in contradiction with y and z being atoms in M. □

Example 4. Consider three Boolean algebras B_1, B_2 and B_3 with three atoms each, $\mathcal{A}(B_i) = \{a_i, b_i, c_i\}$ for $i = 1, 2, 3$. Let $I = \{0, x, x', 1\}$ and consider the two OMP-morphisms $\phi_i : I \to B_i$ $(i = 1, 2)$, such that $\phi_i(x) = c_i$. B_1 is a stable logic,

and c_1 is clearly a free atom, so after Theorem 4, $L = B_1|I|B_2$ is a stable OMP. L is isomorphic to the OMP in Figure 2, by considering $b_3 = [\phi_1(x)] = [\phi_2(x)]$. Since L is stable, and b_5 is a free atom, we can compose it with B_3, by means of the morphisms ϕ_L and ϕ_3, provided by $\phi_L(x) = b_5$ and $\phi_3(x) = c_3$. The Greechie diagram of $L|I|B_3$ is depicted at the top of Figure 5. Thanks to Theorem 5, we can now split any atom of $L|I|B_3$, obtaining, for example, the stable OMP depicted at the bottom of Figure 5.

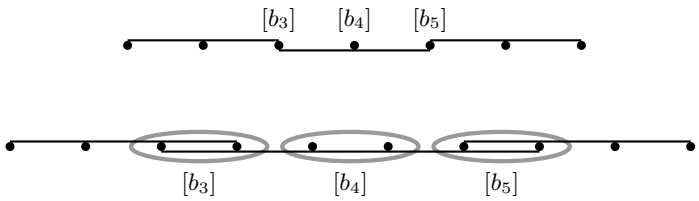


Fig. 5. Greechie diagrams of two OMPs. The OMP depicted below, is obtained from the one depicted above, by refining the atoms as shown. Since the OMP above is stable, so is the one below.

A free atom of the operands can be refined both after and before the composition operation, in this second way, only one of the two refining atoms will be used as interface with the appended sequential component, the other one remaining free for further composition. With this method, one can iterate the composition operation without worrying about exhaustion of available free atoms of the original system.

Example 5. Consider a system made of two sequential components each of which can get to a state for which they require the same resource. If each of these components can be in two additional states, the regional OMP for this system is represented as L_1 in Figure 6. B_1 and B_2 represent the two sequential components, and x_1, x_2 correspond to their mutually exclusive states. B_c represents the state of the resource c, it can be in state x_i, indicating that B_i holds c, or in state y_1, indicating that c is available. When c is available, no other component is involved in the coresponding local state of B_c, and so y_1 is a free atom. L_1 is isomorphic to the OMP at the top of Figure 5, and was shown to be stable in Example 4. We may want to make the resource c available for a third sequential component B_3, so we can use Theorem 4 to compose L_1 and B_3 on y_1, obtaining the stable $L_2 = L_1|I|B_3$ a shown in Figure 6. However, in this new compound system, the resource must be held by one of the three components B_i, as B_c has no more free atoms. No additional component can be added to the system, to compete for c. Instead of performing the composition $L_1|I|B_3$ directly, we can first make use of Theorem 5, and refine y_1 in L_1 into two new free atoms x_3, y_2, thus obtaining the OMP L_3 of Figure 6. We can now compose it with B_3 on x_3, thus obtaining $L_4 = L_3|I|B_3$, which is stable. One can see that y_2 remains a free

atom, a local state representing that the resource is available. This process can be iterated, to obtain a system with n sequential components competing for the same resource.

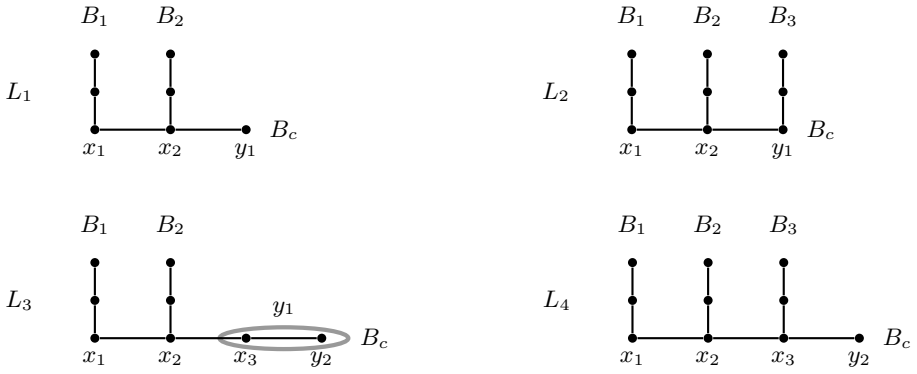


Fig. 6. Refining a free atom

4 Conclusion

In [4], a collection of regional OMPs were shown to be stable. Furthermore, the parallel composition operation was shown to preserve stability. In the present work, we have formalised two additional operations which preserve stability. One corresponds to the refinement of a local state into two. The second corresponds to the extension of a system with a sequential component which synchronises with it over a local state which is not already a synchronisation. With these elements, we can define an algebra of system OMPs, such that all its elements are stable.

However, not every regional OMP can be obtained with the defined operations. For instance, a strong limitation is the restriction of the composition operation to interfacing on free atoms. Another limitation of this operation is that it does not allow for extending a system with a sequential component that would synchronise with the system at more than one state. A clear goal in our approach is to extend the composition operation so as to generate every regional OMP. In this way, by showing that we can generate all regional OMPs with stability preserving operations, we would prove the conjecture that all regional OMPs are stable.

Acknowledgements

We wish to thank Lucia Pomello, and Luca Bernardinello for the fruitful discussions. Thanks to the three anonymous reviewers for the useful remarks and suggestions. This work is partially supported by MIUR.

References

1. Luca Bernardinello. Synthesis of net systems. In Marco Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993, 14th International Conference, Chicago, Illinois, USA, June 21-25, 1993, Proceedings*, volume 691 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1993.
2. Luca Bernardinello, Carlo Ferigato, and Lucia Pomello. An algebraic model of observable properties in distributed systems. *Theoretical Computer Science*, 290(1):637–668, 2003.
3. Luca Bernardinello, Carlo Ferigato, Lucia Pomello, and Adrián Puerto Aubel. Synthesis of transition systems from quantum logics. *Fundamenta Informaticae*, 154(1-4):25–36, 2017.
4. Luca Bernardinello, Carlo Ferigato, Lucia Pomello, and Adrián Puerto Aubel. On stability of regional orthomodular posets. *Transactions on Petri Nets and Other Models of Concurrency*, 13:52–72, 2018.
5. Luca Bernardinello, Carlo Ferigato, Lucia Pomello, and Adrián Puerto Aubel. On the decomposition of regional events in elementary systems. In Wil M. P. van der Aalst, Robin Bergenthum, and Josep Carmona, editors, *Proceedings of the International Workshop ATAED 2018 Satellite event of the conferences: ICATPN 2018 and ACSD 2018, Bratislava, Slovakia, June 25, 2018.*, volume 2115 of *CEUR Workshop Proceedings*, pages 39–55. CEUR-WS.org, 2018.
6. Garrett Birkhoff and John Von Neumann. The logic of quantum mechanics. *Annals of Mathematics*, 37(4):823–843, 1936.
7. Gunter Bruns and John Harding. Amalgamation of ortholattices. *Order*, 14(3):193–209, Sep 1997.
8. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2001.
9. Jörg Desel and Wolfgang Reisig. The synthesis problem of Petri nets. *Acta Informatica*, 33(4):297–315, Jun 1996.
10. Andrzej Ehrenfeucht and Grzegorz Rozenberg. Partial (set) 2-structures. part I: basic notions and the representation problem. *Acta Informatica*, 27(4):315–342, 1990.
11. Andrzej Ehrenfeucht and Grzegorz Rozenberg. Partial (set) 2-structures. part II: state spaces of concurrent systems. *Acta Informatica*, 27(4):343–368, 1990.
12. D. J. Foulis and C. H. Randall. Operational Statistics. I. Basic Concepts. *Journal of Mathematical Physics*, 13(11):1667–1675, 1972.
13. Robert Goldblatt. *Topoi: the categorial analysis of logic*. Dover books on mathematics. Dover Publ., New York, NY, 2013.
14. R.I.G. Hughes. *The Structure and Interpretation of Quantum Mechanics*. Harvard University Press, 1989.
15. Mogens Nielsen, Grzegorz Rozenberg, and P. S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96(1):3–33, 1992.
16. Pavel Pták and Sylvia Pulmannová. *Orthomodular Structures as Quantum Logics*. Kluwer Academic Publishers, 1991.
17. C. A. Petri. General net theory. computing system design. In *Joint IBM-University of Newcastle upon Tyne Seminar, Sept. 1976, Proceedings*, 1977.
18. C. H. Randall and D. J. Foulis. Operational statistics. II. Manuals of operations and their logics. *Journal of Mathematical Physics*, 14(10):1472–1480, 1973.

Emails Analysis for Business Process Discovery

Nassim LAGA[1], Marwa ELLEUCH[1,2], Walid GAALOUL[2], and Oumaima ALAOUI ISMAILI[1]

[1] Orange Labs, France, {FirstName}.{LastName}@orange.com
[2] Telecom SudParis, France, {FirstName}.{LastName}@telecom-sudparis.eu

Abstract. Most often, process mining consists of discovering models of actual processes from structured event logs. However, some business processes (BP), or at least some parts of them, are not necessary supported by an information system (IS), and consequently do not leave any structured events log. Therefore, applying traditional process mining techniques would generate a partial view of such processes. Process actors often rely on communication tools to collaboratively execute their business processes in such situations. However, given the unstructured nature of communication tools traces, process mining techniques could not be applied directly; thus it is necessary to generate structured event logs by recognizing the process-related items (activities, actors, instances, etc.). In this paper, we address this challenge in order to mine business processes from email exchange traces. We introduce an approach that minimizes users' efforts to manage the growing amounts of exchanged emails: It enables to *collaboratively*, and *gradually* build an annotated corpus of messages, and to *automatically* classify these ones into process, instance and activity IDs using machine learning techniques. Compared to related works, we facilitate the task of obtaining annotated datasets and we investigate the use of email exchange histories, correspondent, references and named entities for building clustering and classification features. The proposed approach is evaluated through a proof of concept and successfully experimented on an email dataset.

Keywords: Process mining · Business process management · Clustering · Supervised learning · Named entities.

1 Introduction

Process mining consists in extracting useful knowledge from event logs generated by a variety of software tools hosted in Information Systems (IS) [21,23]. It enables business experts to discover new processes, new practices, as well as BP limitations. However, most of them assume that: (1) Event logs have a **structured** format, and (2) the business process is **totally executed in IS**, and consequently event logs contain the trace of **all** executed business tasks. However, several business activities could be achieved using informal methods, such as communication tools (e.g. email exchange, IM, etc.). As a consequence, the traces of these activities are not present in traditional event logs. In addition, they are often not structured. One of the important tasks to be handled

when starting from unstructured log data to mine processes is how to convert it into structured event logs, which is compatible with the available process mining techniques. The task of constructing structured event log starting from unstructured log data of a given trace of communication consists mainly in recognizing the process-related items (name, activity and instance).

Up until recently, only few studies have properly addressed the recognition of all these related information. Most of them focus on the unstructured email exchange traces to mine business processes by using learning [8,4,17,10,7,9] or pattern matching [11,1] techniques. These proposals suffer from the following limitations. Firstly, they require a considerable **human intervention** with time consuming tasks. In the case of pattern matching based approaches, patterns are defined manually. As for the supervised learning based approaches, a huge human effort is required for building a training dataset: In most cases, a big amount of unorganized and unlabeled data has to be manually annotated. Even if unsupervised learning techniques can be introduced as a possible alternative to avoid preparing such a corpora [8,9], some manual tasks are still needed; they consist in labeling or modifying (correcting) the generated clusters and tuning the parameters of some parametric algorithms such as kmean and hierarchical algorithms. Second, they tend to generate **unreliable business process models**. This can be the result of: (1) Relying only on clustering techniques, which is error prone [8,9], and (2) using features that are not discriminative enough to recognize some business process related information [4,17,10,7]. Discriminative features are the most relevant variables for clustering. These latter depend on the type of the knowledge that we want to extract. In the case of process mining, existing works mostly exploit the entire content of the unstructured email related data (subject or email body) to build some of their learning features [12,8,9]. These features are used then for all kind of BP knowledge extraction tasks. Typically, the textual data of emails may contain key terms which can help to separate emails according to their BP for example. However, given emails belonging to the same BP but to different instances, their textual data are likely to share the same BP key terms, which means that using them entirely probably increases the confusion between instance clusters. On the other hand, named entities (e.g. person, company names) and references (e.g. customer reference, product reference) differ probably from one instance to another even if they belong to the same BP, which means that they can have a considerable contribution in separating emails into instances.

In this paper, we address these challenges in order to mine business processes from email logs. We consider that human intervention is often required to generate reliable business process models but we aim to minimize it. We propose then an approach that enables users to *collaboratively* and *gradually* build an annotated corpus of messages and to automatically classify these ones into process, instance and activity IDs using machine learning techniques. Our proposal differs from existing works by: (1) Reducing the human effort required for obtaining an annotated dataset through a collaborative and progressively learning approach (2) Investigating the use of email exchange histories, its participant correspon-

dent entities, references and named entities existing in its body for building clustering and classification features (3) Applying a fast and non-parametric clustering algorithm for process instance detection.

The rest of the paper is organized as follows. First, we introduce in section II an overview of our contributions, the overall algorithm, and functional entities. Then, we detail them in section III. In section IV, we validate the proposals by a proof of concept and its application to detect some processes (e.g. hiring process and patent application process) taken from real data. We discuss the related work in section V and conclude with a summary and perspectives in section VI.

2 Overview

Our proposal combines classification and clustering techniques for mining process models from email exchange traces. To achieve this task, a structured event log which contains process, activity and process instance labels must be generated. In this paper, we assume that each email is related to one activity, one process, and one instance. Our approach is summarized in Figure 1 and is a sequential combination of the following steps:

Step 1: Process and activity labels generation: This step is initially done manually and collaboratively by users. Then, a predictive model is trained gradually with the obtained annotated data for recognizing process and activity names. The classification features, which are used in the training phase, are built from the following email parts: subject, content, historical exchange and correspondent entities of email participants. Once reaching reliable prediction performances, the task of process and activity labels generation will be automatically performed by the obtained predictive models.

Step 2: Process instance detection: The purpose of this step is to detect the process instance related to each email. A clustering algorithm is applied at this step. The used distance matrix is defined as a weighted sum of sub-distances related to the following email parts: (1) time, (2) correspondent entities of email participants, and (3) content and subject reduced into references and named entities.

Step 3: Event log generation: The goal is to generate time-ordered per-process event sets. Each event presents an email with its timestamp and its activity, process and instance ID labels.

Step 4: Process model discovery: Any process discovery algorithm can be applied here to mine the business process models.

3 Approach

3.1 Step1: Process and activiy labels generation

The goal of this step is to associate process and activity labels to new incoming emails. More formally, it is a function $F : EM \rightarrow P \times A$ where EM denotes a set of emails e, P presents a set of process names and A presents a set of activities.

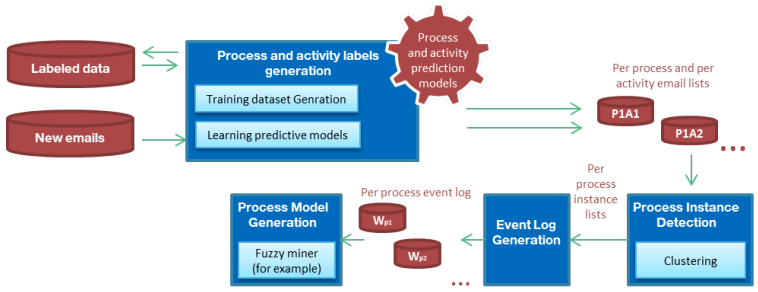


Fig. 1. Process Mining From Email logs: Main Steps

Algorithm 1 Process and activity labeling of one incoming email

procedure TAGEMAIL($e, aEL, pEL, pET, emC, erC, clfs, minBatch$)

▷ e: the email to be annotated
▷ aEL: the list of annotated emails
▷ pEL: the pseudo event log
▷ pET: the error rate threshold
▷ emC: the email count since model train (It is initialized outside the procedure)
▷ erC: the errors count since model train (It is initialized outside the procedure)
▷ clfs: the classifiers
▷ minB: the minimum size of the new data to train new models

$p \leftarrow False$ ▷ will contain the predicted process
$a \leftarrow False$ ▷ will contain the predicted activity
$uP \leftarrow False$ ▷ will contain the user given process
$uA \leftarrow False$ ▷ will contain the user given activity
$emC \leftarrow emC + 1$
if $clfs$ **then**
  $p, a \leftarrow PredictLabels(e, clfs)$
$uP, uA \leftarrow ManualTagging(e)$
if uP AND uA **then**
  $erC \leftarrow erC + 1$
  $p \leftarrow uP$
  $a \leftarrow uA$
  $aEL \leftarrow aEL \cup \{(e, uP, uA)\}$
  if $erC/emC > pET$ AND $emC > minBatch$ **then**
    $clfs \leftarrow TrainPredictiveModels(aEL)$
    $emC \leftarrow 0$
    $erC \leftarrow 0$
if p AND a **then**
  $pEL \leftarrow newEvent(pEL, e, p, a)$
return $p, a, erC, emC, aEL, clfs, pEL$

Each email e in EM is defined as 7-uplet : FROM (email sender), TOs (email recipients), CCs (emails addresses that are in copy of the email), SUB (email subject), CONT (email content), Timestamp, HistExch (The content of historical exchange of an email). This task is done manually and collaboratively by users until reaching reliable predictive models. Then, it will be performed automatically by these models which are trained using Mini-batch learning approach. This approach consists on waiting until obtaining a batch of new observations and then train the already existing models on this whole batch. The pseudo code of this step is described in Algorithm 1.

First, the associated process and activity names for each email e are predicted using existing models (line 8). Then, if manual annotations (process and activity names) are given by the user (this could happen if the annotations are not, or wrongly, predicted) errors rate is checked, if it is higher than a threshold (pET) and a minimum size of data is collected (minB), we train again the models (process and activity classifiers) (lines 10→16). Finally, the predicted, or corrected, process and activity names, associated to the email, are saved in the pseudo event log (pEL) dataset (line 20).

For learning or updating predictive models, we have to dispose a training dataset that must be converted into a (X,Y) couple, where X is a matrix having in each row the feature values of each sample and Y is a vector representing the corresponding targets. We further detail in the following sections our classification features and preprocessing steps applied to generate this matrix X.

a) Defining classification features: We split here the task of selecting the efficient attributes for building our classification features according to the prediction task type. To predict activities, we use the correspondent entities of email participants (FROM, TOs, and CCs) and the email subjects (SUB) and we add the content parts because correspondent and subject parts are not sufficient enough to recognize activities since they lack precision about them. We have handled also the case where the email contains one short sentence. This type of email can be sent, for example, to confirm or deny what has been said in previous emails. In this case, we use also the email exchange history (HistExch) because it is not obvious to understand the business goal of sending such emails without analyzing the history of the concerned discussions. To recognize Business processes, we use only the correspondent entities and the subject parts because we noticed that content parts degrade the recognition accuracy when they are introduced with the same preprocessing steps as in the activity prediction phase.

b) Defining preprocessing steps: We summarize these steps as follows:
Replace Particular Expressions: We detect, using regular expressions, special expression within the subject (and the body in the case of activity prediction) (e.g. HTTP links, phone numbers, special references if known, and file names and their extension), and replace these expressions with a special tag (e.g. HTTP_LINK, PHONE_NUMBER, IS_REF, and PPT_FILENAME etc.).
Remove Person Names: We detect all the correspondents' first names and last names, and then remove them from the textual data.
Lemmatize The Text: This method reduces the dimension of the resulting matrix (Eq (2)). It consists in reducing the different forms of word to a single form (e.g. words "thinks", and "thinking" into one single form "think").
Remove Stop Words: This function is useful to remove the most common words in the emails that can be distracting and non-informative. We used the nltk[3] library, enriched with additional words detected during the data exploration step (e.g. regards, hello, outlook prefixes, etc.).
Generate 1-gram,2-gram Vocabulary: We first split the remaining text into a list of words and detect the different sequential combinations where each item

[3] https://www.nltk.org

has the size of 1 to 2 words (1-gram, 2-gram). Then, we remove the most and the less frequent terms to improve generalization of our predictive models. In fact, sparse terms can generate wrong associations and overly common words don't present relevant information to differentiate between email intents.
Generate TFIDF (Term frequency-inverse document frequency): This function generates a TFIDF matrix which encodes the frequency of 1-gram and 2-gram terms in an email with respect to the rest of the corpus. The size of this matrix is equal to (N, T) where N is the total number of emails and T is the total number of different 1-gram and 2-gram terms.
Generate Interaction Matrix: In order to emphasize the contribution of the email correspondent entities in defining process and activity names, we build a presence matrix (PM) reflecting for each email the interlocutors' entities (sender entity, recipient entity, and copied entity).

Definition 1 *Let E be the set of emails of length N, L be the list of different entities of length M, C be a list where each element corresponds to the list of participant correspondents' entities of each email, PM be a matrix whose columns represent L and rows represent E. PM can be defined as follows:*

$$PM(i,j)_{i,j\in[0,N-1]\times[0,M-1]} = \begin{cases} 1 & \text{if } L[j] \in C[i] \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

GenerateTheFinalMatrix: This function generates the X matrix which is a weighted concatenation of PM and TFIDF matrices.

$$X = \left[\beta_1 TFIDF_{(N\times T)}\ \beta_2 PM_{(N\times M)}\right] \tag{2}$$

Where the weights β_1 and β_2 will be defined empirically.

3.2 Step2: Process Instance Detection

The goal of this step is to detect a specific occurrence or execution of the same business process, which is known as process instance. We use the *pseudoEventLog* dataset (pEL) generated as an output of step 1 which contains the list of emails correctly annotated with corresponding process and activity names. We first divide it into per-process groups. Then, we apply a clustering technique on each obtained group to detect clusters corresponding to process instances.

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) should have similar properties or features, while objects in different groups should have highly dissimilar ones. A clustering algorithm takes as input a similarity matrix M which defines the similarity between each couple of emails. This matrix is formally specified in Definition 2

Definition 2 *Let N be the total number of emails, E be the set of emails in our corpus and $f : E \times E \to R$ be the similarity function that calculates a similarity value between two emails. The similarity matrix M can be defined as a square matrix of size $N \times N$ where $M[i,j]_{i,j\in[1,N]} = f(E[i], E[j])$*

In our case, the similarity function f is a distance function defined by Eq(3).

Our clustering phase goes mainly through the following sub-steps: (1) Identify clustering features. (2) Generate the similarity matrix. (3)Apply a clustering algorithm.

a) Identify clustering features For building our clustering features, we focus the analysis on: (1) Subject and content reduced to references and named entities (2) Time (3) Correspondent entities. In fact, we believe that emails belonging to the same process instance are likely to have close reception dates and to share the same named entities, the same references and the same correspondents entities (from, dest, and CC).

A named entity is a real-world object, such as persons, locations, organizations, products... etc that can be denoted with a proper name. The references are the information generated by business applications used for executing some process tasks (e.g. customer number, purchase request ID etc.,). The purpose of reducing the email into references and named entities is to conserve only the significant contextual data in relation with the business process instances. In fact, the entire content of these email parts often contain additional vocabulary, which degrades instance detection accuracy.

The named entities are detected through two methods which are complementary in our case: We first use the Polygot NER method [2]. Named-entity recognition (NER) is a subtask of information extraction. It seeks to locate and classify named entity mentions in unstructured text into pre-defined categories such as the person names, organization, time expression, monetary values, percentage etc.Unlike the other existing pipelines (NLTK, standford, OpenNLP) where most languages are unsupported, Polygot NER is a multilingual named entity recognition tool that supports 40 major languages. It is also automatic but not complete (some named entities are not detected). To overcome this limitation, we express explicitly the non-detected patterns. As for reference detection, we inject specific regular expressions.

b) Similarity function Our similarity function is a distance function which is defined as a weighted sum of sub-distances related to our clustering features. It has the following formula:

$$f(x,y) = w_0 DC(x,y) + w_1 DT(x,y) + w_2 DNE(x,y) \tag{3}$$

Where the weights w0, w1 and w2 are defined empirically according to each process type and DC(x,y), DT(x,y) and DNE(x,y) are defined as follows:
- **DC(x, y)** is the correspondent distance between two emails x and y. We define it as a Jaccard distance between the correspondent entity sets of their interlocutors which is equal to the cardinality of their intersection divided by the cardinality of their union. More formally, let C(x) be the list of correspondents of the email x, and C(y) be the list of correspondents of the email y. DC(x,y) is then defined as follows:

$$DC(x,y) = \frac{|C(x) \cap C(y)|}{|C(x) \cup C(y)|} \tag{4}$$

- **DT(x, y)** is the time distance between emails x and y. The emails belonging to the same process instance are likely to have close reception dates. We assume that the inter-arrival duration follows an exponential law and is expressed in number of days. Consequently, we define the distance through the following formula:

$$DT(x, y) = 1 - e^{-\lambda(ts(x)-ts(y))} \tag{5}$$

ts(x) and ts(y) refer to the timestamp of x and y and λ is the time, expressed in the number of days, which separates two emails arrivals. We estimate this value by:

$$\lambda = \frac{date\_max - date\_min}{number\_of\_emails} \tag{6}$$

- **DNE(x,y)** is the distance related to the named entities and the references present in the subject and the content of the email x and those present in email y. Emails belonging to the same process instance are likely to share the same named entities and references. We define the distance DNE as a Jaccard distance:

$$DNE(x, y) = \frac{|NE(x) \cap NE(y)|}{|NE(x) \cup NE(y)|} \tag{7}$$

NE(x) is the set of named entities and references present in email x, and NE(y) is the set of named entities and references present in email y.

3.3 Step 3: Event Logs Generation

Algorithm 2 Event logs generation

```
1: procedure GenerateEventLog(pEL)
                                                              ▷
   ▷ pEL: The pseudo event log
2:    pEL_ordered ← TimeBasedOrdering(pEL)
3:    EM_P ← PerProcessSplitting(pEL_ordered)
4:    EM_P_I ← InstanceDetection(EM_P)
5:    InstanceIdentifierGeneration(EM_P_I)
6:    EventLogs = PerProcessGrouping(EM_P_I)
7:    return EventLogs
```

Once the process and activity names are identified and the emails belonging to different process instances are grouped, a dataset labelled with process, activity, timestamp and instance_ID labels can be obtained. The event logs generation function aims to construct, from this new dataset, time-ordered per-process event sets. Algorithm 2 summarizes our steps to generate it. From the pseudo event logs obtained from step 1, we generate a time-ordered event logs using the email

timestamp variable (line 2). Then, we split it into several subsets, each one containing a single process emails list (line 3). For each process subset, we apply a clustering algorithm in order to detect instance groups and we associate to each group a unique identifier (line 4, 5). Finally, all these groups are regrouped into a per process dataset (line 6) so that a process mining algorithm could be applied.

4 Validation

We validate our contribution through a proof of concept and experimentations carried on our dataset composed of 1026 emails and on the email environment of two employees (Microsoft Outlook as an email client, and Microsoft Exchange as an email server). In these experimentations we succeed to discover two processes: (1) a hiring process and (2) a patent application process. In this section we detail only the hiring process discovery.

4.1 Proof Of Concept

Our tool is implemented through three components:

a) Frontend component: This component is a Microsoft Outlook 2010 plugin developed using C# programming language. It has four main functionalities. First, it enables the user to manually annotate his emails. Thus, it provides the GUI that enables the user to select the email and the related process and activity names. This association is sent to the backend (SetTag interface) as a JSON object containing the email parameters and the associated process and activity names. Second, it captures incoming and outgoing messages, constructs a JSON object for each email, sends it to the backend for analysis (GetTag interface), retrieves the results (JSON object representing the detected process name and activity name) and associates the tags to the email. Third, it enables the display of the process and activity related to each email along with the email. Finally, the plugin provides users with advanced functionalities such as email search by related business process and activity.

b) Backend component: The backend component is implemented through three HTTP interfaces (SetTag, GetTag, and GetAllProcesses), with a MySQL database containing two tables: training dataset table and event logs table. The training dataset contains the following columns (id, source, destination, cc, subject, content, received_date, process_name, activity_name). The event log dataset contains the following columns (id, source, destination, cc, subject, content, received_date, process_name, activity_name, instance_id).

- SetTag Interface: This interface is used to enrich the learning database from one hand. It is invoked by the frontend when the user annotates manually an email. It receives the JSON object representing the email and the associated process and activity names. This information is inserted into the training dataset table as well as into the event log table, in which we set the instance_id column to NULL value, as we don't know yet to which instance the email belongs to.

- GetTag Interface: This interface is invoked by the Microsoft Outlook Plugin each time the user sends or receives an email. It receives a JSON object representing an email, analyzes it using the trained predictive models, and returns as a result the predicted process and activity name IDs. The result is also inserted into the event log table.
To build the machine learning models, a multinomial version of the logistic regression (LR) classifier is employed. It estimates the parameters of a logistic model for multiclass prediction task. The stochastic Gradient Descent (SGD) is used as an optimizer in the training phase. In fact, it can converge faster than batch training because it performs updates more frequently. Therefore, it has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing fields.
- GetProcesses Interface: This interface enables the email client user to display existing tags (process names and activity names). It supports him in the process of enriching the learning database. Basically, this interface is invoked when the user is about to manually annotate an email. It enables to retrieve the list of available annotation in the training dataset table. For each process name, we also retrieve the list of associated activities.

c) Instance separation component: To detect instances, we applied the clustering algorithm HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise). It extends DBSCAN by converting it into a hierarchical clustering algorithm, and then using a technique to extract a flat clustering based on the stability of clusters. The choice of HDBSCAN is justified by two reasons: (1) HDBSCAN does not require human intervention. In fact, it is a fast and non-parametric algorithm that does not require setting any parameters even the number of clusters. (2) HDBSCAN can generate clusters of different sizes, shapes and densities, which can enhance clustering accuracy.

4.2 Experimentations

We carried here experimentations to justify our choices in each step and then to evaluate their performances and to present the obtained results.

a) Validation of process and activity prediction: We manually annotated 1026 emails to obtain a correctly annotated email corpus with process and activity IDs. There are 13 processes (e.g. Hiring, PatentApplication, Command, ConferenceParticipation, travel expense refund, etc.) and 116 activities. Taking the example of hiring process, we identified the hiring steps achieved through emails: “describe the position”, “publish the position”, “receive applications”, “setting the interviews”, “asking for documents”, and “notifying the decision”.

The performances of process and activity prediction phase highly depend on the choice of the machine learning algorithms and the data preprocessing actions. To select these latter, we have tested different techniques until good prediction performances are reached. Better performances are noticed when:
- Using only the subject and the correspondent entities for process prediction.
- Applying the preprocessing steps detailed in II.B.2.b when we consider that the most frequent terms in the documents have a frequency greater than 5% and

the less frequent ones have a frequency less than 0.1%.
- Assuming that short emails contain less than 40 words and that email exchange history is constructed from the four previous emails.
- Setting the weights of Eq (2) as follows: $\beta_1 = 0.8$ and $\beta_2 = 0.2$.
- Using the LR with SGD optimizer for training predictive models. This result is obtained after testing two other prediction algorithms (Random Forest (RF) and Support Vector Machine (SVM)). The performances of each one were estimated by using 5-fold cross-validation method and by calculating the F1 Score. This score is a measure that combines precision and recall. Precision is known as positive predictive value while recall is called the sensitivity of the classifier. Mathematically, the F1 score is defined as:

$$F1Score = \frac{2 \times precision \times recall}{precision + recall} \tag{8}$$

The obtained F1 scores are summarized as follows: 0.8072 for Random Forest, 0.8626 for LR with SGD and 0.8094 for SVM.

b) Validation of Process Instance Detection: In this subsection, we evaluate the performances of our selected clustering technique HDBSCAN on our data set composed of 180 emails related to a hiring process.

We manually generated the emails clusters related to the process instances where we obtained 11 clusters. To compare this manual clustering with the results of HDBSCAN, we computed the Adjusted Mutual Information[4] (AMI) metric [22]. We tested different values of the weights related to distance matrix computation (Eq (3)), and we obtained an optimal configuration (using these values $w_0 = \frac{1}{2}$, $w_1 = \frac{1}{4}$, and $w_2 = \frac{1}{4}$). The AMI value obtained with this configuration is 0.86.

c) Validation of Process Model Discovery: S05cm To validate this step,

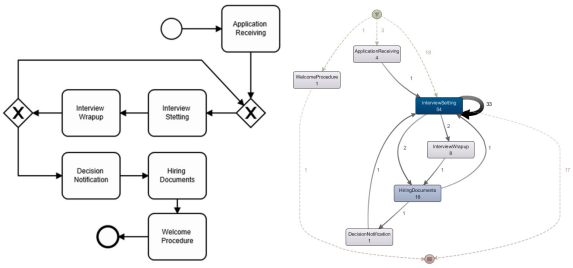


Fig. 2. Hiring process models: Real model VS Captured model

we applied the heuristic miner algorithm [23] on the automatically generated

[4] AMI computes the metric to evaluate the similarity between two hdbscan partition and real partition. It returns a value between 0 and 1. This value is closed to 1 when the two partitions are strongly matched and closed to 0 when the two partitions are weakly matched

event log of the hiring process. Figure 2 shows the theoretical and the detected model. We can notice that the behavior captured in the event log is almost in conformity with the theoretical one. Nevertheless, two discrepancy types can be detected at low frequency: (1) Unfitting model behavior which refers to behaviors observed in the theoretical model that are not allowed by the captured one (e.g."Welcome procedure" is performed after "Decision Notification"). (2) Additional model behavior which refers to behaviors allowed in the captured model but does not exist in the theoretical one such as: "Welcome Procedure" is done initially and asking for"Hiring Documents" or "Decision Notifying" are done before "Interview Setting". Actually, these discrepancy types can be caused either by errors accumulated through our log building system, or by the log miner technique that we have used or by a real difference between the process as observed in the emails and the related theoretical BP.

5 Related Works

Up until recently, only few studies have considered business process mining from email exchange. They have been mainly interested in: Activity and process names recognition, process instances detection and process discovery. They do require human intervention and may generate unreliable business process models. In this section, we discuss them according to three categories:

5.1 Non-learning based methods

One of the first proposals for mining business processes from emails assumes that the associated business process is explicitly included in the email subject [1]. Such an approach requires a significant human intervention and involvement. Indeed, email interlocutors must include the business process name and related attributes in the email subject, which is not realistic.

Another proposal [11] assumes that the manual task is an association of (1) classical manual task of the BPMN2.0 specification [13] and (2) a set of semantic patterns that enable to validate whether a given communication content is part of a business process, activity and a given business process instance. The limitation of such system is the necessity of anticipating and manually defining all semantic patterns related to each task which is time consuming and not scalable.

E-Mail Mining [20] is a method for semi-automatic discovery of knowledge-intensive process. From a set of emails belonging to a BP, e-Mail Mining aims to discover the amount of knowledge embedded in the execution of its activities. This knowledge consists of : (1) BP participants and their social interactions (2) Relevant terms that are related to the BP domain (3) BP activities defined by three elements : Actors, candidate actions and parameters. Relevant BP activities are selected manually from a list of candidate activities. These latter are generated after splitting emails into sentences and by assuming that each sentence is composed of : (1) a noun phrase object which can describes the agent

performing an action or the resource that receives the effect of the executed action (2) a verb phrase that describes the activity performed by the agent. This approach has an interesting contribution in the field of process mining from emails. In fact, it allows the detection of multiple activities in one email as well as the metadata embedded in the execution of a given BP. However, it requires manual tasks during its execution (e.g for selecting relevant activities or sample of emails related to one BP). Moreover, it considers emails as storytelling textual data to mine the candidate activities. Actually , emails do not have the same structure as narrative textual data (which generally describes activities in a more formal way than e-mails). For instance, the proposal does not seem to handle passive-voice sentences where actors do not appear or where their positions are switched with those of resources.

5.2 Act theory based methods

This category deals with activity name recognition by using act theory based methods: The idea behind this theory is to classify emails according to the sender's intent [19,18]. Thus, two possible classifications of speech acts are proposed: (1) Illocutionary act classes; Assertive, Commissive, Directive, Expressive, declarations. (2) Speech act verbs: Propose, Request, Deliver, Commit, etc. Some proposals set email speech acts in advance. Then, they apply a supervised learning algorithm to classify emails as containing or not containing the specific acts [4,17]. Other works treat the problem of process detection as a problem of conversation finder such as [12]: It suggests firstly classifying emails into business and non-business process related. The business-oriented email messages are then grouped into threads to detect conversations using a refined version of Vector Space Model and a semantic similarity measure. Finally, the interactions in each conversation are labeled by applying the classification of illocutionary acts [19].

An iterative relational learning approach for email task management was also suggested [10]. It exploits the mutual performance improvement between the extraction of speech acts and the identification of related emails. In fact, after initializing both of them using automatic methods, a supervised learning algorithm (SMO implementation of SVM [16]) is applied on incoming emails: It takes into account related emails as a feature to recognize the correspondent speech acts. Then, a relational learning terminology [14] is exploited: It similarly uses speech act as a feature to predict relations between the incoming and the existing messages.

Obviously, all of these works require labeled data for training statistical speech acts recognizers which leads to a huge human intervention. Furthermore, business process tasks differ from one process to another. Thus, setting a unique list of activities in advance, degrades the performances of generating the right business process models.

5.3 Unsupervised learning based methods

In order to minimize the human intervention and to avoid preparing a labeled dataset, there exist propositions that have integrated unsupervised learning techniques in their approaches to mine business processes from emails [8,9,6,5]. One of these propositions identify the process and the instance clusters by applying a hierarchical clustering method (Bottom up) [8]. In order to find process groups, the distance used combines the subject and body attributes. Then, to detect instances, the timestamp attribute is added. As for the process activities identification phase, the K-mean algorithm is adopted. The approach proposes a customization method to set the number (K) and the initial centers, on the basis of the instance clusters obtained from the previous step. Then, it applies a distance formula that takes into consideration the meanings similarity of the words present in the subjects and the bodies.

Another approach proposes a two step algorithm to discover the processes and activities from emails [9]. A hierarchical clustering is sequentially applied: first to deduce process clusters and then to deduce sub-clusters corresponding to activity types. The similarity measurement is based on word2vect method: It aims to exploit the hidden semantic relations between words existing in email contents and subjects. An activity labeling technique is also proposed: It recommends to the user the most frequent contiguous sequence of n items existing in an activity type cluster.

These studies aim to minimize the human intervention. However, they have some limitations: First, the hierarchical clustering requires a human effort in tuning its parameters. Additionally, its quality highly depends on how these parameters are set [15,3]. Second, it is computational hard. Hence, applying the same algorithm twice in the same method increases the computational complexity and the execution time [8]. Third, the activity identification quality highly depends on the instance clustering phase [8]. In fact, as the K-mean algorithm is sensitive to the initial start centers, poor instance detection quality can certainly lead to a bad convergence. Finally, the automatic generation of labels considerably increases the risk of error and interpretation [9].

MailOfMine [6,5] proposes to mine artful business processes and to define them with a "declarative" approach. It suggests to start from some assumptions which map email and BP structures: (1) Each conversation presents an activity trace (2) Each activity presents a set of elementary tasks deduced from conversation key parts (3) Each process is composed by a set of activities. MailOfMine approach consists basically of: (1) Applying three times a similarity clustering algorithm: to cluster emails into conversation threads, to cluster these threads into activity types and finally, to cluster each activity key parts into task types. During the clustering process, email body, the names of attached files and some email header fields are taken into consideration.(3) Applying supervised learning process to assign activities to different processes (4) Automatically labeling activity tasks with the possibility of customizing them and manually assigning activity and process names (5) Mining constraints between tasks (activities) among each activity (process) . The proposed work has the advantage of dis-

covering BP with different level of granularity (Process, subprocess or activity, task) and describing them with declarative approach, which is more flexible than the classical imperative approach. Nevertheless, this work suffers from some limitations: (1) Its execution time can diverge when applying it on large number of traces containing various tasks and activity types, e.g; it is linear (quadratic) time with respect to the number, size of traces. (2) A considerable human intervention is required to manually assigning activity and process names and to initiate activity classification step.

6 Conclusion & Discussion

In this paper, a solution for mining business processes from email logs was proposed and evaluated through a proof of concept implementation and successful experimentations on our dataset.

To build a training dataset from existing corpus of emails, we proposed a collaborative and iterative approach which is implemented through graphical interfaces and automatic prediction functionalities. This has the advantage of encouraging users to be involved in building an annotated dataset since it facilitates the tagging task and minimizes the required effort. Consequently, the training dataset will be built gradually and will be available instantly without the need to dispose a lot of time and human resources. However, this approach still relies on human involvement. Moreover, tagging collaboratively the same dataset can lead to dispose samples belonging to the same cluster but with different annotations. Therefore, tag normalization step is required.

The prediction entity is based on a supervised learning technique. For building classification features, we have investigated, according to the prediction goal, some or all of these variables: email participant correspondent entities, subject, content and exchange history. Our experiments revealed that email contents degrade the performances of process name recognition. Even if this assumption seems contradictory to existing works [8,9], it can be justified by the nature of our dataset and our preprocessing steps.

Our instance detection approach differs from related works by using a fast and non parametric clustering algorithm which is non sensitive to the noise and which can generate clusters with different shapes, sizes and densities. Moreover, we have reduced the body and the subject into references and named entities for clustering emails into BP instances. To improve the detection quality of named entities and references, we have defined explicitly the non-detected patterns. This action has the advantage of having good performances, however, it is manual and by consequence costly. We have assumed that three variables (timestamp, correspondent entities and named entities) can contribute to separate BP instances. Actually, the contribution of each variable depends on the nature of the BP. This is why we have introduced weights correlated to each variable to be tuned by users according to their expertise. For instance, the timestamp variable can help to separate instances in the case of BP with time constraints (e.g; the accounting closing process that is carried out regularly on scheduled dates) while in the case

of BP whose instances are independent of time, the same variable seems to have no effect.

In our approach, we have handled some research questions related to BP mining from emails by supposing that one email can be affected to one process, one activity and one instance. Actually, messaging systems such as emails allows users to discuss BP issues with informal way without respecting such constraints; in one email, user can discuss more than one activity and more than one instance. This kind of challenges was addressed in previous works such as [20] by assuming that activities are expressed and generally correlated with their metadata at sentence level. In future works, we plan to study these challenges. We plan also to more automate the BP discovery pipeline since the current approach still requires human involvement. Finally, we suggest to employ similarity meaning measures for constructing learning features based on email contents.

References

1. van der Aalst, W.M., Nikolov, A.: Emailanalyzer: an e-mail mining plug-in for the prom framework. BPM Center Report BPM-07-16, BPMCenter. org (2007)
2. Al-Rfou, R., et al.: Polyglot-ner: Massive multilingual named entity recognition. In: SIAM International Conference on Data Mining. pp. 586–594. SIAM (2015)
3. Ciosici, M.R.: Improving quality of hierarchical clustering for large data series. arXiv preprint arXiv:1608.01238 (2016)
4. Cohen, W.W., et al.: Learning to classify email into"speech acts". In: Empirical Methods in Natural Language Processing (2004)
5. Di Ciccio, C., Mecella, M.: Minerful, a mining algorithm for declarative process constraints in mailofmine. Department of Computer and System Sciences Antonio Ruberti Technical Reports **4**(3) (2012)
6. Di Ciccio, C., Mecella, M., Scannapieco, M., Zardetto, D., Catarci, T.: Mailofmine–analyzing mail messages for mining artful collaborative processes. In: International Symposium on Data-Driven Process Discovery and Analysis. pp. 55–81. Springer (2011)
7. Jeong, M., et al.: Semi-supervised speech act recognition in emails and forums. In: Empirical Methods in Natural Language Processing. vol. 3, pp. 1250–1259. Association for Computational Linguistics (2009)
8. Jlailaty, D., et al.: A framework for mining process models from emails logs. arXiv preprint arXiv:1609.06127 (2016)
9. Jlailaty, D., et al.: Mining business process activities from email logs. In: Cognitive Computing (ICCC). pp. 112–119. IEEE (2017)
10. Khoussainov, R., Kushmerick, N.: Email task management: An iterative relational learning approach. In: CEAS (2005)
11. Laga, N., et al.: Communication-based business process task detection-application in the crm context. In: Enterprise Distributed Object Computing Workshop (EDOCW). pp. 1–8. IEEE (2016)
12. Mavaddat, M., et al.: Facilitating business process discovery using email analysis. In: The First International Conference on Business Intelligence and Technology. Citeseer (2011)
13. Model, B.P.: Notation (bpmn) version 2.0. OMG Specification, Object Management Group pp. 22–31 (2011)

14. Neville, J., Jensen, D.: Iterative classification in relational data. In: Learning Statistical Models from Relational Data. pp. 13–20 (2000)
15. Oyang, Y.J., et al.: Characteristics of a hierarchical data clustering algorithm based on gravity theory. Tech. rep., Technical Report of NTUCSIE 02-01.(Available at http://mars. csie. ntu. edu ... (2001)
16. Platt, J.C.: 12 fast training of support vector machines using sequential minimal optimization. Advances in kernel methods pp. 185–208 (1999)
17. Qadir, A., Riloff, E.: Classifying sentences as speech acts in message board posts. In: Empirical Methods in Natural Language Processing. pp. 748–758. Association for Computational Linguistics (2011)
18. Searle, J.R.: A taxonomy of illocutionary acts (1975)
19. Searle, J.R., Searle, J.R.: Speech acts: An essay in the philosophy of language, vol. 626. Cambridge university press (1969)
20. Soares, D.C., Santoro, F.M., Baião, F.A.: Discovering collaborative knowledge-intensive processes through e-mail mining. Journal of Network and Computer Applications **36**(6), 1451–1465 (2013)
21. Van Der Aalst, W., et al.: Process mining manifesto. In: International Conference on Business Process Management. pp. 169–194. Springer (2011)
22. Vinh, N., et al.: Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. Journal of Machine Learning Research **11**(Oct), 2837–2854 (2010)
23. Weijters, A., et al.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP **166**, 1–34 (2006)

On the Hardness of Synthesizing Boolean Nets

Ronny Tredup and Christian Rosenke

Universität Rostock, Institut für Informatik, Theoretische Informatik,
Albert-Einstein-Straße 22, 18059, Rostock

Abstract. Boolean Petri nets are differentiated by types of nets based on which of the interactions nop, inp, out, set, res, swap, used, and free they apply or spare. From the 256 thinkable types only a few have yet been explicitly defined, as for instance contextual nets {nop, inp, out, used, free} and trace nets {nop, inp, out, set, res, used, free}. The synthesis problem relative to a specific type of nets τ is to find, for a given transition system A, a boolean τ-net with state graph isomorphic to A. It is known to be NP-hard for elementary nets systems {nop, inp, out} and tractable for flip-flop nets {nop, inp, out, swap}. This paper presents a general reduction scheme for the NP-hardness of boolean net synthesis and identifies 67 new types with a hard synthesis problem.

1 Introduction

Boolean Petri nets have been widely regarded as a fundamental model for concurrent systems. These Petri nets allow at most one token per place in every reachable marking. Accordingly, a place p can be regarded as a boolean condition which is *true* if p contains a token and is *false* if p is empty, respectively. A place p and a transition t of a boolean Petri net are connected by one of the following (boolean) *interactions*: *no operation* (nop), *input* (inp), *output* (out), set, *reset* (res), *inverting* (swap), *test if true* (used), and *test if false* (free). An interaction defines which pre-condition p has to satisfy to activate t and it determines p's post-condition after t has fired: inp (out) mean that p has to be *true* (*false*) to allow t's firing and if t fires then p become *false* (*true*). The interaction free (used) says that if t is activated then p is *false* (*true*) and t's firing as no impact on p. The other interactions nop, set, res, swap are pre-condition free, that is, neither *true* nor *false* prevent t's firing. Moreover, nop means that the firing of t has no impact and leaves p's boolean value unchanged. By res (set), t's firing determine p to be *false* (*true*). Finally, swap says that if t fires then it inverts p's boolean value.

Boolean Petri nets are differentiated by types of nets τ accordingly to the boolean interactions they allow. Since we have eight interactions to choose from, this results in a total of 256 different types. Yet, research has explicitly defined seven of them: *Elementary net systems* {nop, inp, out} [10], *Contextual nets* {nop, inp, out, used, free} [7], *event/condition nets* {nop, inp, out, used} [2], *inhibitor nets* {nop, inp, out, free} [9], *set nets* {nop, inp, set, used} [6], *trace nets* {nop, inp, out, set, res, used, free} [3], and *flip flop nets* {nop, inp, out, swap} [11].

Type of net τ	Complexity status	Quantity
$\tau = \{\mathsf{nop}, \mathsf{res}\} \cup \omega$, $\omega \subseteq \{\mathsf{inp}, \mathsf{used}, \mathsf{free}\}$	polynomial time	8
$\tau = \{\mathsf{nop}, \mathsf{set}\} \cup \omega$, $\omega \subseteq \{\mathsf{out}, \mathsf{used}, \mathsf{free}\}$	polynomial time	8
$\tau = \{\mathsf{nop}, \mathsf{swap}\} \cup \omega$, $\omega \subseteq \{\mathsf{inp}, \mathsf{out}, \mathsf{used}, \mathsf{free}\}$	polynomial time	16
$\tau = \{\mathsf{nop}\} \cup \omega$, $\omega \subseteq \{\mathsf{used}, \mathsf{free}\}$	polynomial time	4
$\tau = \{\mathsf{nop}, \mathsf{inp}, \mathsf{free}\}$ or $\tau = \{\mathsf{nop}, \mathsf{inp}, \mathsf{used}, \mathsf{free}\}$	NP-complete	2
$\tau = \{\mathsf{nop}, \mathsf{out}, \mathsf{used}\}$ or $\tau = \{\mathsf{nop}, \mathsf{out}, \mathsf{used}, \mathsf{free}\}$	NP-complete	2
$\tau = \{\mathsf{nop}, \mathsf{set}, \mathsf{res}\} \cup \omega$, $\emptyset \neq \omega \subseteq \{\mathsf{used}, \mathsf{free}\}$	NP-complete	3
$\tau = \{\mathsf{nop}, \mathsf{inp}, \mathsf{out}\} \cup \omega$, $\omega \subseteq \{\mathsf{used}, \mathsf{free}\}$	NP-complete	4
$\tau = \{\mathsf{nop}, \mathsf{inp}, \mathsf{res}, \mathsf{swap}\} \cup \omega$, $\omega \subseteq \{\mathsf{used}, \mathsf{free}\}$	NP-complete	4
$\tau = \{\mathsf{nop}, \mathsf{out}, \mathsf{set}, \mathsf{swap}\} \cup \omega$, $\omega \subseteq \{\mathsf{used}, \mathsf{free}\}$	NP-complete	4
$\tau = \{\mathsf{nop}, \mathsf{inp}, \mathsf{set}\} \cup \omega$, $\omega \subseteq \{\mathsf{out}, \mathsf{res}, \mathsf{swap}, \mathsf{used}, \mathsf{free}\}$	NP-complete	24+8
$\tau = \{\mathsf{nop}, \mathsf{outres}\} \cup \omega$, $\omega \subseteq \{\mathsf{inp}, \mathsf{set}, \mathsf{swap}, \mathsf{used}, \mathsf{free}\}$	NP-complete	24+8

Fig. 1. Summary of the complexity results for boolean net synthesis. Grey colored area: Results of [14] reestablishing the result for flip flop nets [11]. Green colored area: Results of this paper. The last two rows intersect in eight supersets of $\{\mathsf{nop}, \mathsf{inp}, \mathsf{out}, \mathsf{set}, \mathsf{res}\}$ and the eighth row includes the already investigated elementary net systems [1]. Altogether, this paper discovers 67 *new* types with an NP-hard synthesis problem.

This paper is devoted to a computational complexity analysis of the boolean net synthesis problem subject to a target type of nets. Synthesis relative to a specific type of nets τ is the challenge to find for a given *transition system* (TSs, for short) A, a boolean τ-net N whose state graph is isomorphic to A if it exists. The complexity of boolean net synthesis has originally been investigated for elementary net systems [1], where it is NP-complete to decide if general TSs can be synthesized. In [15, 12] this has been confirmed even for strongly restricted input TSs. On the contrary, [11] shows for flip flop nets, simply extending elementary net systems by swap, synthesis is doable in polynomial time. Inspired by these results, it is the (global) goal of our research to obtain a dichotomy result that fully characterizes which synergies of interactions make synthesis intractable or tractable, respectively. After resolving the complexity of synthesis for elementary nets [1, 15, 12] and flip flop nets [11], the next big step towards a complete characterization of boolean net synthesis is taken in [14]. Here, besides the already investigated flip flop nets, 42 further boolean types of nets are covered at one blow, cf. Figure 1. Besides the 128 practically less relevant types without nop, there are 84 nop-afflicted boolean types of nets left where the synthesis' complexity has not been settled, yet.

In this paper, we tackle 67 of them with a common NP-hardness proof scheme and, additionally, reestablish the result for elementary net systems [1]. Except flip flop nets, our result covers all types of nets previously considered in the literature. In particular, we show that synthesis is hard for all types that are a superset of $\{\mathsf{nop}, \mathsf{inp}, \mathsf{out}\}$ that excludes swap and for all supersets of $\{\mathsf{nop}, \mathsf{inp}, \mathsf{set}\}$ and $\{\mathsf{nop}, \mathsf{out}, \mathsf{res}\}$, cf. Figure 1.

Aside from the actual identification of 67 new types with hard net synthesis, this paper's contribution is also a very generic reduction scheme for NP-hardness

proofs of boolean net synthesis. This methodology significantly generalizes preliminary methods that we developed in [15, 12] to derive the hardness of synthesizing elementary net systems from strongly restricted TSs. Unlike those premature approaches, the present solution abstracts from individual types of nets and bases on the throughout analysis of properties gained by available interactions. Moreover, the approach from [14], although again slightly similar, is incompatible with the generic scheme in this paper.

To develop the generic reduction scheme, we deal with the synthesis problem's decision version, called *feasibility*. The reason is that complexity analysis rather works with decision problems than search problems. Instead of really finding a net N with state graph isomorphic to a given TS, it is sufficient for feasibility to just decide if the target type contains N. If feasibility is NP-complete, then synthesis is an NP-hard computational problem with no obvious efficient solutions.

To simplify our argumentation is it meaningful to detach our notions from Petri nets and focus on TSs. For this purpose, we use the well known equality between feasibility and the conjunction of the state separation property (SSP) and the event state separation property (ESSP) [2], which are solely defined on the input TSs. The presented polynomial time reduction scheme translates the NP-complete cubic monotone one-in-three 3-SAT problem [8] into the ESSP of the considered 68 boolean net types. As we also make sure that given boolean expressions φ are transformed to TSs $A(\varphi)$ where the ESSP relative to the considered type implies the SSP, we always show the NP-completeness of the ESSP and feasibility at the same time. Instead of 68 individual proofs, our scheme covers all cases by just three reductions following a common pattern.

Due to space limitation, we are not able to present all proofs. However, all omitted proofs are given in the technical report [13].

2 Preliminary Notions

This section provides short formal definitions of all preliminary notions used in the paper. A *transition system* (TS, for short), $A = (S, E, \delta)$ is a directed labeled graph with nodes S, events E and partial transition function $\delta : S \times E \longrightarrow S$, where $\delta(s, e) = s'$ is interpreted as $s \xrightarrow{e} s'$. An event e *occurs* at a state s, denoted by $s \xrightarrow{e}$, if $\delta(s, e)$ is defined. An *initialized* TS $A = (S, E, \delta, s_0)$ is a TS with a distinct state $s_0 \in S$. TSs in this paper are *deterministic* by design as their state transition behavior is given by a (partial) function. Initialized TSs are also required to make every state *reachable* from s_0 by a directed path.

x	$\mathsf{nop}(x)$	$\mathsf{inp}(x)$	$\mathsf{out}(x)$	$\mathsf{set}(x)$	$\mathsf{res}(x)$	$\mathsf{swap}(x)$	$\mathsf{used}(x)$	$\mathsf{free}(x)$
0	0		1	1	0	1		0
1	1	0		1	0	0	1	

Fig. 2. All interactions in I. An empty cell means that the column's function is undefined on the respective x. The entirely undefined function is missing in I.

A (boolean) *type of net* $\tau = (\{0,1\}, E_\tau, \delta_\tau)$ is a TS such that E_τ is a subset of the boolean interactions: $E_\tau \subseteq I = \{\mathsf{nop}, \mathsf{inp}, \mathsf{out}, \mathsf{set}, \mathsf{res}, \mathsf{swap}, \mathsf{used}, \mathsf{free}\}$. The interactions $i \in I$ are binary partial functions $i : \{0,1\} \to \{0,1\}$ as defined in the listing of Figure 2. For all $x \in \{0,1\}$ and all $i \in E_\tau$ the transition function of τ is defined by $\delta_\tau(x,i) = i(x)$. Notice that I contains all possible binary partial functions $\{0,1\} \to \{0,1\}$ except for the entirely undefined function $\perp$. Even if a type τ includes $\perp$, this event can never occur, so it would be useless. Thus, I is complete for deterministic boolean types of nets, and that means there are a total of 256 of them. By definition, a (boolean) type τ is completely determined by its event set E_τ. Hence, in the following we will identify τ with E_τ, cf. Figure 3. Moreover, for readability, we group interactions by $\mathsf{enter} = \{\mathsf{out}, \mathsf{set}, \mathsf{swap}\}$, $\mathsf{exit} = \{\mathsf{inp}, \mathsf{res}, \mathsf{swap}\}$, $\mathsf{keep}^+ = \{\mathsf{nop}, \mathsf{set}, \mathsf{used}\}$, and $\mathsf{keep}^- = \{\mathsf{nop}, \mathsf{res}, \mathsf{free}\}$.

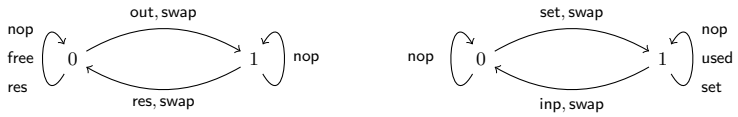


Fig. 3. Left: $\tau = \{\mathsf{nop}, \mathsf{out}, \mathsf{res}, \mathsf{swap}, \mathsf{free}\}$. Right: $\tilde{\tau} = \{\mathsf{nop}, \mathsf{inp}, \mathsf{set}, \mathsf{swap}, \mathsf{used}\}$. τ and $\tilde{\tau}$ are isomorphic. The isomorphism $\phi : \tau \to \tilde{\tau}$ is given by $\phi(s) = 1 - s$ for $s \in \{0,1\}$, $\phi(i) = i$ for $i \in \{\mathsf{nop}, \mathsf{swap}\}$, $\phi(\mathsf{out}) = \mathsf{inp}$, $\phi(\mathsf{res}) = \mathsf{set}$ and $\phi(\mathsf{free}) = \mathsf{used}$.

A boolean Petri net $N = (P, T, M_0, f)$ of type τ, (τ-net, for short) is given by finite and disjoint sets P of places and T of transitions, an initial marking $M_0 : P \longrightarrow \{0,1\}$, and a (total) flow function $f : P \times T \to \tau$. The meaning of a boolean net is to realize a certain behavior by firing sequences of transitions. In particular, a transition $t \in T$ can fire in a marking $M : P \longrightarrow \{0,1\}$ if $\delta_\tau(M(p), f(p,t))$ is defined for all $p \in P$. By firing, t produces the next marking $M' : P \longrightarrow \{0,1\}$ where $M'(p) = \delta_\tau(M(p), f(p,t))$ for all $p \in P$. This is denoted by $M \xrightarrow{t} M'$. Given a τ-net $N = (P, T, M_0, f)$, its behavior is captured by a transition system $A(N)$, called the state graph of N. The state set of $A(N)$ consists of all markings that, starting from initial state M_0, can be reached by firing a sequence of transitions. For every reachable marking M and transition $t \in T$ with $M \xrightarrow{t} M'$ the state transition function δ of A is defined as $\delta(M,t) = M'$.

Boolean net synthesis for a type τ is going backwards from input TS $A = (S, E, \delta, s_0)$ to the computation of a τ-net N with $A(N)$ isomorphic to A, if such a net exists. In contrast to $A(N)$, the abstract states S of A miss any information about markings they stand for. Accordingly, the events E are an abstraction of N's transitions T as they relate to state changes only globally without giving the information about the local changes to places. After all, the transition function $\delta : S \times E \to S$ still tells us how states are affected by events.

To prove net synthesis of τ-nets NP-hard, we show the NP-completeness of the corresponding decision version: *τ-feasibility* is the problem to decide the existence of a τ-net N with $A(N)$ isomorphic to the given TS A. To describe feasibility

without referencing the searched τ-net N, in the sequel, we introduce the *τ-state separation property* (*τ-SSP*, for short) and the *τ-event state separation property* (*τ-ESSP*, for short) for TSs. In conjunction, they are equivalent to τ-feasibility. The following notion of τ-regions allows us to define the announced properties.

A τ-region of a given $A = (S, E, \delta, s_0)$ is a pair (sup, sig) of *support* $sup : S \to S_\tau = \{0, 1\}$ and *signature* $sig : E \to E_\tau = \tau$ where every transition $s \xrightarrow{e} s'$ of A leads to a transition $sup(s) \xrightarrow{sig(e)} sup(s')$ of τ. While a region divides S into the two sets $sup^{-1}(b) = \{s \in S \mid sup(s) = b\}$ for $b \in \{0, 1\}$, the events are cumulated by $sig^{-1}(i) = \{e \in E \mid sig(e) = i\}$ for all available interactions $i \in \tau$. We also use $sig^{-1}(\tau') = \{e \in E \mid sig(e) \in \tau'\}$ for $\tau' \subseteq \tau$.

For a TS $A = (S, E, \delta, s_0)$ and a type of nets τ, a pair of states $s \neq s' \in S$ is *τ-separable* if there is a τ-region (sup, sig) such that $sup(s) \neq \sup(s')$. Accordingly, A has the τ-SSP if all pairs of distinct states from A are τ-separable. Secondly, an event $e \in E$ is called *τ-inhibitable* at a state $s \in S$ if there is a τ-region (sup, sig) where $sup(s) \xrightarrow{sig(e)}$ does not hold, that is, the interaction $sig(e) \in \tau$ is not defined on input $sup(s) \in \{0, 1\}$. A has the τ-ESSP if for all states $s \in S$ it is true that all events $e \in E$ that do not occur at s, meaning $\neg s \xrightarrow{e}$, are τ-inhibitable at s. It is well known from [2] that a TS A is τ-feasible, that is, there exists a τ-net N with $A(N)$ isomorphic to A, if and only if A has τ-SSP *and* the τ-ESSP. Types of nets τ and $\tilde{\tau}$ have an isomorphism ϕ if $s \xrightarrow{i} s'$ is a transition in τ if and only if $\phi(s) \xrightarrow{\phi(i)} \phi(s')$ is one in TS $\tilde{\tau}$. By the following lemma, we benefit from the eight isomorphisms that map nop to nop, swap to swap, inp to out, set to res, used to free, and vice versa:

Lemma 1 (Without proof). *If τ and $\tilde{\tau}$ are isomorphic types of nets then a TS A has the (E)SSP for τ if and only if A has the (E)SSP for $\tilde{\tau}$.*

3 Main Result

Theorem 1 (Main result). *Let $\tau_1 = \{\mathsf{nop}, \mathsf{inp}, \mathsf{out}\}$, $\tau_2 = \{\mathsf{nop}, \mathsf{inp}, \mathsf{res}, \mathsf{swap}\}$, $\tilde{\tau}_2 = \{\mathsf{nop}, \mathsf{out}, \mathsf{set}, \mathsf{swap}\}$, $\tau_3 = \{\mathsf{nop}, \mathsf{inp}, \mathsf{set}\}$ and $\tilde{\tau}_3 = \{\mathsf{nop}, \mathsf{out}, \mathsf{res}\}$. If $\tau = \tau' \cup \omega$ for $\tau' \in \{\tau_1, \tau_2, \tilde{\tau}_2\}$ with $\omega \subseteq \{\mathsf{used}, \mathsf{free}\}$ or $\tau \supseteq \tau_3$ or $\tau \supseteq \tilde{\tau}_3$ then τ-feasibility is NP-complete.*

In total, Theorem 1 covers 68 types, cf. Figure 1, including the *elementary net systems* [1]. It is straight forward that τ-feasibility is a member of NP for all considered type of nets τ: By definition, there are at most $|S|^2$ pairs of states (s, s') to separate and at most $|E| \cdot |S|$ pairs (e, s) of event and state where e has to be inhibited at s. In a non-deterministic computation, one can simply guess and check in polynomial time for all pairs the region that separates s and s', respectively inhibits e at s, or reject the input if such a region does not exist. Hence, for the proof of Theorem 1 it remains to prove τ-feasibility to be NP-hard for all types of nets. Although this demands for 68 NP-hardness proofs, we manage to reduce it to three. Every reduction bases on one scheme using the

cubic monotone one-in-three-3-SAT problem, (P1, for short), which has been shown to be NP-hard in [8]:

CUBIC MONOTONE ONE-IN-THREE-3-SAT (P1)

Instance: negation-free boolean expression $\varphi = \{\zeta_0, \ldots, \zeta_{m-1}\}$ of three-clauses $\zeta_0, \ldots, \zeta_{m-1}$ with variable set $V(\varphi)$, every variable occurs in exactly three clauses

Question: Is there a subset $M \subseteq V(\varphi)$ such that $|M \cap \zeta_i| = 1$ for $i \in \{0, \ldots, m-1\}$?

Starting from a common construction principle, we choose one of our three reductions by a turn-switch σ. In every switch position $\sigma_1, \sigma_2, \sigma_3$, the chosen reduction works for multiple boolean types based on mutually shared interactions and isomorphisms. Before we set out the details, the following subsection introduces our way of easily combining gadget TSs for our NP-completeness proofs.

3.1 Unions of Transition Systems

If $A_0 = (S_0, E_0, \delta_0, s_0^0), \ldots, A_n = (S_n, E_n, \delta_n, s_0^n)$ are (initialized) TSs with pairwise disjoint states (but not necessarily disjoint events) we say that $U(A_0, \ldots, A_n)$ is their *union*. By $S(U)$, we denote the entirety of all states in $A_0, \ldots, A_n$ and $E(U)$ summarizes all events. For a flexible formalism, we allow to build unions recursively: Firstly, we allow empty unions and identify every TS A with the union containing only A, that is, $A = U(A)$. Next, if $U_1 = U(A_0^1, \ldots, A_{n_1}^1), \ldots, U_m = (A_0^m, \ldots, A_{n_m}^n)$ are unions (possibly with $U_i = U()$ or $U_i = A_i$) then $U(U_1, \ldots, U_m)$ is the union $U(A_0^1, \ldots, A_{n_1}^1, \ldots, A_0^m, \ldots, A_{n_m}^n)$.

We lift the concepts of regions, SSP, and ESSP to unions $U = U(A_0, \ldots, A_n)$ as follows: A τ-region (sup, sig) of U consists of $sup : S(U) \to S_\tau$ and $sig : E(U) \to E_\tau$ such that, for all $i \in \{0, \ldots, n\}$, the projections $sup_i(s) = sup(s), s \in S_i$ and $sig_i(e) = sig(e), e \in E_i$ provide a region (sup_i, sig_i) of A_i. U has the τ-SSP if for all different states $s, s' \in S(U)$ of the same TS A_i there is a τ-region (sup, sig) of U with $sup(s) \neq sup(s')$. Moreover, U has the τ-ESSP if for all events $e \in E(U)$ and all states $s \in S(U)$ with $\neg s \xrightarrow{e}$ there is a τ-region (sup, sig) of U such that $\neg sup(s) \xrightarrow{sig(e)}$. Naturally, U is called τ-feasible if it has the τ-SSP *and* τ-ESSP. To merge a union $U = U(A_0, \ldots, A_n)$ into a single TS, we define the joining $A(U)$: If $s_0^0, \ldots, s_0^n$ are the initial states of U's TSs then $A(U) = (S(U) \cup \bot, E(U) \cup \odot \cup \ominus, \delta, \bot_0)$ is a TS with additional connector states $\bot = \{\bot_0, \ldots, \bot_n\}$ and fresh events $\odot = \{\odot_0, \ldots, \odot_n\}$, $\ominus = \{\ominus_1, \ldots, \ominus_n\}$ joining the individual TSs of U by δ as defined in Figure 4.

$$\delta(s,e) = \begin{cases} s_0^i, & \text{if } s = \bot_i \text{ and } e = \odot_i, \\ \bot_{i+1}, & \text{if } s = \bot_i \text{ and } e = \ominus_{i+1}, \\ \delta_i(s,e), & \text{if } s \in S_i \text{ and } e \in E_i, \end{cases}$$

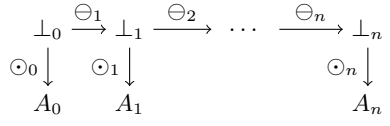


Fig. 4. Left: $A(U)$'s transition function δ. Right: An abstract representation of $A(U)$.

Hence, $A(U)$ puts the connector states into a chain of the events from $\ominus$ and links the initial states of TSs from U to this chain using events from $\odot$. The following lemma certifies the validity of the joining operation for the unions and the types of nets that occur in our reduction scheme.

Lemma 2. *Let τ be a type of nets such that $\mathsf{nop}, \mathsf{inp} \in \tau$ and $\tau \cap \mathsf{enter} \neq \emptyset$. If $U = U(A_0, \ldots, A_n)$ is a union of TSs $A_0, \ldots, A_n$ where, for every event e in $E(U)$, there is at least one state s in $S(U)$ with $\neg s \xrightarrow{e}$ then U has the τ-(E)SSP if and only if $A(U)$ has the τ-(E)SSP.*

Notice that Lemma 2 does not cover all types mentioned in Theorem 1. However, this is not necessary as every type τ from Theorem 1 missed by Lemma 2 is indirectly covered in the Lemma by an isomorphic type $\tilde{\tau}$.

3.2 The General Reduction Scheme

Our general scheme can be set up to a specific reduction by the turn switch σ. In each of its three positions, σ covers a whole collection of net types. Therefore, we simply understand the positions $\sigma_1, \sigma_2, \sigma_3$ as the type sets managed by the respective reductions:

$$\sigma_1 = \{\tau_1 \cup \omega \mid \omega \subseteq \{\mathsf{used}, \mathsf{free}\}\} \cup \{\tau_3 \cup \omega \mid \omega \subseteq \{\mathsf{out}, \mathsf{res}, \mathsf{used}, \mathsf{free}\}$$
$$\sigma_2 = \{\tau_3 \cup \{\mathsf{swap}\} \cup \omega \mid \omega \subseteq \{\mathsf{out}, \mathsf{res}, \mathsf{used}, \mathsf{free}\}\}$$
$$\sigma_3 = \{\tau_2 \cup \omega \mid \omega \subseteq \{\mathsf{used}, \mathsf{free}\}\}$$

The input of our scheme is the switch position $\sigma \in \{\sigma_1, \sigma_2, \sigma_3\}$ and an instance φ of P1. The result is a union U_φ^σ of gadget TSs satisfying the conditions of Lemma 2, that is, U_φ^σ is τ-feasible if and only if $A(U_\varphi^\sigma)$ is τ-feasible for all $\tau \in \sigma$. Moreover, the union U_φ^σ satisfies the following properties:

1. The variables $V(\varphi)$ are a subset of $E(U_\varphi^\sigma)$, the union events.
2. Event $k \in E(U_\varphi^\sigma)$ is to inhibit at state $h_{0,6} \in S(U_\varphi^\sigma)$ and there are events $V = \{v_0, \ldots, v_{3m-1}\} \subseteq E(U_\varphi^\sigma)$ and $W = \{w_0, \ldots, w_{3m-1}\} \subseteq E(U_\varphi^\sigma)$.
3. If $\tau \in \sigma$ and (sup, sig) a τ-region inhibiting k at $h_{0,6}$ then one of the following conditions is true:
   (a) $sig(k) = \mathsf{inp}$ and $V \subseteq sig^{-1}(\mathsf{enter})$ and $W \subseteq sig^{-1}(\mathsf{keep}^-)$,
   (b) $sig(k) = \mathsf{out}$ and $V \subseteq sig^{-1}(\mathsf{exit})$ and $W \subseteq sig^{-1}(\mathsf{keep}^+)$.
4. If (sup, sig) is a region of U_φ^σ satisfying Condition 3.a or Condition 3.b then $M = \{X \in V(\varphi) \mid sig(X) \neq \mathsf{nop}\}$ is a one-in-three model of φ.
5. If φ has a one-in-three model M and $\tau \in \sigma$ then there is a τ-region (sup, sig) with $sig(k) = \mathsf{inp}$ and $sup(h_{0,6}) = 0$. Especially, (sup, sig) inhibits k at $h_{0,6}$ and satisfies Condition 3.a.
6. If $\tau \in \sigma$ and k is τ-inhibitable at $h_{0,6}$ then U_φ^σ has the τ-ESSP and the τ-SSP.

A polynomial time reduction scheme with these properties proves Theorem 1 as the following implications are justified:

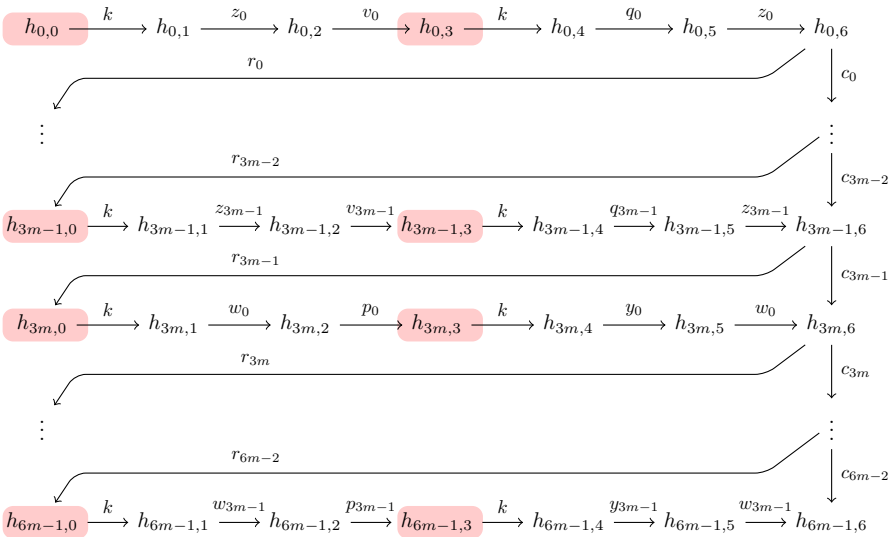
φ is one-in-three satisfiable $\overset{5.}{\Longrightarrow}$ k is τ-inhibitable at $h_{0,6}$ in U_φ^σ $\overset{6.}{\Longrightarrow}$ U_φ^σ has the τ-ESSP & τ-SSP $\overset{\text{def.}}{\Longrightarrow}$ U_φ^σ is τ-feasible $\overset{\text{def.}}{\Longrightarrow}$ U_φ^σ has the τ-ESSP $\overset{\text{def.}}{\Longrightarrow}$ k is τ-inhibitable at $h_{0,6}$ in U_φ^σ $\overset{3./4.}{\Longrightarrow}$ φ is one-in-three satisfiable.

Especially, φ is one-in-three satisfiable if and only if U_φ^σ is τ-feasible. By Lemma 2, this proves NP-hardness of τ-feasibility for all τ in the positions $\sigma_1, \sigma_2, \sigma_3$. Secondly, every remaining type $\tilde{\tau}$ of Theorem 1 is isomorphic to one of the already covered cases τ. Hence, by Lemma 1, this also proves NP-hardness of $\tilde{\tau}$-feasibility which, by feasibility being in NP, justifies Theorem 1.

In the sequel, we develop the reduction of U_φ^σ and show that it satisfies Condition 1-Condition 5. Notice that this proves φ is one-in-three satisfiable if and only if k is inhibitable at $h_{0,6}$. Due to space limitation, the proof that U_φ^σ also has Condition 6 is moved to the technical report [13].

3.3 Details for Condition 2 and Condition 3

To satisfy Condition 2, every union U_φ^σ implements the following transition system H that provides the event k, the state $h_{0,6}$, where $\neg h_{0,6} \xrightarrow{k}$, and the events of $Z = \{z_0, \ldots, z_{3m-1}\}$, $V = \{v_0, \ldots, v_{3m-1}\}$ and $W = \{w_0, \ldots, w_{3m-1}\}$ (the colored areas are to be explained later):



So far Condition 2 is already satisfied. For Condition 3 we observe that, by definition, there are basically four interactions possibly useful for $sig(k)$: inp, out, used, free. The other interactions res, set, swap, nop are defined on both 0 and 1 and, hence, not suitable to inhibit events. H alone generally does not guarantee that a region (sup, sig) inhibiting k at $h_{0,6}$ satisfies Condition 3. Thus, to achieve this goal other gadgets are necessary. By theirs different types (having

different interactions), it depends on σ *which* gadgets are necessary. We proceed step by step and develop the construction for σ_1, σ_2 and σ_3 in the given order. In the sequel, if not explicitly stated otherwise, by (sup, sig) we refer to a τ-region of U^{σ}_{φ}, $\tau \in \sigma$, that inhibits k at $h_{0,6}$. The union $U^{\sigma_1}_{\varphi}$ implements additionally the following two TSs F_0, F_1:

$$F_0 = f_{0,0} \xrightarrow{k} f_{0,1} \xrightarrow{n_0} f_{0,2} \xrightarrow{z_0} f_{0,3} \xrightarrow{k} f_{0,4} \qquad F_1 = f_{1,0} \xrightarrow{q_0} f_{1,1} \xrightarrow{k} f_{1,2}$$

We argue that every region (sup, sig) of $U^{\sigma_1}_{\varphi}$ satisfies Condition 3: If $sig(k) = \mathsf{used}$ then, by definition of used we get $sup(s) = sup(s') = 1$ for every transition $s \xrightarrow{k} s'$. Hence, we have $sup(f_{0,3}) = sup(f_{1,1}) = sup(h_{0,4}) = 1$. By definition of $\mathsf{inp}, \mathsf{res}$ we have that $\xrightarrow{e} s$ and $sig(e) \in \{\mathsf{inp}, \mathsf{res}\}$ implies $sup(s) = 0$. Hence, by $\xrightarrow{z_0} f_{0,3}$ and $\xrightarrow{q_0} f_{1,1}$ we have that $sig(z_0), sig(q_0) \notin \{\mathsf{inp}, \mathsf{res}\}$. Moreover, we observe that $\mathsf{swap} \notin \tau$ for all $\tau \in \sigma_1$. Thus, $sig(z_0), sig(q_0) \in \mathsf{keep}^+$ such that $sup(h_{0,4}) = sup(h_{0,5}) = sup(h_{0,6}) = 1$ which contradicts $\neg sup(h_{0,6}) \xrightarrow{sig(k)}$. Hence, $sig(k) \neq \mathsf{used}$. Similarly, one argues that $sig(k) \neq \mathsf{free}$ implies that $sup(h_{0,6}) = 0$, again a contradiction. Hence, we have that $sig(k) = \mathsf{inp}$ and $sup(h_{0,6}) = 0$ or $sig(k) = \mathsf{out}$ and $sup(h_{0,6}) = 1$.

As a next step, we show that $sig(k) = \mathsf{inp}$ and $sup(h_{0,6}) = 0$ implies $sig(v_0) \in \mathsf{enter}$ and $sig(z_0) \in \mathsf{keep}^-$: By $sig(k) = \mathsf{inp}$ and $\xrightarrow{k} h_{0,1}$ and $h_{0,3} \xrightarrow{k}$ we have that $sup(h_{0,1}) = 0$ and $sup(h_{0,3}) = 1$. By $\xrightarrow{z_0} h_{0,6}$, $sup(h_{0,6}) = 0$ and $\mathsf{swap} \notin \tau$ we have that $sig(z_0) \in \mathsf{keep}^-$. Moreover, by $sup(h_{0,1}) = 0$ and $sig(z_0) \in \mathsf{keep}^-$ it is $sup(h_{0,2}) = 0$ which, by $h_{0,2} \xrightarrow{v_0} h_{0,3}$ and $sup(h_{0,3}) = 1$, implies $sig(v_0) \in \mathsf{enter}$. Notice, that this reasoning purely bases on $sig(k) = \mathsf{inp}$ and $sup(h_{0,6}) = 0$. Similarly, one argues that $sig(k) = \mathsf{out}$ and $sup(h_{0,6}) = 1$ implies $sig(v_0) \in \mathsf{exit}$ and $sig(z_0) \in \mathsf{keep}^+$. $U^{\sigma_1}_{\varphi}$ uses for every $i \in \{0, \ldots, 6m-2\}$ the following TS $G^{c,c}_i$ to transfer the support-value $sup(h_{0,6})$ to the states $h_{1,6}, \ldots, h_{6m-1,6}$:

$$G^{c,c}_i = \begin{array}{ccc} g^{c,c}_{i,0} & \xrightarrow{c_i} & g^{c,c}_{i,1} \\ {\scriptstyle k}\downarrow & & \downarrow{\scriptstyle k} \\ g^{c,c}_{i,2} & \xrightarrow[c_i]{} & g^{c,c}_{i,3} \end{array}$$

By doing so, it transfers the outcome of the latter observation to the events $v_1, \ldots, v_{3m-1}$ and $w_0, \ldots, w_{3m-1}$: If $sig(k) = \mathsf{inp}$, then we have $sup(g^{c,c}_{i,0}) = sup(g^{c,c}_{i,0}) = 1$ and $sup(g^{c,c}_{i,2}) = sup(g^{c,c}_{i,3}) = 0$, that is, $sig(c_i) = \mathsf{nop}$. Symmetrically, if $sig(k) = \mathsf{out}$ then $sig(c_i) = \mathsf{nop}$. Hence, if $sig(k) = \mathsf{inp}$ and $sup(h_{0,6}) = 0$ then $sup(h_{i,6}) = 0$ for all $i \in \{0, \ldots, 6m-1\}$. Perfectly similar to the discussion for z_0 and v_0 we obtain that $V \subseteq sig^{-1}(\mathsf{enter})$ and $W \subseteq sig^{-1}(\mathsf{keep}^-)$, respectively. Symmetrically, $sig(k) = \mathsf{out}$ and $sup(h_{0,6}) = 1$ imply $V \subseteq sig^{-1}(\mathsf{exit})$ and $W \subseteq sig^{-1}(\mathsf{keep}^+)$. Hence, $U^{\sigma_1}_{\varphi}$ satisfies Condition 3.

Unfortunately, if $\sigma \in \{\sigma_2, \sigma_3\}$ then the introduced gadgets F_0, F_1 (alone) are not powerful enough to ensure Condition 3. This is mainly due to the interaction swap. However, we tackle this problem by the application of other gadgets. While, due to theirs different interaction sets, σ_2 and σ_3 have different requirements,

both of $U_\varphi^{\sigma_2}$ and $U_\varphi^{\sigma_3}$ implement for every $i \in \{0, \dots, 6m-2\}$ the gadget $G_i^{c,c}$ and for $i \in \{0, \dots, 3m-1\}$ following gadget TSs $G_i^{\text{-},q}$ and $G_i^{\text{-},y}$:

$$G_i^{\text{-},q} = \begin{array}{ccc} g_{i,0}^{\text{-},q} & \xrightarrow{\text{-}} & g_{i,1}^{\text{-},q} \\ {\scriptstyle k}\downarrow & & \downarrow{\scriptstyle k} \\ g_{i,2}^{\text{-},q} & \xrightarrow[q_i]{} & g_{i,3}^{\text{-},q} \end{array} \qquad G_i^{\text{-},y} = \begin{array}{ccc} g_{i,0}^{\text{-},y} & \xrightarrow{\text{-}} & g_{i,1}^{\text{-},y} \\ {\scriptstyle k}\downarrow & & \downarrow{\scriptstyle k} \\ g_{i,2}^{\text{-},y} & \xrightarrow[y_i]{} & g_{i,3}^{\text{-},y} \end{array}$$

Here and in the sequel, the purpose of underscore-labeled edges is essentially to ensure reachability of the TSs. Every underscore represents an arbitrary unique event occurring only at this edge. For the sake of readability we do not define these events explicitly. Again for readability, we define $Q = \{q_0, \dots, q_{3m-1}\}$ and $Y = \{y_0, \dots, y_{3m-1}\}$. Moreover, $U_\varphi^{\sigma_2}$ adds the TS F_0 (originally introduced for σ_1) and the next TS $G_0^{\text{n},\text{-}}$ while $U_\varphi^{\sigma_3}$ uses also F_0 and the following TS F_2:

$$G_0^{\text{n},\text{-}} = \begin{array}{ccc} g_{0,0}^{\text{n},\text{-}} & \xrightarrow{n_0} & g_{0,1}^{\text{n},\text{-}} \\ {\scriptstyle k}\downarrow & & \downarrow{\scriptstyle k} \\ g_{0,2}^{\text{n},\text{-}} & \xrightarrow[\text{-}]{} & g_{0,3}^{\text{n},\text{-}} \end{array} \qquad F_2 = \begin{array}{ccc} f_{2,0} & \xrightarrow{n_0} & f_{2,1} \\ {\scriptstyle k}\downarrow & & \uparrow{\scriptstyle k} \\ f_{2,2} & \xrightarrow[\text{-}]{} & f_{2,3} \end{array}$$

That $U_\varphi^{\sigma_2}$ and $U_\varphi^{\sigma_3}$ differ in $G_0^{\text{n},\text{-}}$ and F_2, respectively, is mainly due to the fact that the interactions of theirs types have different requirements to satisfy Condition 5 and Condition 6. To argue for $U_\varphi^{\sigma_3}$'s and $U_\varphi^{\sigma_4}$'s functionality, respectively, we firstly show that $sig(k) \in \{\mathsf{inp}, \mathsf{out}\}$ for (sup, sig): If $sig(k) = \mathsf{used}$ then $s \xrightarrow{k} s'$ implies $sup(s) = sup(s') = 1$. Applying this to $F_2, G_0^{\text{n},\text{-}}$ and $G_0^{\text{-},q}$ we obtain that $sig(n_0), sig(q_0) \in \mathsf{keep}^+$. By $sig(n_0) \in \mathsf{keep}^+$ and $sup(f_{0,1}) = 1$ we obtain $sup(f_{0,2}) = 1$ which, by $sup(f_{0,3}) = 1$, implies $sig(z_0) \in \mathsf{keep}^+$. Finally, by $sup(h_{0,3}) = 4$ and $sig(z_0), sig(q_0) \in \mathsf{keep}^+$, we get $sup(h_{0,6}) = 1$ which contradicts $\neg \sup(h_{0,6}) \xrightarrow{sig(k)}$. Similarly, the assumption $sig(k) = \mathsf{free}$ yields the same contradiction. Thus, $sig(k) \in \{\mathsf{inp}, \mathsf{out}\}$.

We argue that $sig(k) = \mathsf{inp}$ and $sup(h_{0,6}) = 0$ implies $V \subseteq sig^{-1}(\mathsf{enter})$ and $W \subseteq sig^{-1}(\mathsf{keep}^-)$: By $sig(k) = \mathsf{inp}$ we get again $sig(c_i) = \mathsf{nop}$ and, thus, $sup(h_{i,6}) = 0$ for all i in question. Moreover, by $sig(k) = \mathsf{inp}$ we get $sup(s) = 0$ for all $\xrightarrow{k} s$. Applying this to $G_i^{\text{-},q}$ and $G_i^{\text{-},y}$ yields $Q, Y \subseteq sig^{-1}(\mathsf{keep}^-)$. By $Q, Y \subseteq sig^{-1}(\mathsf{keep}^-)$ and $sup(h_{i,4}) = 0$ (for all $i \in \{0, \dots, 6m-1\}$) we get $sup(h_{i,5}) = 0$ which, by $sup(h_{i,6}) = 0$, implies $W, Z \subseteq sig^{-1}(\mathsf{keep}^-)$. Finally, by $Z \subseteq sig^{-1}(\mathsf{keep}^-)$ and $sup(h_{i,1}) = 0$ we conclude $sup(h_{i,2}) = 0$ implying with $sup(h_{i,3}) = 1$ that $sig(v_i) \in \mathsf{enter}$ for every $i \in \{0, \dots, 3m-1\}$: $V \subseteq sig^{-1}(\mathsf{enter})$. Symmetrically, $sig(k) = \mathsf{out}$ and $sup(h_{0,6}) = 1$ implies $W \subseteq sig^{-1}(\mathsf{keep}^+)$ and $V \subseteq sig^{-1}(\mathsf{exit})$. Hence, $U_\varphi^{\sigma_2}$ and $U_\varphi^{\sigma_3}$ satisfy Condition 3.

3.4 Details for Condition 1 and Condition 4

Essential for the realization of Condition 1 and Condition 4 is the observation that for every τ-region $R = (sup, sig)$ and path $P = s \xrightarrow{e_1} \dots \xrightarrow{e_n} s'$ the image $R(P) = sup(s) \xrightarrow{sig(e_1)} \dots \xrightarrow{sig(e_n)} sup(s')$ is a path in τ. Moreover, if $sup(s) \neq sup(s') = 0$ then there has to be an event e_i mapped to an interaction of τ that allows a state change in τ: $sig(e_i) \notin \{\mathsf{nop}, \mathsf{used}, \mathsf{free}\}$. Our target is

to exploit this observation in the following way: Firstly, represent every clause $\zeta_i = \{X_{i,0}, X_{i,1}, X_{i,2}\}$ by a path $P_{i,0} = t_{i,0,2} \xrightarrow{X_{i,0}} t_{i,0,3} \xrightarrow{X_{i,1}} t_{i,0,4} \xrightarrow{X_{i,2}} t_{i,0,5}$. Secondly, ensure that region (sup, sig) (inhibiting k at $h_{0,6}$) requires $sup(t_{i,0,2}) \neq sup(t_{i,0,5})$ for every $i \in \{0, \ldots, m-1\}$. Thirdly, ensure that there is exactly one variable event $X \in \{X_{i,0}, X_{i,1}, X_{i,2}\}$ whose image $sig(X)$ allows the necessary state change in τ and that both of the others are mapped to nop. Such a region implies that $M = \{X \in V(\varphi) \mid sig(X) \neq \mathsf{nop}\}$ is a one-in-three model of φ as two different clauses ζ_i, ζ_j are represented by different paths $P_{i,0}, P_{j,0}$. Let's discuss the case that region (sup, sig) satisfies $sup(t_{i,0,2}) = 1$ and $sup(t_{i,0,5}) = 0$. Unfortunately, generally there are many possibilities for the signature of the variable events $X_{i,0}, X_{i,1}, X_{i,2}$ and not even one of them need to be mapped to nop. For example, $sig(X) = \mathsf{res}$ or $sig(X) = \mathsf{swap}$ for $X \in \{X_{i,0}, X_{i,1}, X_{i,2}\}$ would be possible, too.

We attack this problem as follows: Firstly, instead of one path for ζ_i we apply the following three forward-backward paths $P_{i,0}, P_{i,1}, P_{i,2}$ which for every variable event $X \in \{X_{i,0}, X_{i,1}, X_{i,2}\}$ use an additional mirror event x:

$$P_{i,0} = t_{i,0,2} \underset{x_{i,0}}{\overset{X_{i,0}}{\rightleftarrows}} t_{i,0,3} \underset{x_{i,1}}{\overset{X_{i,1}}{\rightleftarrows}} t_{i,0,4} \underset{x_{i,2}}{\overset{X_{i,2}}{\rightleftarrows}} t_{i,0,5} \qquad P_{i,1} = t_{i,1,2} \underset{x_{i,1}}{\overset{X_{i,1}}{\rightleftarrows}} t_{i,1,3} \underset{x_{i,2}}{\overset{X_{i,2}}{\rightleftarrows}} t_{i,1,4} \underset{x_{i,0}}{\overset{X_{i,0}}{\rightleftarrows}} t_{i,1,5}$$

$$P_{i,2} = t_{i,2,2} \underset{x_{i,2}}{\overset{X_{i,2}}{\rightleftarrows}} t_{i,2,3} \underset{x_{i,0}}{\overset{X_{i,0}}{\rightleftarrows}} t_{i,2,4} \underset{x_{i,1}}{\overset{X_{i,1}}{\rightleftarrows}} t_{i,2,5}$$

Notice that, by the arbitrariness of i, we have for every $X \in \{X_0, \ldots, X_{m-1}\} = V(\varphi)$ its unique mirror event $x \in \{x_0, \ldots, x_{m-1}\}$. Moreover, by definition, if $s \xrightarrow{X} s'$ is a an edge (of any forward-backward path) then $s \xleftarrow{x} s'$ is too. The paths $P_{i,0}, P_{i,1}, P_{i,2}$ are similar but the variable events are once and twice left-shifted, respectively. Secondly, we ensure that (sup, sig) also satisfies $sup(t_{i,1,2}) = sup(t_{i,2,2}) = 1$ and $sup(t_{i,1,5}) = sup(t_{i,2,5}) = 0$, that is, sup synchronizes the states $t_{i,0,2}, t_{i,1,2}, t_{i,2,2}$ and the states $t_{i,0,5}, t_{i,1,5}, t_{i,2,5}$, respectively. As a result, we have for every variable event $X \in \{X_{i,0}, X_{i,1}, X_{i,2}\}$ an edge $\xrightarrow{X} s$ such that $sup(s) = 0$. Consequently, we have that $sig(X) \notin \{\mathsf{out}, \mathsf{set}, \mathsf{used}\}$, as $\xrightarrow{i} x$ and $i \in \{\mathsf{out}, \mathsf{set}, \mathsf{used}\}$ requires $x = 1$. Moreover, we also have for every variable event an edge $s \xrightarrow{X}$ such that $sup(s) = 1$, such that $sig(X) \neq \mathsf{free}$. This leaves $\mathsf{nop}, \mathsf{inp}, \mathsf{res}, \mathsf{swap}$ as remaining candidates for $sig(X)$.

As a matter of fact, the construction already ensures that a variable event $X \in \{X_{i,0}, X_{i,1}, X_{i,2}\}$ with $sig(X) \in \{\mathsf{inp}, \mathsf{res}\}$ implies that $sig(Y) = \mathsf{nop}$ for $Y \in \{X_{i,0}, X_{i,1}, X_{i,2}\} \setminus \{Y\}$. We explicitly justify this claim only for $X = X_{i,0}$ as, by symmetry, the cases $X = X_{i,1}$ and $X = X_{i,2}$ are quite similar: By $sig(X_{i,0}) \in \{\mathsf{inp}, \mathsf{res}\}$ we have $sup(t_{i,0,3}) = 0$ which by $sup(t_{i,0,2}) = 1$ and $t_{i,0,2} \xleftarrow{x_{i,0}} t_{i,0,3}$ implies $sig(x_{i,0}) \in \{\mathsf{out}, \mathsf{set}, \mathsf{swap}\}$. Again by $sig(X_{i,0}) \in \{\mathsf{inp}, \mathsf{res}\}$ we have $sup(t_{i,2,4}) = 0$ implying with $sig(x_{i,0}) \in \{\mathsf{out}, \mathsf{set}, \mathsf{swap}\}$ that $sup(t_{i,2,3}) = 1$. By $t_{i,2,2} \xrightarrow{X_{i,2}} t_{i,2,3}$, $sup(t_{i,2,2}) = sup(t_{i,2,3}) = 1$ and $sig(X_{i,2}) \in \{\mathsf{nop}, \mathsf{inp}, \mathsf{res}, \mathsf{swap}\}$ we conclude $sig(X_{i,2}) = \mathsf{nop}$. Furthermore, by $sup(t_{i,1,5}) = 0$ and $sig(x_{i,0}) \in \{\mathsf{out}, \mathsf{set}, \mathsf{swap}\}$ we have $sup(t_{i,1,4}) = 1$. By $sig(X_{i,2}) = \mathsf{nop}$, $t_{i,1,3} \xrightarrow{X_{i,2}} t_{i,1,4}$

and $sup(t_{i,1,4}) = 1$ we obtain that $sup(t_{i,1,3}) = 1$. Finally, by $sup(t_{i,1,2}) = sup(t_{i,1,3}) = 1$, $t_{i,1,2} \xrightarrow{X_{i,1}} t_{i,1,3}$ and $sig(X_{i,1}) \in \{\mathsf{nop}, \mathsf{inp}, \mathsf{res}, \mathsf{swap}\}$ we obtain $sig(X_{i,1}) = \mathsf{nop}$.

So far, we have already argued that for types τ with $\mathsf{swap} \notin \tau$ the following is true: A τ-region that synchronizes $t_{i,0,2}, t_{i,1,2}, t_{i,2,2}$ and $t_{i,0,5}, t_{i,1,5}, t_{i,2,5}$ as announced ensures that there is exactly one variable event of $X_{i,0}, X_{i,1}, X_{i,2}$ with a signature different from nop. This concerns the types which are covered by σ_1.

For the types τ with $\mathsf{swap} \in \tau$, covered by σ_2 and σ_3, it remains to ensure the following: If $X \in \{X_{i,0}, X_{i,1}, X_{i,2}\}$ and $sig(X) = \mathsf{swap}$ then $sig(Y) = \mathsf{nop}$ for $Y \in \{X_{i,0}, X_{i,1}, X_{i,2}\} \setminus \{X\}$. Unfortunately, the previous construction (alone) can not afford this. However, we overcome this problem by another enhancement which we develop in the sequel.

Notice that, by the former discussion, $sig(X) = \mathsf{swap}$ already implies $sig(Y) \notin \{\mathsf{inp}, \mathsf{res}\}$ for $Y \in \{X_{i,0}, X_{i,1}, X_{i,2}\} \setminus \{X\}$. Hence, its simple maths that if $sig(X) = \mathsf{swap}$ then either *all* variable events are mapped to swap or *exactly one* of them: Either we have three changes in τ to get from 1 to 0 or exactly one. Moreover, it is easy to see that if $sig(X) = \mathsf{swap}$ for all $X \in \{X_{i,0}, X_{i,1}, X_{i,2}\}$ then $sig(x) = \mathsf{swap}$ for all $x \in \{x_{i,0}, x_{i,1}, x_{i,2}\}$. Thus, it is sufficient to prevent that *any* x is mapped to swap. To do so, the union $U_\varphi^{\sigma_2}$ installs for every mirror event $x_i \in \{x_0, \ldots, x_{m-1}\}$, that is, especially for $x_{i,0}, x_{i,1}, x_{i,2}$, the gadget TS $G_i^{x,\text{-}}$ which is defined in Figure 5.4. As (sup, sig) satisfies $sig(k) \in \{\mathsf{inp}, \mathsf{out}\}$ we have $sup(g_{i,0}^{x,\text{-}}) = sup(g_{i,1}^{x,\text{-}})$ which implies $sig(x_i) \neq \mathsf{swap}$ for $i \in \{0, \ldots, m-1\}$. Similarly, the union $U_\varphi^{\sigma_3}$ installs for every $i \in \{0, \ldots, m-1\}$ the gadget TS $G_i^{\text{-},x}$ which is defined in Figure 5.8. The reason for these differences between $U_\varphi^{\sigma_2}$ and $U_\varphi^{\sigma_3}$ is again due to theirs different types and the target to satisfy Condition 5 and Condition 6. We will give a detailed explanation later.

Altogether, we have argued that a reduction with these features ensures, that the existence of (sup, sig) implies that $M = \{X \in V(\varphi) \mid sig(X) \neq \mathsf{nop}\}$ is a one-in-three model of φ. Moreover, an analogous argument shows that $sup(t_{i,0,2}) = sup(t_{i,1,2}) = sup(t_{i,2,2}) = 0$ and $sup(t_{i,0,5}) = sup(t_{i,1,5}) = sup(t_{i,2,5}) = 1$ also implies that exactly one variable event (per clause) has a signature different from nop. But *how* can we ensure that the states $t_{i,0,2}, t_{i,1,2}, t_{i,2,2}$ and the states $t_{i,0,5}, t_{i,1,5}, t_{i,2,5}$ become synchronized? To achieve this, we enhance $P_{i,0}, P_{i,1}, P_{i,2}$ as follows: If $\sigma \in \{\sigma_1, \sigma_2\}$ then we enhance $P_{i,0}, P_{i,1}, P_{i,2}$ to $T_{i,0}, T_{i,1}, T_{i,2}$ in accordance to Figure 5.1- Figure 5.3, respectively. $T_{i,0}, T_{i,1}$ and $T_{i,2}$ apply the event k and the events $v_{3i}, v_{3i+1}, v_{3i+2}$ and $w_{3i}, w_{3i+1}, w_{3i+2}$. To make it clear: Besides the corresponding gadgets introduced in Section 3.3, $U_\varphi^{\sigma_1}$ and $U_\varphi^{\sigma_2}$ (additionally) implement the TSs $T_{i,0}, T_{i,1}$ and $T_{i,2}$ for every $i \in \{0, \ldots, m-1\}$. Moreover, $U_\varphi^{\sigma_2}$ implements also $G_i^{x,\text{-}}$ for every $i \in \{0, \ldots, m-1\}$.

The region (sup, sig) uses the latest enhancement as follows: As already discussed in the former section, $sig(k) = \mathsf{inp}$ (and $sup(h_{0,6}) = 0$) imply $V \subseteq sig^{-1}(\mathsf{enter})$ and $W \subseteq sig^{-1}(\mathsf{keep}^-)$. Moreover, by $sig(K) = \mathsf{inp}$, we have $sup(t_{i,0,1}) = sup(t_{i,1,1}) = sup(t_{i,2,1}) = 0$. Altogether, this implies $sup(t_{i,0,2}) = sup(t_{i,1,2}) = sup(t_{i,2,2}) = 1$ and $sup(t_{i,0,5}) = sup(t_{i,1,5}) = sup(t_{i,2,5}) = 0$.

Moreover, if $sig(k) = \mathsf{out}$ then $V \subseteq sig^{-1}(\mathsf{exit})$, implying $sup(t_{i,0,2}) = sup(t_{i,1,2}) = sup(t_{i,2,2}) = 0$, and $W \subseteq sig^{-1}(\mathsf{keep}^+)$, implying $sup(t_{i,0,5}) = sup(t_{i,1,5}) = sup(t_{i,2,5}) = 1$. As discussed, this implies a one-in-three model of φ.

For $U_\varphi^{\sigma_3}$ we need a slightly different construction: The union $U_\varphi^{\sigma_3}$ implements for every $i \in \{0, \dots, m-1\}$ instead of $T_{i,0}, T_{i,1}, T_{i,2}$ the transition system $T'_{i,0}, T'_{i,1}, T'_{i,2}$ defined in Figure 5.4-Figure 5.6, respectively. $T'_{i,0}, T'_{i,1}, T'_{i,2}$ also implement $P_{i,0}, P_{i,1}, P_{i,2}$ but switch the position of $v_{3i}, v_{3i+1}, v_{3i+2}$ with $w_{3i}, w_{3i+1}, w_{3i+2}$, respectively. By symmetry, the arguments that $U_\varphi^{\sigma_3}$ satisfies Condition 4 are similar to the ones for $U_\varphi^{\sigma_1}$ and $U_\varphi^{\sigma_2}$.

Altogether, we have argued that U_φ^{σ} satisfies Condition 1 and Condition 4 for $\sigma \in \{\sigma_1, \sigma_2, \sigma_3\}$. In the next section we talk about the fifth condition. By doing so, we also justify for why $U_\varphi^{\sigma_3}$'s construction different to $U_\varphi^{\sigma_2}$. An essential reason for this is that we have to fulfill both: Condition 4 *and* Condition 5.

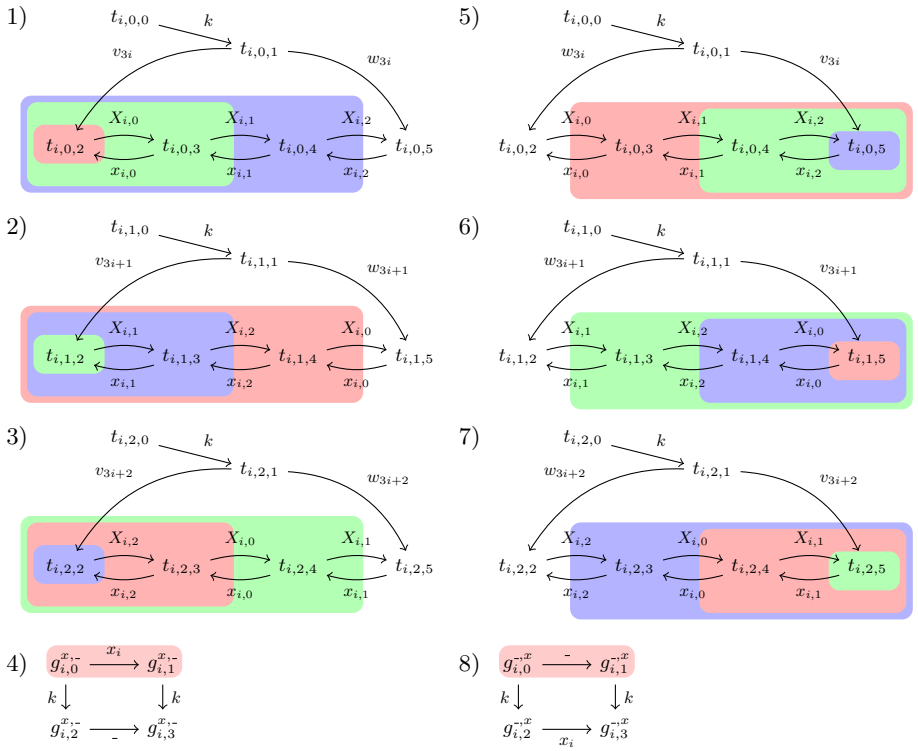


Fig. 5. (1-3) $T_{i,0}, T_{i,1}, T_{i,2}$, (4) $G_i^{x,\text{-}}$, (5-7) $T'_{i,0}, T'_{i,1}, T'_{i,2}$, (8) $G_i^{\text{-},x}$. (1-3,5-7): The colored areas mark the supports of the three possible regions for Condition 5 as described in Section 3.5: red: $X_{i,0} \in M$, green: $X_{i,1} \in M$ and blue: $X_{i,2} \in M$. The (initial) states $t_{i,0,0}, t_{i,1,0}$ and $t_{i,2,0}$ are mapped to 1 for all cases. (4,8): The colored areas define the support of (sup, sig) for Condition 5 as given in Section 3.5.

3.5 Details for Condition 5

For Condition 5, we start from a given one-in-three model M of φ and show that U_φ^σ allows a region (sup, sig) such that $sig(k) = \mathsf{inp}$ and $sup(h_{0,6}) = 0$ and $V \subseteq sig^{-1}(\mathsf{enter})$ and $W \subseteq sig^{-1}(\mathsf{keep}^-)$.

For a start, let's restrict to σ_1, σ_2 and the TSs $T_{i,0}, T_{i,1}T_{i,2}$ implemented by $U_\varphi^{\sigma_1}, U_\varphi^{\sigma_2}$ and $G_i^{x,\text{-}}$ used by $U_\varphi^{\sigma_2}$ where $i \in \{0, \ldots, m-1\}$: We define $sig(k) = \mathsf{inp}$, $sig(v_{3i}), sig(v_{3i+1}), sig(v_{3i+2}) \in \mathsf{enter}$ and $sig(w_{3i}), sig(w_{3i+1}), sig(w_{3i+2}) \in \mathsf{keep}^-$, $i \in \{0, \ldots, m-1\}$. This requires $sup(t_{i,0,2}) = sup(t_{i,1,2}) = sup(t_{i,2,2}) = 1$ and $sup(t_{i,0,5}) = sup(t_{i,1,5}) = sup(t_{i,2,5}) = 0$. That is, for every $T_{i,0}, T_{i,1}, T_{i,2}$ we have a path from 1 to 0 labeled by $X_{i,0}, X_{i,1}, X_{i,2}$ and a path from 0 to 1 labelled by $x_{i,0}, x_{i,1}, x_{i,2}$. For the former path, we define $sig(X) = \mathsf{inp}$ if $X \in M$ and $sig(X) = \mathsf{nop}$ if $Y \in V(\varphi) \setminus M$. Accordingly, for the latter path we let $sig(x) \in \mathsf{enter}$ if $sig(X) = \mathsf{inp}$ and, otherwise $sig(x) = \mathsf{nop}$, cf. Figure 5.1-Figure 5.3: The red area sketches the case $sig(X_{i,0}) = \mathsf{inp}$, the green area $sig(X_{i,1}) = \mathsf{inp}$ and the blue area $sig(X_{i,2}) = \mathsf{inp}$. States within the corresponding colored area are mapped to 1 the others to 0, respectively. If $\sigma = \sigma_1$ then, by Figure 5.1-Figure 5.3 and by recalling that every variable occurs exactly three times in exactly three clauses, it is easy to see that this yields a well defined region. Considering $T_{i,0}, T_{i,1}, T_{i,2}$ alone, this is also true for σ_2. However, for σ_2 we also need to take the TSs $G_0^{x,\text{-}}, \ldots, G_{m-1}^{x,\text{-}}$ into account. Here, by $sig(k) = \mathsf{inp}$ we have that $sup(g_{i,0}^{x,\text{-}}) = sup(g_{i,1}^{x,\text{-}}) = 1$ such that $sig(x_i) \in \mathsf{keep}^+$. Hence, we need that $sig(x_i) \in \mathsf{enter} \cap \mathsf{keep}^+$, that is, $sig(x_i) = \mathsf{set}$. Fortunately, *all* types $\tau \in \sigma_2$ has the property $\mathsf{set} \in \tau$. We argue that (sup, sig) is extendable to $U_\varphi^{\sigma_1}$ and $U_\varphi^{\sigma_2}$ and their other TSs: With the help of the colored areas of the introduced TSs we extend (sup, sig) as follows. For every implemented gadget TS, the red colored area refers to states s such that $sup(s) = 1$, otherwise, $sup(s) = 0$. If $s \xrightarrow{e} s'$ where s is in a red colored area and s' is not, define $sig(e) = \mathsf{inp}$. Similarly, if s' is in a red colored area and s is not, define $sig(e) \in \mathsf{enter}$. Finally, if either both of s, s' are red or both not then define $sig(e) = \mathsf{nop}$. One easily verifies that this yields a well defined region such such that $U_\varphi^{\sigma_1}, U_\varphi^{\sigma_2}$ satisfy Condition 5.

Unfortunately, as $\mathsf{set} \notin \tau$ for $\tau \in \sigma_3$ this region *can not* exist for σ_3 which is one reason why σ_3 needs another construction. For $U_\varphi^{\sigma_3}$ we obtain a corresponding region as follows: We define $sig(k) = \mathsf{inp}$ and $V \subseteq sig^{-1}(\mathsf{enter})$ and $W \subseteq sig^{-1}(\mathsf{keep}^-)$ which requires $sup(t_{i,0,2}) = sup(t_{i,1,2}) = sup(t_{i,2,2}) = 0$ and $sup(t_{i,0,5}) = sup(t_{i,1,5}) = sup(t_{i,2,5}) = 1$. That is, for every $T'_{i,0}, T'_{i,1}, T'_{i,2}$ we have a path from 0 to 1 which is labeled by $X_{i,0}, X_{i,1}, X_{i,2}$ and a path from 1 to 0 labelled by $x_{i,0}, x_{i,1}, x_{i,2}$. For the former path, we define $sig(X) = \mathsf{swap}$ if $X \in M$ and $sig(X) = \mathsf{nop}$ if $Y \in V(\varphi) \setminus M$, cf. Figure 5.4-Figure 5.7. Accordingly, for the latter path we let $sig(x) = \mathsf{res}$ if $sig(X) = \mathsf{swap}$ and, otherwise $sig(x) = \mathsf{nop}$. We take the TSs $G_0^{\text{-},x}, \ldots, G_{m-1}^{\text{-},x}$ into account and, by $sig(k) = \mathsf{inp}$ we have $sup(g_{i,2}^{\text{-},x}) = sup(g_{i,3}^{\text{-},x}) = 0$ which is compatible with $sig(x_i) = \mathsf{res}$. The extension of (sup, sig) to the remaining gadgets of $U_\varphi^{\sigma_3}$ works analogously to $U_\varphi^{\sigma_2}$, where border-crossing events from 0 to 1 are mapped to swap (instead of set). Altogether, we have proven that U_φ^σ satisfies Condition 5 for $\sigma \in \{\sigma_1, \sigma_2, \sigma_3\}$.

4 Concluding Remarks

In this paper we present a proof scheme to show the NP-hardness of synthesis, for 68 boolean Petri net types. Together with our previous work from [14], this already makes 111 out of 256 boolean net types where the complexity of synthesis has been figured out. In fact, with respect to the practically more relevant nop-afflicted types of nets, there are only 17 cases left open. However, in view of our main target (dichotomy result) it remains for future work to characterize the synthesis complexity for all the remaining 145 types.

The NP-hardness results of the investigated synthesis problems motivates the search for (fixed-parameter) tractable special cases. There are at least two ways to (possibly) obtain such cases: Firstly, one can put (structural) restrictions on the τ-net N to be synthesized (the output) and require, for example, that N has to be *free-choice* or a *marked graph* [4]. To investigate the impact of such output-restrictions on the complexity of boolean net synthesis is certainly an interesting direction for further research.

Secondly, one can put restrictions on the TS from which N is to synthesize (the input). One of the most obvious parameters here is the TS's *state degree* g, that is, the maximum number of ingoing and outgoing edges at a state. Actually, TSs of benchmarks of the digital hardware design community often have a low state degree [5]. However, our result show that it is very unlikely that feasibility is (fixed-parameter) tractable for parameter g: All gadget TSs have at most two ingoing and at most two outgoing edges per state, that is $g \leq 2$, and this property is preserved by the joining-operation for $A(U_\varphi^\sigma)$. Clearly, this leaves us with the question if synthesis becomes tractable if $g \leq 1$. On the one hand, that this can not generally be true has been shown in [15]. On the other hand, one observes that our reduction's functionality heavily bases on the ability to prevent a res-, set- and swap-signature of certain events. The core gadgets here are $T_{i,0}, T_{i,1}, T_{i,2}$ (implementing the paths $P_{i,0}, P_{i,1}, P_{i,2}$) and the TSs $G_i^{x,\text{-}}$ and $G_i^{\text{-},x}$. These gadgets are reliant on the possibility to have two ingoing and outgoing edges per state. Hence, the case $g \leq 1$ is an object worth to study in future work, at least for the discussed types satisfying $\tau \cap \{\mathsf{set}, \mathsf{res}, \mathsf{swap}\} \neq \emptyset$.

References

1. Badouel, E., Bernardinello, L., Darondeau, P.: The synthesis problem for elementary net systems is np-complete. Theor. Comput. Sci. **186**(1-2), 107–134 (1997). https://doi.org/10.1016/S0304-3975(96)00219-8
2. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Texts in Theoretical Computer Science. An EATCS Series, Springer (2015). https://doi.org/10.1007/978-3-662-47967-4
3. Badouel, E., Darondeau, P.: Trace nets and process automata. Acta Inf. **32**(7), 647–679 (1995). https://doi.org/10.1007/BF01186645
4. Best, E.: Structure theory of petri nets: the free choice hiatus. In: Advances in Petri Nets. Lecture Notes in Computer Science, vol. 254, pp. 168–205. Springer (1986). https://doi.org/10.1007/BFb0046840

5. Cortadella, J.: Private correspondance (2017)
6. Kleijn, J., Koutny, M., Pietkiewicz-Koutny, M., Rozenberg, G.: Step semantics of boolean nets. Acta Inf. **50**(1), 15–39 (2013). https://doi.org/10.1007/s00236-012-0170-2
7. Montanari, U., Rossi, F.: Contextual nets. Acta Inf. **32**(6), 545–596 (1995). https://doi.org/10.1007/BF01178907
8. Moore, C., Robson, J.M.: Hard tiling problems with simple tiles. Discrete & Computational Geometry **26**(4), 573–590 (2001). https://doi.org/10.1007/s00454-001-0047-6
9. Pietkiewicz-Koutny, M.: Transition systems of elementary net systems with inhibitor arcs. In: ICATPN. Lecture Notes in Computer Science, vol. 1248, pp. 310–327. Springer (1997). https://doi.org/10.1007/3-540-63139-9_43
10. Rozenberg, G., Engelfriet, J.: Elementary net systems. In: Petri Nets. Lecture Notes in Computer Science, vol. 1491, pp. 12–121. Springer (1996). https://doi.org/10.1007/3-540-65306-6_14
11. Schmitt, V.: Flip-flop nets. In: STACS. Lecture Notes in Computer Science, vol. 1046, pp. 517–528. Springer (1996). https://doi.org/10.1007/3-540-60922-9_42
12. Tredup, R., Rosenke, C.: Narrowing down the hardness barrier of synthesizing elementary net systems. In: CONCUR. LIPIcs, vol. 118, pp. 16:1–16:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2018). https://doi.org/10.4230/LIPIcs.CONCUR.2018.16
13. Tredup, R., Rosenke, C.: Towards completely characterizing the complexity of boolean nets synthesis. CoRR **abs/1806.03703** (2018), http://arxiv.org/abs/1806.03703
14. Tredup, R., Rosenke, C.: The complexity of synthesis for 43 boolean petri net types. In: Theory and Applications of Models of Computation. Theoretical Computer Science and General Issues, vol. 11436. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-14812-6
15. Tredup, R., Rosenke, C., Wolf, K.: Elementary net synthesis remains np-complete even for extremely simple inputs. In: Petri Nets. Lecture Notes in Computer Science, vol. 10877, pp. 40–59. Springer (2018). https://doi.org/10.1007/978-3-319-91268-4_3

Reviving Token-based Replay: Increasing Speed While Improving Diagnostics

Alessandro Berti[0000−0003−1830−4013] and Wil van der Aalst[0000−0002−0955−6940]

Process and Data Science group, Lehrstuhl für Informatik 9 52074 Aachen, RWTH Aachen University, Germany

Abstract. Token-based replay used to be the standard way to conduct conformance checking. With the uptake of more advanced techniques (e.g., alignment based), token-based replay got abandoned. However, despite decomposition approaches and heuristics to speed-up computation, the more advanced conformance checking techniques have limited scalability, especially when traces get longer and process models more complex. This paper presents an improved token-based replay approach that is much faster and scalable. Moreover, the approach provides more accurate diagnostics that avoid known problems (e.g., "token flooding") and help to pinpoint compliance problems. The novel token-based replay technique has been implemented in the PM4Py process mining library. We will show that the replay technique outperforms state-of-the-art techniques in terms of speed and/or diagnostics.

Keywords: Log-Model Replay · Process Diagnostics · Localized Conformance Checking

1 Introduction

The importance of conformance checking is growing as is illustrated by the new book on conformance checking [8] and the Gartner report which states "we see a significant trend toward more focus on conformance and enhancement process mining types" [9]. Conformance checking aims to compare an event log and a process model in order to discover deviations and obtain diagnostics information [15]. Deviations are related to process executions not following the process model (for example, the execution of some activities may be missing, or the activities are not happening in the correct order), and are usually associated with higher throughput times and lower quality levels. Hence, it is important to detect them, understand their causes and re-engineer the process in order to avoid such deviations. A prerequisite for both conformance checking and performance analysis is a replay technique, that relates and compares the behavior observed in the log with the behavior observed in the model. Different replay techniques have been proposed, like *token-based replay* [17] and *alignments* [8, 6]. In recent years, alignments have become the standard-de-facto technique since they are able to find an optimal match between the process model and a process execution contained in the event log. Unfortunately, their performance on complex process models and large event logs is poor.

Token-based replay used to be the default technique, but has been almost abandoned in recent years, because the handling of invisible transitions, that are contained in the output models of algorithms like the heuristics miner or the inductive miner, is based on heuristics and the technique suffer of several know drawbacks. For example, models may get flooded with tokens in highly non-conforming executions, enabling unwanted parts of the process model and hampering the overall fitness evaluation. Moreover, detailed diagnostics have been introduced only for alignments.

In this paper, a revival of token-based replay is proposed by addressing some of the weaknesses of traditional token-based replay techniques. The new approach is supported by the PM4Py process mining library[1].

The remainder of the paper is organized as follows: in Section 2 an introduction to token-based replay and alignments is provided. Section 3 presents the novel approach which modifies the original technique and uses a different implementation strategy. Section 4 proposes different ways to localize conformance checking both prior (simplifying the model, reducing the complexity and the time required to do token-based replay) and after the replay (evaluating which elements of the Petri net are used and/or have encountered problems during the replay operation). In Section 5, additional diagnostics are introduced based on the localized replay output. Section 6 concludes the paper.

2 Background and Related Work

Petri nets are the most widely used process model in process mining frameworks: popular discovery algorithms like the alpha miner and the inductive miner (through conversion of the resulting process tree) can produce Petri nets. An accepting Petri net is a Petri net along with a final marking.

Definition 1 (Accepting Petri nets). *A (labeled, marked) accepting Petri net is a net of the form* $PN = (P, T, F, W, M_0, M_F, l)$*, which extends the elementary net so that:*

- (P, T, F) *is a net (*P *and* T *are disjoint finite sets of places and transitions;* $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs).*
- $W : F \to \mathbb{N}$ *is an arc multiset, so that the count (or weight) for each arc is a measure of the arc multiplicity.*
- $M_0 : P \to \mathbb{N}$ *is the initial marking*[2]*.*
- $M_F : P \to \mathbb{N}$ *is the final marking.*
- $l : T \to \sum \cup \{\tau\}$ *is a labeling function that assigns to each transition* $t \in T$ *either a symbol from* $\sum$ *(the set of labels) or the empty string* τ*.*

The *preset* of a place, $\bullet p$, is the set of all transitions $t \in T$ such that $(t, p) \in F$. The *postset* of a place, $p\bullet$, is the set of all transitions $t \in T$ such that $(p, t) \in F$. The preset and postset of a transition could be defined in a similar way. A

[1] The official website of the library is http://www.pm4py.org

[2] A marking $M : P \to \mathbb{N}$ is a place multiset.

transition t is said to be *visible* if $l(t) \in \sum$; is said to be *hidden* if $l(t) = \tau$. If for all $t \in T$ such that $l(t) \neq \tau$, $|\{t' \in T | l(t') = l(t)\}| = 1$, then the Petri net contains *unique visible* transitions; otherwise, it contains *duplicate* transitions. The initial marking is corresponding the initial state of a process execution. Process discovery algorithms may associate also a final marking to the Petri net, that is the state in which the process execution should end. The execution semantics of a Petri net is the following:

- A transition $t \in T$ is *enabled* (it may *fire*) in M if there are enough tokens in its input places for the consumptions to be possible, i.e. iff $\forall s \in \bullet t : M(s) \geq W(s,t)$.
- Firing a transition $t \in T$ in marking M consumes $W(s,t)$ tokens from each of its input places s, and produces $W(t,s)$ tokens in each of its output places s.

For a process supported by an information system, an event log is a set of cases, each one corresponding to a different execution of the process. A case contains the list of events that are executed (in the information system) in order to complete the case. To each case and event, some attributes can be assigned (e.g. the activity and the timestamp at the event level). A classification of the event is a string describing the event (e.g. the activity is a classification of the event). For each case, given a classification function, the corresponding trace is the list of classifications associated with the events of the case.

The application of token-based replay is done on a trace of an event log and an accepting Petri net. The output of the replay operation is a list of transitions enabled during the replay, along with some numbers: c is the number of consumed tokens (during the replay), p is the number of produced tokens, m is the number of missing tokens, r is the number of remaining tokens. At the start of the replay, it is assumed that the tokens in the initial marking are inserted by the environment, increasing p accordingly (for example, if the initial marking consists of one token in one place, then the replay starts with $p = 1$). The replay operation considers, in order, the activities of the trace. In each step, the set of enabled transitions in the current marking is retrieved. If there is a transition corresponding to the current activity, then it is fired, a number of tokens equal to the sum of the weight of input arcs is added to c, and a number of tokens equal to the sum of the weight of output arcs is added to p. If there is not a transition corresponding to the current activity enabled in the current marking, then a transition in the model corresponding to the activity is searched (if there are duplicate corresponding transitions, then [17] provides an algorithm to choose between them). Since the transition could not fire in the current marking, the marking is modified by inserting the token(s) needed to enable it, and m is increased accordingly. At the end of the replay, if the final marking is reached, it is assumed that the environment consumes the tokens from the final marking, and c is increased accordingly. If the marking reached after the replay of the trace is different from the final marking, then missing tokens are inserted and remaining tokens r are set accordingly.

The following relations hold during the replay: $c \leq p + m$ and $m \leq c$. The relation $p + m = c + r$ holds at the end of the replay. A fitness value could be calculated for the trace as:

$$f_\sigma = \frac{1}{2}\left(1 - \frac{m}{c}\right) + \frac{1}{2}\left(1 - \frac{r}{p}\right)$$

For each case L_i of the event log L, let c_i be the number of consumed tokens, p_i the number of produced tokens, m_i the number of missing tokens and r_i the number of remaining tokens. Then, the following formula calculates the fitness at the log level

$$f_L = \frac{1}{2}\left(1 - \frac{\sum_{L_i \in L} m_i}{\sum_{L_i \in L} c_i}\right) + \frac{1}{2}\left(1 - \frac{\sum_{L_i \in L} r_i}{\sum_{L_i \in L} p_i}\right)$$

This quantity is different from the average of fitness values at trace level. When, during the replay, a transition corresponding to the activity could not be enabled, and invisible transitions are present in the model, a technique is deployed to traverse the state space (see [17]) and possibly reach a marking in which the given transition is enabled. A heuristic (see [17]) that uses the shortest sequence of invisible that enables a visible task is proposed. This heuristic tries to minimize the possibility that the execution of an invisible transition interferes with the future firing of another activity.

A well-known problem for token-based replay is the *token flooding problem* [8]. Indeed, when the case differs much from the model, and a lot of missing tokens are inserted during the replay, it happens that also a lot of tokens remain unused and many transitions are enabled. This leads to misleading diagnostics because unwanted parts of the model may be activated, and so the fitness value for highly problematic executions may be too high. To illustrate the token-flooding problem consider a process model without concurrency (only loops, sequences, and choices) represented as a Petri net. At any stage, there should be at most one token in the Petri net. However, each time there is a deviation, a token may be added resulting in a state which was never reachable from the initial state.

The original token-based replay implementation [17] was only implemented in earlier versions of the ProM framework (ProM4 and ProM5) and proposes localized metrics on places of the Petri net that help to understand which parts of the model are more problematic. To improve performance in the original implementation, a preprocessing step could be used to group cases having the same trace. In this way, the replay of a unique trace is done once by the token-based replay. Alternatively, more ad-hoc token-based replay approaches were used by the heuristic miner and the genetic miner. In the latter approach, the qualities of candidate models are derived. These techniques tend to put multiple dimensions (replay fitness, precision, etc.) into a single fitness measure.

Currently, the standard replay technique on Petri nets is the computation of alignments. There are different approaches on alignments [8, 6]. In the assessment, we are considering the approach described in [6]. Execution speed of

alignments on process models containing a lot of different states may be problematic, although some techniques have been proposed, such as decomposing alignments [2] and recomposing them [10]. Moreover, the approach described in [18] is also helping to handle bigger instances, making the user decide about the granularity of the alignment steps.

3 Improved Token-Based Replay

3.1 Changes to the Approach

The approach proposed in [17] is relatively fast when there are no duplicate or silent transitions. However, in comparison to the alignments, managing invisible transitions may be time-consuming due to the necessary state-space explorations.

The idea proposed in this paper is to perform a pre-processing step in order to store a map of the shortest paths between places, and then use this map when hidden transitions need to be traversed. This saves the time necessary to perform the state-space explorations. Therefore, the proposed approach works with accepting Petri nets that have no invisible transitions with empty preset or postset, since they would not belong to any shortest path between places.

3.2 Preprocessing Step: Shortest Paths Between Places

Given an accepting Petri net $PN = (P, T, F, W, M_0, M_F, l)$, it is possible to define a directed graph $G = (V, A)$ such that the vertices V are the places P of the Petri net, and $A \subseteq P \times P$ is such that $(p_1, p_2) \in A$ if and only if at least one invisible transition connects p_1 to p_2. Then, to each arc $(p_1, p_2) \in A$, a transition $\tau(p_1, p_2)$ could be associated picking one of the invisible transitions connecting p_1 to p_2.

Using an informed search algorithm for traversing the graph G, the shortest paths between nodes are found. These are a sequence of edges $\langle a_1, \dots a_n \rangle$ of minimal length, that correspond to a sequence of transitions $\langle t_1, \dots, t_n \rangle$ using the mapping provided by τ.

Given a marking M such that $M(p_1) > 0$ and $M(p_2) = 0$, a marking M' where $M'(p_2) > 0$ could be reached by firing the sequence $\langle t_1, \dots, t_n \rangle$ that is the shortest path in G between p_1 and p_2. The following subsection will explain how to apply the shortest paths to traverse invisible transitions and reach a marking where a transition is enabled.

3.3 Enabling Transitions

The approach described in this subsection helps to enable a transition t through the traversal of invisible transitions. This helps in avoiding the insertion of missing tokens when an activity needs to be replayed on the model, but no corresponding transition is enabled in the current marking M. Moreover, it helps to avoid time-consuming state-space explorations that are required by the approach proposed in [17].

For a marking M and a transition t, it is possible to define the following sets:

- $\Delta(M,t) = \{p \in {\bullet t} \mid M(p) < W(p,t)\}$ is the set of places that miss some tokens to enable transition t. If the set $\Delta(M,t)$ is not empty, then the transition t could not be enabled in the marking M.
- $\Lambda(M,t) = \{p \in P \mid W(p,t) = 0 \wedge M(p) > 0\}$ is the set of places for which the marking has at least one token and t does not require any of these places to be enabled.

When t is not enabled, the set $\Delta(M,t)$ is not empty. The idea is about using places in $\Lambda(M,t)$ (that are not useful to enable t) and, through the shortest paths, reach a marking M' where t is enabled.

Given a place $p_1 \in \Lambda(M,t)$ and a place $p_2 \in \Delta(M,t)$, if a path exists between p_1 and p_2 in G, then it is useful to see if the corresponding shortest path $\langle t_1, \ldots, t_n \rangle$ could fire in marking M. If that is the case, a marking M' could be reached having at least one token in p_2. However, the path may not be not realizable, or may require a token from one of the input places of t. So, the set $\Delta(M',t)$ may be smaller than $\Delta(M,t)$, since p_2 gets at least one token. The approach is about considering all the combinations of places $(p_1,p_2) \in \Lambda(M,t) \times \Delta(M,t)$ such that a path exists between p_1 and p_2 in G. These combinations, namely $\{(p_1,p_2),(p_1',p_2'),(p_1'',p_2'')\ldots\}$, are corresponding to some shortest paths $S = \{\langle t_1,\ldots,t_m\rangle, \langle t_1',\ldots,t_n'\rangle, \langle t_1'',\ldots,t_o''\rangle\}$ in G.

The algorithm to enable transition t through the traversal of invisible transitions considers the sequences of transitions in S, ordered by length, and tries to fire them. If the path can be executed, a marking M' is reached, and the set $\Delta(M',t)$ may be smaller than $\Delta(M,t)$, since a place in $\Delta(M,t)$ gets at least one token in M'. However, one of the following situations could happen: 1) no shortest path between combinations of places $(p_1,p_2) \in \Lambda(M,t) \times \Delta(M,t)$ could fire: in that case, we are "stuck" in the marking M, and the token-based replay is forced to insert the missing tokens; 2) a marking M' is reached, but $\Delta(M',t)$ is not empty, hence t is still not enabled in marking M'. In that case, the approach is iterated on the marking M'; 3) a marking M' is reached, and $\Delta(M',t)$ is empty, so t is enabled in marking M'. When situation (2) happens, the approach is iterated. A limit on the number of iterations may be set, and if it is exceeded then the token-based replay proceeds to insert the missing tokens in marking M.

The approach is straightforward when sound workflow nets without concurrency (only loops, sequences, and choices) are considered, since in the considered setting (M marking where transition t is not enabled) both sets $\Lambda(M,t)$ and $\Delta(M,t)$ have a single element, a single combination $(p_1,p_2) \in \Lambda(M,t) \times \Delta(M,t)$ exists and, if a path exists between p_1 and p_2 in G, and the shortest path could fire in marking M, a marking M' will be reached such that $\Delta(M',t) = \emptyset$ and transition t is enabled. Moreover, it performs particularly well on models that are output of popular process discovery algorithms, e.g., inductive miner, heuristics miner, etc., where potentially long chains of invisible (skip, loop) transitions needs to be traversed in order to enable a transition. The approach described in this subsection can also manage duplicate transitions corresponding to the activity that needs to be replayed. In that case, we are looking to enable any one of the

transitions belonging to the set $T_C \subseteq T$ that contains all the transitions corresponding to the activity in the trace. The approach is then applied on the shortest paths between places $(p_1, p_2) \in \cup_{t \in T_C} \Lambda(M,t) \times \Delta(M,t)$. A similar approach can be applied to reach the final marking when, at the end of the replay of a trace, a marking M is reached that is not corresponding to the final marking. In that case, $\Delta = \{p \in P \mid M(p) < M_F(p)\}$ and $\Lambda = \{p \in P \mid M_F(p) = 0 \wedge M(p) > 0\}$. This does not cover the case where the reached marking contains the final marking but has too many tokens.

3.4 Token Flooding Problem

To address the token flooding problem, which is one of the most severe problems when using token-based replay, we propose several strategies. The final goal of these strategies is to avoid the activation of transitions that shall not be enabled, keeping the fitness value low for problematic parts of the model. The common pattern behind these strategies is to determine *superfluous tokens*, that are tokens that cannot be used anymore. During the replay, f (initially set to 0) is an additional variable that stores the number of "frozen" tokens. When a token is detected as superfluous, it is "frozen": that means, it is removed from the marking and f is increased. Frozen tokens, like remaining tokens, are tokens that are produced in the replay but never consumed. Hence, at the end of the replay $p+m = c+r+f$. To each token in the marking, an *age* (number of iterations of the replay for which the token has been in the marking without being consumed) is assigned. The tokens with the highest age are the best candidates for removal. The techniques to detect superfluous tokens are deployed when a transition required the insertion of missing tokens to fire, since the marking would then possibly contain more tokens. One of the following strategies can be used:

1. Using a decomposition of the Petri net in semi-positive invariants [11] or S-components [1] to restrict the set of allowed markings. Considering S-components, each S-component should hold at most 1 token, so it is safe to remove the oldest tokens if they belong to a common S-component.
2. Using place bounds [12]: if a place is bounded to N tokens and during the replay operation the marking contains $M > N$ tokens for the place, the "oldest" tokens according to the age are removed.

3.5 Changes to the Implementation to Improve Performance

The implementation of the approach proposed in [17] has been made more efficient thanks to ideas adopted from the alignments implementation in ProM6 [5]:

1. *Post-fix caching*: a post-fix is the final part of a case. During the replay of a case, the couple marking+post-fix is saved in a dictionary along with the list of transitions enabled from that point to reach the final marking of the

model. For the next replayed cases, if one of them reaches exactly a marking + post-fix setting saved in the dictionary, the final part of the replay could be retrieved from the dictionary.
2. *Activity caching*: activity caching means saving in a dictionary, during the replay of a case, the list of hidden transitions enabled from a given marking to reach a marking where a particular transition is enabled. For the next replayed cases, if one of them reaches a marking + target transition setting saved in the dictionary, then the corresponding hidden transitions are fired accordingly to enable the target transition.

3.6 Evaluation

In this section, the token-based replay (as implemented in the PM4Py library) is assessed, looking at the speed and the output of the replay, against the alignments approach (as implemented in the "Replay a Log on Petri Net for Conformance Analysis" plug-in of ProM6). Alignments produce results that differ from token-based replay, so results are not directly comparable. Both are replay techniques, so the goal of both techniques is to provide information about how much a process execution is fit according to the process model (albeit the fitness measures are defined in a different way, and so are intrinsically different). This is valid in particular for the comparison of execution times: a trace may be judged fitting according to a process model in a significantly lower amount of time using token-based replay in comparison to alignments. If an execution is unfit according to the model, it can also be judged unfit in a significantly lower amount of time. For a comparison between the two approaches, read Section 8.4 of book [8] or consult [16, 3].

Table 1: Performance of PM4Py token-based replayer on real-life logs in comparison to the alignments approach implemented in ProM6 on models extracted by the inductive miner implementation in PM4Py.

Log	**Cases**	**Variants**	**T.I.P4Pys**	**A.I.P6s**	**Speedup**
repairEx	1104	77	0.06	0.2	3.3
reviewing	100	96	0.10	0.4	4.0
bpic2017	42995	16	0.30	1.5	5.0
receipt	1434	116	0.09	0.8	8.9
roadtraffic	150370	231	1.03	5.5	5.3
Billing	100000	1020	1.36	8.0	5.9

In Table 1, an evaluation of the performance of the token-based replayer on real-life logs respectively is provided. Tests have been done on a Intel I7-5500U powered computer with 16 GB DDR4 RAM. The logs can be retrieved from the 4TU log repository[3]. The *T.I.P4Pys* column shows the execution time (in seconds) of the token-based replay implementation in PM4Py on a model extracted by the inductive miner approach on the given log, the *A.I.P6s* column shows the

[3] The logs are available at the URL https://data.4tu.nl/repository/collection:event_logs

execution time of the alignment-based implementation in ProM6 on the same log and model. The Speedup column shows how many times the token-based replay is faster than the alignment-based implementation. For real-life logs and models extracted by the inductive miner, the token-based replay implementation in PM4Py is 5 times faster on average. Even for large logs, the replay time is less than a few seconds.

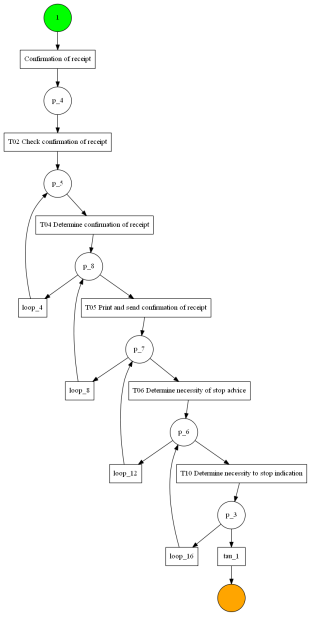


Fig. 1: Model extracted by the inductive miner implementation in PM4Py on a filtered version of the "Receipt phase of an environmental permit application process" event log. Excluding the activities of the log that are not in the model, only 53% of cases of the original log are fitting according to this model.

Table 2: Comparison in token-based replay execution times on models extracted by inductive miner on the given logs with or without postfix and activity caching.

Log	**No caching(s)**	**PC(s)**	**AC(s)**	**PC + AC(s)**
repairEx	0.10	0.08	0.08	*0.06*
reviewing	0.33	0.42	0.14	*0.10*
bpic2017	0.37	0.42	0.30	*0.30*
receipt	0.17	0.15	0.12	*0.09*
roadtraffic	1.58	2.08	1.18	*1.03*
Billing	2.23	1.91	1.45	*1.36*

In Table 2, the effectiveness of the implementation is evaluated in order to understand how the improvements in the implementation contribute to the overall efficiency of the approach. Columns in the table represent the execution time of the replay approach when no caching, only post-fix caching, only activity

caching and the sum of post-fix caching and activity caching is deployed. In the vast majority of logs, the combination of post-fix caching and activity caching provides the best efficiency.

Table 3: Fitness evaluation comparison between the PM4Py token-based replayer (without the token flood cleaning procedure), the token-based replayer in ProM5 and the alignments approach implemented in ProM6 on models extracted by the alpha miner and the inductive miner implementations in PM4Py. Since inductive miner returns a Petri net with perfect fitness, it is expected that the token-based replayer is able to replay the log returning fitness 1.0 for all such combinations. On models extracted by the alpha miner, that do not generally provide perfect fitness, it is expected that the implementation in PM4Py (without the token flood cleaning procedure) is equivalent to the token-based replay implementation in ProM5.

Log	**F.I.PM4Py**	**F.I.P5**	**F.I.P6**	**F.A.PM4Py**	**F.A.P5**
repairEx	1.0	1.0	1.0	0.88	0.88
reviewing	1.0	1.0	1.0	1.0	1.0
bpic2017	1.0		1.0	0.72	
receipt	1.0	1.0	1.0	0.39	0.39
roadtraffic	1.0		1.0	0.62	
Billing	1.0		1.0	0.69	

In Table 3, a comparison between the fitness values recorded by the token-based replay implementation in PM4Py, the token-based replay implementation in ProM5 and the alignments implementation in ProM6 is provided, for both alpha miner and inductive miner models. The meaning of the columns is the following: *F.I.PM4Py* is the fitness value achieved by the token-based replay implementation in PM4Py on a model extracted by the inductive miner approach on the given log, *F.I.P5* is the fitness value achieved by the token-based replay implementation in ProM5 on a model extracted by the inductive miner approach on the given log, *F.I.P6* is the fitness value achieved by the alignments implementation in ProM6 on a model extracted by the inductive miner approach on the given log, *F.A.PM4Py* is the fitness value achieved by the token-based replay implementation in PM4Py on a model extracted by the alpha miner approach on the given log, *F.A.P5* is the fitness value achieved by the token-based replay implementation in ProM5 on a model extracted by the alpha miner approach on the given log. For some real-life logs (bpic2017, roadtraffic, Billing) the token-based replay implementation in ProM5 did not succeed in the replay in 10 minutes (an empty space has been reported in the corresponding columns). Alignments have not been evaluated on the models extracted by alpha miner since it is not assured to have a sound workflow net to start with. The fitness values obtained in Table 3 show that the token-based replay implementation in PM4Py (without the token flood cleaning procedure), on these logs and the models extracted from them by the inductive miner, is as effective in exploring hidden transitions as the token-based replay implementation in ProM5 and the alignments implementation in ProM6.

Table 4: Comparison between the output of the token-based and alignments applied on some logs and the models extracted by the inductive miner implementation in PM4Py on a filtered version of these logs (using the auto filter method of PM4Py). The set of transitions activated in the model by the token-based replay and the alignments for each case has been considered (the middle columns report the overall number of transitions activated in the model by both approaches). Then, a similarity score has been calculated for each case considering the size of the intersection between the two sets and the size of the union. The minimum, maximum, average and median similarity score for the cases in the log has been reported in the right columns of the table, along with the fitness values provided by alignments and token-based replay.

Log	**Tot.T.Al.**	**Tot.T.TR.**	**Min.s.**	**Max.s.**	**Avg.s.**	**Med.s.**	**Fit.al.**	**Fit.tr.**
repairEx	18879	18459	0.538	1.0	0.977	1.0	0.977	0.986
reviewing	2658	2621	0.88	1.0	0.935	0.928	0.900	0.946
bpic2017	171980	171980	1.0	1.0	1.0	1.0	1.0	1.0
roadtraffic	1368414	815326	0.333	1.0	0.591	0.667	0.667	0.758

In order to compare token-based replay and alignments, a comparison between the output of the two approaches has been proposed in Table 4. Some popular logs, that are taken into account also for previous evaluations, are being filtered in order to discover a model (using inductive miner) that is not perfectly fit against the original log. Instead of comparing the fitness values, the comparison is done on the similarity between the set of transitions that were activated in the model during the alignments and the set of transitions that were activated in the model during the token-based replay. The more similar are the two sets, the higher should be the value of similarity. The similarity is calculated as the ratio of the size of the intersection of the two sets and the size of the union of the two sets. This is a simple approach, with some limitations: 1) transitions are counted once during the replay 2) the order in which transitions are activated is not important 3) the number of transitions activated by the alignments is intrinsically higher: while token-based replay could just insert missing tokens and proceed, alignments have to find a path in the model from the initial marking to the final marking, so a higher number of transitions is expected. In Table 4, the meaning of the columns is the following: *Tot.T.Al.* is the number of transitions activated by the alignments approach (a path leading from the initial to the final marking); *Tot.T.TR.* is the number of transititions activated by the token-based replay approach (that is not necessarily a path from the initial to the final marking); *Min.sim.* is the minimum similarity score between the alignments and the token-based replay approach on a case; *Max.sim.* is the maximum similarity score; *Avg.sim.* is the average similarity score; *Med.sim.* is the median similarity score; *Fit.al.* is the fitness value provided by alignments, *Fit.tr.* is the fitness value provided by token-based replay. This comparison, aside fitness values, confirm that the result of the two replay operations, represented as a set

of transitions activated in the model, is very similar, with the exception of the "Road Traffic Fine Management Process" log. For this log, the auto-filtering procedure of PM4Py produces an overly simple model, where token-based replay could survive by inserting missing tokens, but alignments cannot, hence the significantly larger number of transitions activated in the model to explain the behavior observed in the log. Table 4 provides some evidence, aside from fitness values, that the output of the two replay techniques is comparable.

To illustrate the importance of handling the token flooding problem, we consider the "Receipt phase of an environmental permit application process" event log. On this log, a sound workflow net has been extracted which is represented in Figure 1. For this log and model, token flooding occurs because the order of activities is interchanged in some variants of the log. As missing tokens are inserted multiple activities become enabled due to the surplus of tokens. As a result, token-based replay using the original approach yields diagnostics very different from the alignment-based approaches. The original values of average trace fitness and log fitness are 0.92 and 0.93 respectively. Applying the token flooding cleaning procedure, the values go down to 0.86 and 0.87 respectively, because the activation of unwanted parts of the process model is avoided. Albeit the underlying concepts/fitness formula are different (see Section 8.4 of [8]), it may be useful to see that the fitness value provided by alignments is 0.82, so with the token flooding cleaning procedure a more similar value of fitness is obtained.

3.7 Problems Not Addressed

The pre-processing step that stores a map of shortest paths between places is sensible to the presence in the model of implicit/redundant places. Indeed, two models with the same behavior can give different values. However, implicit places can be removed as a pre-processing step on the model. Token-based replay can return a list of transitions that have been activated in the model to replay the trace. However, this does not imply that a path through the model, from the initial to the final marking, is provided, since the insertion of missing tokens can happen if a transition needs to be enabled.

4 Approach: Localization of Conformance Checking Results

Next to providing an overall measure for conformance, conformance checking should also provide diagnostics pinpointing compliance problems. Therefore, we propose two localization approaches:

- The simplification of the original Petri net, in order to make the replay execution speed faster considering only the most problematic parts of a process model.
- The localization of problems encountered during the replay, that permits to understand where deviations happened and their effects.

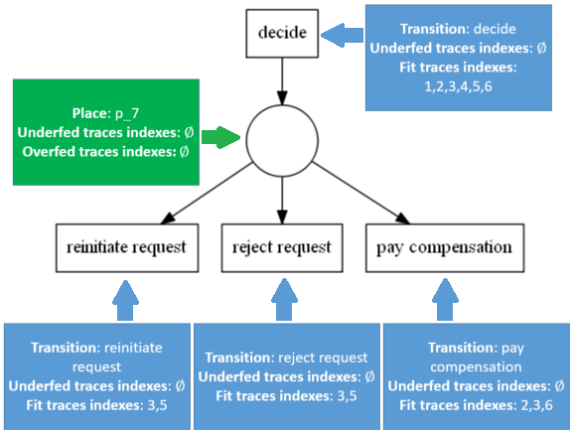


Fig. 2: Petri net, obtained from the "Running example" log, projected on a specific place. This kind of simplification helps to reduce the execution time of the replay operation, and to avoid the token flooding problem. The diagnostics obtained by applying our improved token-based replay are represented.

4.1 Simplification of the Original Petri Net

Replay operations on large models may take too much time. However, it is possible to simplify the model, keeping only parts that are problematic, in order to reduce the execution time of the replay operation.

The decomposition techniques presented in [2, 13, 14, 7] have been used to decompose a Petri net in several subnets for performance reasons. However, for diagnostic purposes an automated decomposition driven only by the model's structure is undesirable. Therefore, we provide the possibility to specify a list of activities in the log and corresponding transitions in the model to check. This is particularly useful when the user knows already which parts of the process are or could be problematic. We also add the possibility to get detailed information about a single element (place or transition) of the Petri net. This information is valuable when comparing fitting executions versus non-fitting executions.

With token-based replay, we propose two simplification approaches to focus attention:

- *Projection on a specific place*: when the preset and the postset of the place are not empty and contain only unique visible transitions, then it is possible to obtain a Petri net containing only the place and the transitions belonging to the preset and the postset. This is particularly useful to detect instances where some tokens are missing / are remaining on the specific place, while not being affected by problems like token flooding. A representation of a Petri net projected on a specific place, obtained from the "Running example" log, is shown in Figure 2.
- *Projection on a set of activities*: it is possible to make selected transitions invisible and retain only the transitions that have a label belonging to a specified set of activities as visible. Then reduction rules are applied to simplify the model with respect to the invisible transitions [4]. This guarantees to get

a Petri net that, for the specific set of activities, has the same language as the original Petri net.

```
from pm4py.objects.log.importer.xes import factory as xes_importer
from pm4py.algo.discovery.inductive import factory as inductive_miner
from pm4py.algo.conformance.tokenreplay import factory
as token_based_replay
from pm4py.evaluation.replay_fitness import factory
as replay_fitness_factory

log = xes_importer.apply("C:\\running-example.xes")
net, im, fm = inductive_miner.apply(log)
aligned_traces = token_based_replay.apply(log, net, im, fm)
fitness = replay_fitness_factory.apply(log, net, im, fm)
```

Fig. 3: Example PM4Py code to apply token-based replay to a log and an accepting Petri net.

4.2 Localization of the Replay Results

Localizing fitness issues in the process model is an essential step in the provision of more detailed diagnostics. The approach described in [17] already provided some diagnostics aimed at localizing the problem:

- *Place underfedness*: when missing tokens are inserted in the place during the replay operation of a case, the place is signed as underfed (it has fewer tokens than needed at some stage) for the specific case.
- *Place overfedness*: when remaining tokens are in the place after the end of the replay of a case, the place is signed as overfed (it has more tokens than needed) for the specific case.

Table 5: Localization of the replay result at place level on the filtered model, represented in Figure 1, obtained from the "Receipt phase" log (only places with problems have been reported).

Place	**# Cases Underfed**	**# Cases Overfed**
p_8	1	0
p_4	35	0
p_7	521	0

To introduce additional localized diagnostics at the transition level, it is important to notice that, when the transition is fired during the replay of a case, is possible to register the *current case status*, for example recording all values of the attributes of the current and of the previous events of the case. The easiest option is to keep a single value for each attribute, that is corresponding to the value of the last occurrence of the given attribute. So, the following localized information could be introduced at the transition level:

- *Transition underfedness*: some tokens needed to fire the transition are missing. It is possible to flag a transition as underfed for the specific case, saving also the status of the case when the transition has been fired.

– *Transition fitness*: the transition could be fired regularly. In this case, it is possible to save the status of the case when the transition has been fired.

It is important also the save information for events with an activity that is not corresponding to any transition in the model. This could be done saving the current case status when such activities happen.

Table 6: Localization of the replay result at the transition level on the filtered model, represented in Figure 1, obtained from the "Receipt phase" log (only transitions with problems have been reported).

Transition	**# Cases Underfed**	**# Cases Fit**
$T05$	1	1299
$T02$	35	1316
$T06$	521	830

The result of localization on a filtered version of the "Receipt phase of an environmental permit application process" event log, and the model represented in Figure 1, is shown in Table 5 (for places with problems) and Table 6 (for transitions with problems). Moreover, in Figure 2 the fitness information has been projected visually on the elements of the Petri net.

5 Advanced Diagnostics

The localized information is useful to compare, for each problematic entity, the set of cases of the log that are fit according to the given entity and the set of cases of the log that are not fit according to the given entity (called "unfit"). In particular, the following questions can be answered:

1. If a given transition is executed in an unfit way, what is the effect on the throughput time?
2. If a given activity that is not contained in the process model is executed, what is the effect on the throughput time?

These questions can be answered by throughput time analysis. Essentially, an aggregation (for example, the median) of the throughput times of fit and unfit cases is taken into account, and the results compared. Usually, transitions executed in an unfit way are corresponding to higher throughput times.

The comparison between the throughput time in non-fitting cases and fitting cases permits to understand, for each kind of deviation, whether it is important or not important for the throughput time. For evaluating this, the "Receipt phase of an environmental permit application process" log is taken. After some filtering operations, the model represented in Figure 1 is obtained. Several activities that are in the log are missing according to the model, while some transitions have fitness issues. After doing the token-based replay enabling the local information retrieval, and applying the *duration_diagnostics.diagnose_from_trans_fitness* function to the log and the transitions fitness object, it can be seen that transition

T06 Determine necessity of stop advice is executed in an unfit way in 521 cases. For the cases where this transition is enabled according to the model the median throughput time is around 20 minutes, while in the cases where this transition is executed in an unfit way the median throughput time is 1.2 days. So, the throughput time of unfit cases is 146 times higher in median than the throughput time of fit cases. Considering activities of the log that are not in the model, that are likely to make the throughput time of the process higher since they are executed rarely, applying the *duration_diagnostics.diagnose_from_not_existing_activities* method it is possible to retrieve the median execution of cases containing these activities, and compare it with the median execution time of cases that do not contain them (that is 20 minutes). Taking into account activity *T12 Check document X request unlicensed*, it is contained in 44 cases, which median throughput time is 6.9 days (505 times higher than standard).

6 Conclusion

In this paper, an improved token-based replay approach has been proposed and has been implemented in the Python process mining library PM4Py[4]. A set of process discovery, conformance checking and enhancement algorithms are provided in the library. An example script, that loads a log, calculates a model, and does conformance checking, is shown in Figure 3. This illustrates that the conformance checking technique presented in this paper can be combined easily with many other process mining and machine learning approaches.

The approach has shown to be more scalable than existing approaches. Due to a better handling of invisible transitions and improved intermediate storage techniques, the approach outperforms the original token-based approaches, and proves to be faster than alignment-based approaches also for models with invisible transitions.

Next to an increase is speed, the problem of token flooding is addressed by "freezing" superfluous tokens (see Section 3.4). This way replay does not lead to markings with many more tokens than what would be possible according to the model, avoiding the activation of unwanted parts of the process models and leading to lower values of fitness for problematic parts of the model.

Localization of conformance checking using token-based replay can be used to simplify the model prior to replay and help to better diagnose where the deviation happened. Moreover, we showed that we are able to diagnose the effects of deviations on the case throughput time.

The approach has been fully implemented in the PM4Py process mining library. We hope that this will trigger a revival of token-based replay, a technique that seemed abandoned in recent years. Especially when dealing with large logs, complex models, and real-time applications, the flexible tradeoff between quality and speed provided by our implementation is beneficial.

[4] It can be installed in Python $\geq$ 3.6 through the command *pip install pm4py*. See http://pm4py.pads.rwth-aachen.de/installation/ for details.

References

1. van der Aalst, W.: Structural characterizations of sound workflow nets. Computing Science Reports **96**(23), 18–22 (1996)
2. van der Aalst, W.: Decomposing Petri nets for process mining: A generic approach. Distributed and Parallel Databases **31**(4), 471–507 (2013)
3. van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **2**(2), 182–192 (2012)
4. van der Aalst, W., van Hee, K.M., ter Hofstede, A.H., Sidorova, N., Verbeek, H., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects of Computing **23**(3), 333–363 (2011)
5. Adriansyah, A.: Aligning observed and modeled behavior (2014)
6. Adriansyah, A., Sidorova, N., van Dongen, B.: Cost-based fitness in conformance checking. In: Application of Concurrency to System Design (ACSD), 2011 11th International Conference on. pp. 57–66. IEEE (2011)
7. van den Broucke, S.K., Munoz-Gama, J., Carmona, J., Baesens, B., Vanthienen, J.: Event-based real-time decomposed conformance analysis. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 345–363. Springer (2014)
8. Carmona, J., Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer (2018)
9. Kerremans, M.: Gartner Market Guide for Process Mining, Research Note G00353970 (2018), www.gartner.com
10. Lee, W.L.J., Verbeek, H., Munoz-Gama, J., van der Aalst, W., Sepúlveda, M.: Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining. Information Sciences **466**, 55–91 (2018)
11. Martínez, J., Silva, M.: A simple and fast algorithm to obtain all invariants of a generalised Petri net. In: Application and Theory of Petri nets, pp. 301–310. Springer (1982)
12. Miyamoto, T., Kumagai, S.: Calculating place capacity for Petri nets using unfoldings. In: Application of Concurrency to System Design, 1998. Proceedings., 1998 International Conference on. pp. 143–151. IEEE (1998)
13. Munoz-Gama, J., Carmona, J., van der Aalst, W.: Conformance checking in the large: Partitioning and topology. In: Business Process Management, pp. 130–145. Springer (2013)
14. Munoz-Gama, J., Carmona, J., van der Aalst, W.: Single-entry single-exit decomposed conformance checking. Information Systems **46**, 102–122 (2014)
15. Rogge-Solti, A., Senderovich, A., Weidlich, M., Mendling, J., Gal, A.: In log and model we trust? a generalized conformance checking framework. In: International Conference on Business Process Management. pp. 179–196. Springer (2016)
16. Rozinat, A., van der Aalst, W.: Conformance testing: measuring the alignment between event logs and process models. Citeseer (2005)
17. Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1), 64–95 (2008)
18. Taymouri, F., Carmona, J.: A recursive paradigm for aligning observed behavior of large structured process models. In: International Conference on Business Process Management. pp. 197–214. Springer (2016)