Workshop Proceedings

Workshop on

Algorithms & Theories for the Analysis of Event Data (ATAED'2022)

Workshop, June 24, 2022

supported by the IEEE Task Force on Process Mining

Satellite event of the conference

43rd International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2022)

Edited by
Robert Lorenz, Jan Martijn van der Werf, and Sebastiaan J. van Zelst

.

Preface

Ehrenfeucht and Rozenberg defined regions more than 30 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 *region*s. 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), Bratislava (2018), Aachen (2019), and virtually in 2020 (due to the COVID-19 pandemic). 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'2022 workshop took place as a physical workshop on June 21st, 2022 and was a satellite event of the 43rd International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2022), held in Bergen, Norway.

Papers related to process mining, region theory and other synthesis techniques were presented at the ATAED'2022, divided over three content-oriented sessions, i.e., "Stochastics & Statistics", "Region Theory" and "Strategies for Behavioral Analysis". All the techniques presented have in common that "lower-

level" behavioral descriptions (event logs, partial languages, transition systems, 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.tf-pm.org/`).

After a careful reviewing process, six papers (out of a total of ten submissions) were accepted for the workshop. We thank the reviewers for providing the authors with valuable and constructive feedback. We thank the authors and the presenters for their wonderful contributions.

Enjoy reading the proceedings!

Robert Lorenz, Jan Martijn van der Werf, and Sebastiaan J. van Zelst
June 2022

Program committee of ATAED'2022

Abel Armas Cervantes, QUT, Australia
Luca Bernardinello, Universitá degli studi di Milano-Bicocca, Italy
Paolo Ceravolo, University of Milan, Italy
Jochen De Weerdt, KU Leuven, Belgium
Benoît Depaire, Hasselt University, Belgium
Jörg Desel, FernUni Hagen, Germany
Claudio Di Ciccio, Sapienza University of Rome, Italy
Chiara Di Francescomarino, FBK-IRST, Italy
Dirk Fahland, TU Eindhoven, The Netherlands
Stefan Haar, LSV CNRS & ENS de Cachan, France
Anna Kalenkova, University Adelaide, Australia
Jetty Kleijn, Leiden University, The Netherlands
Sander Leemans, RWTH Aachen University, Germany
Robert Lorenz, University of Augsburg, Germany (co-chair)
Lisa Mannel, RWTH Aachen University, Germany
Marta Pietkiewicz-Koutny, Newcastle University, United Kingdom
Andrey Rivkin, Free University of Bozen Bolzano, Italy
Daniel Schuster, Fraunhofer FIT/RWTH Aachen University, Germany
Arik Senderovich, York University, Canada
Ronny Tredup, University of Rostock, Germany
Lijie Wen, Tsinghua University, China
Alex Yakovlev, Newcastle University, United Kingdom
Jan Martijn van der Werf, Utrecht University, The Netherlands (co-chair)
Sebastiaan J. van Zelst, Fraunhofer FIT/RWTH Aachen University, Germany (co-chair)

Leveraging frequencies in event data

a pledge for stochastic process mining

Sander J.J. Leemans[1]

[1]*RWTH, Aachen, Germany*

Abstract

Process mining aims to obtain insights from event logs. In this extended abstract, we will show that it is useful to take the frequency perspective (that is, stochastic behaviour) into account, and will discuss several stochastic process mining techniques.

Keywords

process mining, stochastic process mining, stochastic process discovery, stochastic conformance checking

1. Process mining

Organisations run on processes: processing an order, onboarding a new hire, getting travel approval; many work performed in organisations can be considered as processes. Process mining aims to optimise these processes through event logs: records of executions of processes, typically obtained from information systems that support the processes.

Figure 1 shows an overview the context and common tasks of process mining. A process is running in an organisation, and through information systems an event log is recorded. Using a process discovery technique, a process model can be discovered. Process discovery techniques need to trade-off several potentially competing model quality aspects, such as readability and filtering noise. Ideally, a process model would be compared to the actual process, however as that is assumed to be unknown, this relation can only be theoretically proven under certain assumptions, or estimated. Rather, in practice a model should be compared to (a separate test) event log, for instance on the quality dimensions of simplicity, fitness – the fraction of behaviour in the event log that is in the process model, and precision – the fraction of behaviour of the model that was observed in the event log.

A process model expresses a set of potential traces that the model supports, and it may be difficult to fully interpret insights gained from such a model by itself. Therefore, in process mining projects, the model is typically enhanced with frequency or performance information, after which the project may continue with repeated drill-down filters, hypotheses and verification [1]. In more advanced settings, process models can be

Algorithms and Theories for the Analysis of Event Data (ATAED'22)

EMAIL: s.leemans@bpm.rwth-aachen.de (S. J.J. Leemans)

URL: https://leemans.ch/ (S. J.J. Leemans)

ORCID: 0000-0002-5201-7125 (S. J.J. Leemans)

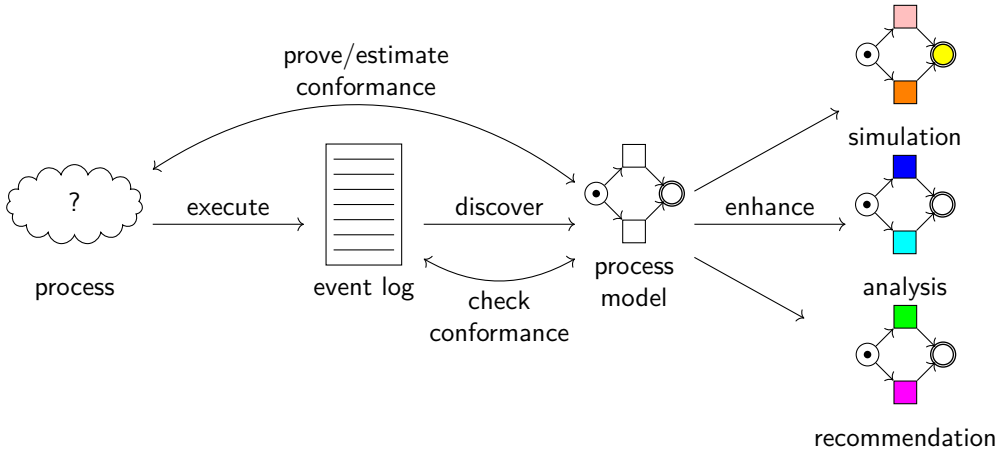

Figure 1: An overview of common tasks of process mining.

simulated. This provides a baseline for, after applying certain changes, comparing process redesigns in a what-if analysis. Finally, if process mining is integrated into daily operations, process models can be used to recommend interventions for traces that are still in the process [2].

2. Frequencies in process mining: the stochastic perspective

Let us consider two event logs:

$$L_1 = [\langle register, check, accept\rangle^{10000}, \langle register, check, reject\rangle^{10000}, \langle register, accept\rangle^{1}]$$

$$L_2 = [\langle register, check, accept\rangle^{9500}, \langle register, check, reject\rangle^{9500} \langle register, accept\rangle^{1001}]$$

These logs have an equivalent control flow: the set of traces in both logs is the same. However, it is obvious that these logs are not from the same process: in L_1, $\langle register, accept\rangle$ occurs once, while in L_2 it occurs more than a thousand times. Any process mining techniques ignoring the stochastic perspective will consider these logs as to come from the Thus, these logs are different mostly because of their frequencies; in process models, we refer to this as the *stochastic perspective*.

The stochastic perspective is obviously present in a process: behaviour has a certain likelihood of appearing. Consequently, an event log derived from a process also has a stochastic perspective: behaviour has a certain likelihood of being recorded in the event log. Thus, the stochastic perspective *is there*.

On th right side of Figure 1, we need an idea of how often behaviour occurs in order to perform analysis: it matters whether behaviour is exceptional or common, and average performance measures are weighted by definition on the multiplicity of behaviour. For simulation, simulation software needs to know how likely each path or decision in the

process is. Similarly, recommendation needs to be aware of how likely behaviour is in order to steer towards more likely favourable outcomes [3]. Thus, the most useful parts of process mining *need* the stochastic perspective. This leaves an obvious gap between the stochastic-having event logs and the stochastic-needing analysis, simulation and recommendation: process models with a stochastic perspective: *stochastic process models*. A stochastic process model not only expresses what behaviour can happen, but also how likely each trace is.

2.1. Analysis, simulation & recommendation

Without existing stochastic process models, existing analysis, simulation and recommendation techniques, which inherently use the stochastic perspective, must obtain this stochastic information in an ad-hoc fashion from the event log [3, 4]. Consequently, such techniques have no idea of the quality of the stochastic perspective they operate on and risk testing on their training logs, which is not good practice. Without explicit stochastic information, one cannot write it down, cannot reason about it, and adjust it in process redesign efforts.

2.2. Precision

Another area where considering stochastic process information is beneficial is in the evaluation of process models: we already discussed fitness, and most fitness measures take the stochastic information into account implicitly: the more likely behaviour in the log, the higher its influence on the fitness measure. Precision measures express the fraction of behaviour of the model that was seen in the event log:

$$\text{precision} = \frac{|\text{model} \cap \text{log}|}{|\text{model}|}$$

An inherent problem with this intuitive informal definition is that one needs a count of behaviour in the model. One cannot simply count traces, as models may express infinitely much behaviour through loops.

We illustrate this for one non-stochastic precision technique [5]. This approach considers the outgoing edges of the state space of a model: they divide the number of edges taken by the total number of edges to arrive at a number. However, these techniques do not consider at all what lies beyond edges that were seen in the model. Thus, unseen behaviour is only counted proportionally to the number of edges that go into that area, irrespective of the "size" of the unseen part [5].

For a stochastic process model, this is not a problem as we have a notion of size: in the state space, it is known exactly how likely each edge is, and that is exactly equal to the size – the probability mass – of the model that lies behind it. Thus, for stochastic process models more intuitive precision measures can be defined.

2.3. Reliability of conclusions

If we consider Figure 1 again, inaccuracies or imprecisions can be introduced at many steps of these common process mining tasks:

- When recording the event log from a process, the quality of the recording may vary, or extraction may be biased;
- When discovering a process model, a process discovery technique may need to make well-known trade-offs between potentially competing quality criteria;
- When estimating the quality of a process model with respect to the process, assumptions and bias may be tested based on a process model [6];
- When comparing a process model with an event log using a conformance checking technique, such a technique will try to squeeze a trace from the log onto the best-fitting path through the model [7]; There might be multiple such best-fitting paths, and there is no guarantee that a best-fitting path is the most likely explanation, yielding ambiguities and potentially inaccuracies;
- When enhancing a process model for analysis, simulation or recommendation, behaviour where log and model do not agree on (non-conforming parts) needs to be handled [4].

All of these steps may be sources of inaccuracies and imprecisions, which may propagate and aggregate over a process mining project. We conjecture that the use of stochastic process models makes it easier to quantify and study these inaccuracies and imprecisions, such that the reliability of conclusions can be quantified [8], and improved.

3. Existing stochastic process mining techniques

Next, we discuss some stochastic process mining techniques that mimic standard non-stochastic techniques: stochastic process discovery and stochastic conformance checking. Furthermore, we discuss completely new types of techniques that require considering stochastic process behaviour.

3.1. Stochastic process discovery

In stochastic process discovery, the aim is to automatically discover a stochastic process model – such as a stochastic labelled Petri net [14] – from an event log. Most stochastic process mining techniques take an existing non-stochastic process model and construct a stochastic process perspective on top if it. For instance, [9] constructs a stochastic perspective through time: it estimates the delay distribution of process steps, which in turn determines their likelihood. Another approach constructs a stochastic perspective for a process model using several estimators, ranging from simple counting to alignments [10].

A technique that does not start from an existing is [11], which starts from the behaviour in the event log and using reduction rules compacts, summarises and abstracts the stochastic behaviour until a suitable model remains.

A completely different approach is taken by [12], which constructs declarative constraints on the log, such that these constraints hold with a certain likelihood. As such, these models describe multiple options for stochastic behaviour, rather than a single stochastic language.

3.2. Stochastic conformance checking

A stochastic conformance checking technique compares with one another an event log and a stochastic process model, or two stochastic process models, or two event logs. Stochastic entropy [13] consists of two measures: recall is the entropy of the common behaviour of log and model – minimal number of bits required to describe the behaviour – divided by the entropy of the log. Precision is then the entropy of the common behaviour of the log and model divided by the entropy of the model.

Another stochastic conformance checking technique is the Earth Movers' Distance [14], which considers both log and model (or any other combination; loops need to be unfolded) as distributions over traces, and then applies the Wasserstein distance principle, which finds the least-cost way to transform one dstribution into the other. That is, both distributions are piles of earth, and the distance says how much earth – trace probability mass – need to be transported over what distance – trace difference – in order to transform one pile into the other. Besides a single conformance number, this measure can also provide detailed insights when projected on a model.

3.3. Goodies

Next to stochastic extensions of techniques, the concept of stochastic process behaviour has enabled some new types of analyses and techniques.

Some event logs have hundreds of activities, which make them challenging to analyse. A way to simplify models is to apply a trace-based filter, thereby focusing the analysis on, for instance, platinum customers, or orders with a value over a certain amount. Cohort analysis recommends filters based on the difference between traces that pass the filter and all the other traces: the filter that is associated with the largest difference in stochastic behaviour would simplify the model the most [15].

Most process mining insights are associational. However, as in associational insights there is no difference between cause and effect, for redesign or what-if analyses it is beneficial to perform causal reasoning. Recent studies have introduced causal reasoning: to discover causal rules from event logs [16], to discover causal relations between decisions in a process model [17], and to perform root-cause analysis [18]. All of these techniques are inherently enabled by the concept of stochastic process behaviour.

Further techniques targeting stochastic behaviour are the detection of differences in a stochastic process over time (concept drift) [19]; to discover anomalies in event logs without the use of process models [20]; and the combination of data-aware and stochastic process models [21].

4. Conclusion

Process mining is an exciting field of research. In this pledge for consideration of the stochastic perspective of process behaviour, we have shown several challenges of process mining concepts and techniques that may benefit from having a stochastic perspective. Several recent stochastic process mining techniques were discussed, both drop-in replacements for well-known process discovery and conformance checking techniques, as well as new techniques that leverage the stochastic perspective of behaviour of event logs to enable new types of analysis.

References

[1] M. L. van Eck, X. Lu, S. J. J. Leemans, W. M. P. van der Aalst, PM ^2 : A process mining project methodology, in: J. Zdravkovic, M. Kirikova, P. Johannesson (Eds.), Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings, volume 9097 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 297–313. URL: https://doi.org/10.1007/978-3-319-19069-3_19. doi:10.1007/978-3-319-19069-3_19.

[2] M. Shoush, M. Dumas, When to intervene? prescriptive process monitoring under uncertainty and resource constraints, CoRR abs/2206.07745 (2022). URL: https://doi.org/10.48550/arXiv.2206.07745. doi:10.48550/arXiv.2206.07745. arXiv:2206.07745.

[3] I. Verenich, M. Dumas, M. L. Rosa, H. Nguyen, Predicting process performance: A white-box approach based on process models, J. Softw. Evol. Process. 31 (2019). URL: https://doi.org/10.1002/smr.2170. doi:10.1002/smr.2170.

[4] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Exploring processes and deviations, in: BPM Workshops, volume 202 of *LNBIP*, 2014, pp. 304–316.

[5] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, W. M. P. van der Aalst, Measuring precision of modeled behavior, Inf. Syst. E Bus. Manag. 13 (2015) 37–67. URL: https://doi.org/10.1007/s10257-014-0234-7. doi:10.1007/s10257-014-0234-7.

[6] G. Janssenswillen, B. Depaire, Towards confirmatory process discovery: Making assertions about the underlying system, Bus. Inf. Syst. Eng. 61 (2019) 713–728. URL: https://doi.org/10.1007/s12599-018-0567-8. doi:10.1007/s12599-018-0567-8.

[7] W. M. P. van der Aalst, A. Adriansyah, B. F. van Dongen, Replaying history on process models for conformance checking and performance analysis, WIREs Data Mining Knowl. Discov. 2 (2012) 182–192. URL: https://doi.org/10.1002/widm.1045. doi:10.1002/widm.1045.

[8] K. Goel, S. J. J. Leemans, N. Martin, M. T. Wynn, Quality-informed process mining: A case for standardised data quality annotations, ACM Trans. Knowl. Discov. Data 16 (2022) 97:1–97:47. URL: https://doi.org/10.1145/3511707. doi:10.1145/3511707.

[9] A. Rogge-Solti, W. M. P. van der Aalst, M. Weske, Discovering stochastic petri nets with arbitrary delay distributions from event logs, in: N. Lohmann, M. Song,

P. Wohed (Eds.), Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers, volume 171 of *Lecture Notes in Business Information Processing*, Springer, 2013, pp. 15–27. URL: https://doi.org/10.1007/978-3-319-06257-0_2. doi:10.1007/978-3-319-06257-0_2.

[10] A. Burke, S. J. J. Leemans, M. T. Wynn, Stochastic process discovery by weight estimation, in: S. J. J. Leemans, H. Leopold (Eds.), Process Mining Workshops - ICPM 2020 International Workshops, Padua, Italy, October 5-8, 2020, Revised Selected Papers, volume 406 of *Lecture Notes in Business Information Processing*, Springer, 2020, pp. 260–272. URL: https://doi.org/10.1007/978-3-030-72693-5_20. doi:10.1007/978-3-030-72693-5_20.

[11] A. Burke, S. J. J. Leemans, M. T. Wynn, Discovering stochastic process models by reduction and abstraction, in: D. Buchs, J. Carmona (Eds.), Application and Theory of Petri Nets and Concurrency - 42nd International Conference, PETRI NETS 2021, Virtual Event, June 23-25, 2021, Proceedings, volume 12734 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 312–336. URL: https://doi.org/10.1007/978-3-030-76983-3_16. doi:10.1007/978-3-030-76983-3_16.

[12] F. M. Maggi, M. Montali, R. Peñaloza, A. Alman, Extending temporal business constraints with uncertainty, in: D. Fahland, C. Ghidini, J. Becker, M. Dumas (Eds.), Business Process Management - 18th International Conference, BPM 2020, Seville, Spain, September 13-18, 2020, Proceedings, volume 12168 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 35–54. URL: https://doi.org/10.1007/978-3-030-58666-9_3. doi:10.1007/978-3-030-58666-9_3.

[13] S. J. J. Leemans, A. Polyvyanyy, Stochastic-aware conformance checking: An entropy-based approach, in: S. Dustdar, E. Yu, C. Salinesi, D. Rieu, V. Pant (Eds.), Advanced Information Systems Engineering - 32nd International Conference, CAiSE 2020, Grenoble, France, June 8-12, 2020, Proceedings, volume 12127 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 217–233. URL: https://doi.org/10.1007/978-3-030-49435-3_14. doi:10.1007/978-3-030-49435-3_14.

[14] S. J. J. Leemans, W. M. P. van der Aalst, T. Brockhoff, A. Polyvyanyy, Stochastic process mining: Earth movers' stochastic conformance, Inf. Syst. 102 (2021) 101724. URL: https://doi.org/10.1016/j.is.2021.101724. doi:10.1016/j.is.2021.101724.

[15] S. J. J. Leemans, S. Shabaninejad, K. Goel, H. Khosravi, S. W. Sadiq, M. T. Wynn, Identifying cohorts: Recommending drill-downs based on differences in behaviour for process mining, in: G. Dobbie, U. Frank, G. Kappel, S. W. Liddle, H. C. Mayr (Eds.), Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings, volume 12400 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 92–102. URL: https://doi.org/10.1007/978-3-030-62522-1_7. doi:10.1007/978-3-030-62522-1_7.

[16] Z. D. Bozorgi, I. Teinemaa, M. Dumas, M. L. Rosa, A. Polyvyanyy, Process mining meets causal machine learning: Discovering causal rules from event logs, in: ICPM, IEEE, 2020, pp. 129–136.

[17] S. J. J. Leemans, N. Tax, Causal reasoning over control-flow decisions in process models, in: X. Franch, G. Poels, F. Gailly, M. Snoeck (Eds.), Advanced Information Systems Engineering - 34th International Conference, CAiSE 2022, Leuven, Belgium,

June 6-10, 2022, Proceedings, volume 13295 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 183–200. URL: https://doi.org/10.1007/978-3-031-07472-1_11. doi:10.1007/978-3-031-07472-1_11.

[18] M. S. Qafari, W. M. P. van der Aalst, Root cause analysis in process mining using structural equation models, in: BPM Workshops, volume 397 of *LNBIP*, 2020.

[19] T. Brockhoff, M. S. Uysal, W. M. P. van der Aalst, Time-aware concept drift detection using the earth mover's distance, in: B. F. van Dongen, M. Montali, M. T. Wynn (Eds.), 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, October 4-9, 2020, IEEE, 2020, pp. 33–40. URL: https://doi.org/10.1109/ICPM49681.2020.00016. doi:10.1109/ICPM49681.2020.00016.

[20] T. Nolle, Process learning for process autonomous anomaly correction (extended abstract), in: W. M. P. van der Aalst, R. M. Dijkman, A. Kumar, F. Leotta, F. M. Maggi, J. Mendling, B. T. Pentland, A. Senderovich, M. Sepúlveda, E. S. Asensio, M. Weske (Eds.), Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track at BPM 2021 co-located with 19th International Conference on Business Process Management (BPM 2021), Rome, Italy, September 6th - to - 10th, 2021, volume 2973 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 6–10. URL: http://ceur-ws.org/Vol-2973/paper_137.pdf.

[21] F. Stertz, J. Mangler, S. Rinderle-Ma, Data-driven improvement of online conformance checking, in: 24th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2020, Eindhoven, The Netherlands, October 5-8, 2020, IEEE, 2020, pp. 187–196. URL: https://doi.org/10.1109/EDOC49727.2020.00031. doi:10.1109/EDOC49727.2020.00031.

Enhancing Stochastic Petri Net-based Remaining Time Prediction using k-Nearest Neighbors

Jarne Vandenabeele[1], Gilles Vermaut[1], Jari Peeperkorn and Jochen De Weerdt

[1]*Co-first authors*

Research Center for Information Systems Engineering (LIRIS), KU Leuven, Leuven, Belgium

Abstract

Reliable remaining time prediction of ongoing business processes is a highly relevant topic. One example is order delivery, a key competitive factor in e.g. retailing as it is a main driver of customer satisfaction. For realising timely delivery, an accurate prediction of the remaining time of the delivery process is crucial. Within the field of process mining, a wide variety of remaining time prediction techniques have already been proposed. In this work, we extend remaining time prediction based on stochastic Petri nets with generally distributed transitions with k-nearest neighbors. The k-nearest neighbors algorithm is performed on simple vectors storing the time passed to complete previous activities. By only taking a subset of instances, a more representative and stable stochastic Petri Net is obtained, leading to more accurate time predictions. We discuss the technique and its basic implementation in Python and use different real world data sets to evaluate the predictive power of our extension. These experiments show clear advantages in combining both techniques with regard to predictive power.

Keywords

business processes, stochastic Petri nets, process mining, predictive process monitoring

1. Introduction

The application of Process-Aware Information Systems (PAIS)s, such as ERP, BPMS and CRM systems to support business processes is increasing [1]. These systems record information of the process execution and possibly the individual events of that process. Drawing insights and conclusions from these data is already possible using various process mining techniques categorized into process discovery, conformance checking, and extension [2]. Within the field of process mining, predictive process monitoring concerns itself with predictive techniques applied to process data. Predominantly, three types of predictions are considered as useful: next activity, outcome, and remaining time [3]. In this work we present a novel technique that combines a data-driven selection of candidate traces with building and simulating stochastic Petri nets, to predict the remaining time of running process instances. It builds upon the technique described by Rogge-Solti and Weske in [4, 5]. This technique, referred to as *generally distributed transition stochastic Petri net (GDT_SPN)*, tailors stochastic Petri nets to incorporate the time passed since the previous completed event. We show that by combining the flexibility of GDT_SPNs with a simple, yet effective, candidate selection approach, we can improve upon the accuracy of the

ALGORITHMS & THEORIES FOR THE ANALYSIS OF EVENT DATA 2022

jari.peeperkorn@kuleuven.be (J. Peeperkorn); jochen.deweerdt@kuleuven.be (J. De Weerdt)

predictions, as demonstrated experimentally on four real-life datasets. Our approach can be summarized as follows:

- Use the K-nearest-neighbor algorithm to identify the k instances most similar to the current prefix of this instance. The algorithm uses vectors based on the timestamps of previous events, while also taking into account the activity types.
- We use these k traces to discover a (new) Petri net using the Inductive Miner [6] and by performing a simulation a stochastic map is obtained which complements the Petri net, to eventually obtain a GDT_SPN.
- A simulation of this GDT_SPN is further used to estimate the remaining time. This is done n times and the actual prediction is taken to be the average of the n simulations.

The remainder of this paper is organised as follows. In Section 2, the most relevant related work is discussed. Section 3 defines some preliminaries, before Section 4 presents the core of our technique, i.e., how we predict the end time, and the basic implementation of it in Python. Section 5 evaluates our technique by comparing the results of different experiments with a number of benchmarks. We end this paper with Section 6, which summarises the paper, touches upon the limitations of our technique and mentions possible further enhancements.

2. Related Work

There already exist multiple techniques for predicting the remaining time of an ongoing process instance. For a comprehensive overview of the most relevant methods, interested readers are referred to Verenich et al. [7]. For the sake of conciseness this section is limited to an overview of those techniques that are either relevant for the vast majority of the methods discussed in [7], or those that are highly related to the framework discussed in this paper. Earlier work concerning remaining time prediction was proposed by van der Aalst et al. [8]. By applying finite state machine techniques on event logs, they learn an annotated transition system. Such a system extends a traditional transition system with predictive information by annotating measurements of time instances at each state. Two follow-up techniques are presented in [9, 10]. They enhance the technique of [8] by clustering the traces of the log in advance and creating an annotated transition system for each of these clusters afterwards. At runtime, a new trace will be assigned to a cluster and the annotated transition system for that cluster will then be used to predict the remaining time of that trace. A similar multi-stage approach is presented in this paper. The application of clustering to predictive business process monitoring has also been studied in other papers, e.g., [11].

Polato et al. [12, 1] present three approaches, all of them using Support Vector Regressors (SVR), to predict the remaining time starting from a certain state. The first two are based on regression techniques with or without control-flow information, where the event log that serves as input is initially transformed in order to be suitable for the ε-SVR algorithm. The last approach is again mainly based on the idea of annotated transition systems as mentioned above [8]. It enriches each state with a Naive Bayes classifier and each transition with a Support Vector Regressor, yielding a Data-Aware Transition System (DATS).

The above mentioned techniques are often based on support vector machines and/or regression. These are, however, not the only machine learning techniques used to create predictive

models. For instance, regression trees [13], XGBoost [14], or random forests [15] have already been applied for remaining time prediction. The increasing interest and latest achievements in deep learning techniques have led to a rise in predictive models using recurrent neural networks, convolutional neural networks, generative adversarial nets and, more recently, transformer nets, together with multistage approaches [16, 17, 18, 19, 20, 21, 22, 23, 24]. While these works show promising results, the presented models are so-called *black box*, i.e. their results are hard or even impossible to interpret. This limits their use in applications where the explainable aspect is key, e.g. root-cause analysis.

We particularly highlight the work of Rogge-Solti and Weske [4, 5], as their approach has been the main inspiration for the technique presented in this paper. Their GDT_SPN-formalism based technique differs from other approaches as it takes into account the time passed since the last observed event, while the other approaches only update predictions upon arrival of finished events. To achieve this, stochastic Petri nets are equipped with generally distributed transitions, in which distributions reflect the duration of the corresponding activities in the real world and can be different from the exponential distribution, i.e., the Markovian property is not enforced. The latter distributions may then be updated, based on the time passed since the last observed event, to achieve more accurate predictions. These models are not *black box*, and can thus be used to explain multiple aspects influencing the execution time of a certain instance.

3. Preliminaries

In this chapter, we describe the preliminary concepts on which our novel prediction technique is based. As mentioned above, the goal of this technique is to predict the remaining time of ongoing cases. This entails that the predictive model can only use information of the current case up until this point in time, supplemented with information from past cases. The assumption is made that traces contain timestamps for each occurring event. Those timestamps indicate at least the end, and in some cases the start, of the activities represented in the trace. Each event in the event log should contain a *trace ID*, indicating to which process execution it belongs, together with an activity type. Our technique is an extension of the work of Rogge-Solti and Weske, who introduce the use of stochastic Petri nets with generally distributed transitions, so-called GDT_SPN models, to make predictions [4, 5]. In this technique, a Petri net is enriched with statistical timing data for each event, which allows end-users to make time predictions. The definition of both a Petri net and a GDT_SPN as presented by Rogge-Solti and Weske is given below [5]:

Definition 1 (Petri Net) A Petri net is a tuple PN = (P, Tr, F, M_0) where:

- P is the set of places.
- Tr is the set of transitions.
- $F \subseteq (P \times Tr) \cup (Tr \times P)$ is the flow relation.
- $M_0 \in P \rightarrow \mathbb{N}_0$ is the initial marking.

The Petri net models used by [5], and hence as well in our technique, are restricted to sound workflow nets.

Definition 2 (Generally Distributed Transition Stochastic Petri Net) A GDT_SPN is a seven-tuple: GDT_SPN = (P, Tr, $\mathscr{P}$, $\mathscr{W}$, F, M_0, $\mathscr{D}$), where (P, Tr, F, M_0) is the basic underlying Petri net as specified above. Additionally:

- The set of transitions Tr is split into immediate transitions Tr_i and timed transitions Tr_t.
- $\mathscr{P}$: $Tr \rightarrow \mathbb{N}_0$ is the assignment of priorities to the different transitions, where $\forall \tau \in Tr_i$: $\mathscr{P}(\tau) \geq 1$ and $\forall \tau \in Tr_t$: $\mathscr{P}(\tau) = 0$.
- $\mathscr{W}$: $Tr_i \rightarrow \mathbb{R}^+$ assigns probabilistic weights to the immediate transitions Tr_i.
- $\mathscr{D}$: $Tr_t \rightarrow D$ is the assignment of probability distribution functions D to timed transitions Tr_t, reflecting the durations of the corresponding activities.

These probability distribution functions D do not need to be exponentially distributed, but can also be normal distributions, uniform distributions, etc. and hence do not necessarily have the Markovian property, i.e., memorylessness. This is an important factor as the absence of this property is used to update the distribution functions based on the passed time. This is the key difference with the more known generalised stochastic Petri nets (GSPN), as presented by Marsan et al. [16]

Rogge-Solti and Weske [4, 5] exploit the idea of memorylessness to update the density function of the original distribution towards a new density function of a *truncated* distribution. The intuition behind this is that when some time has passed, the probability increases that the activity will be completed in the nearer future as some work might already have been done. This is in contrast with the above mentioned Markovian property. This is done by using the elapsed time since the last event. Let $F_\delta(t)$ be the duration distribution function of that activity. By differentiating $F_\delta(t)$ you can obtain a density function $f_\delta(t)$. We then use t_0, the current time since enabling the transition, to truncate the distribution as follows [5]:

$$f_\delta(t|t \geq t_0) = \begin{cases} 0 & t < t_0, F_\delta(t_0) < 1 \\ \frac{f_\delta(t)}{1-F_\delta(t_0)} & t \geq t_0, F_\delta(t_0) < 1 \\ f_{\delta_{Dirac}}(t - t_0) & F_\delta(t_0) = 1 \end{cases}$$

The part of the density function above (or in this case more correctly *after*) t_0 is rescaled in such a way that it integrates to 1. For the case where $F_\delta(t_0) = 1$, i.e. when the current time has progressed further than the density functions supports, we use the Dirac delta function $f_{\delta_{Dirac}}$ (a function whose value is 0 everywhere except at one peak, and whose full integral is equal to 1), with a peak at t_0. This happens when the current event is taking longer to complete than the events in the training log corresponding to the same activity type. The basic idea is that the predictions are only based on those cases for whom the corresponding activity would not have been already completed at the current time t_0. For a more extensive explanation, together with the effects of this truncation of different types of distributions, interested readers are referred to [5]. In addition, n is the number of simulations performed by the GDT_SPN, for a given sample case. The eventual prediction is equal to the mean duration. As will be explained in Section 4, our proposed prediction technique combines the algorithm as described by [4, 5] with the basic idea of the kNN algorithm. We have adopted the notion of finding the k most representative training instances for the to-be-predicted instance. These k representative cases found during kNN are used to build a predictive GDT_SPN model that can then be used to make a prediction.

4. Remaining Time Prediction using GDT_SPN_*k*NN

In this Section, we introduce generally distributed transition stochastic Petri net with *k*NN-based candidate selection, referred to as GDT_SPN_*k*NN. The algorithm depends on the number of neighbors taken into account, hyperparameter k. In order to make predictions, the algorithm further requires a log file containing complete traces of the business process (the *training log*), the time passed since the start of the case t_0, and a partial trace T, containing events with timestamps until t_0. The key extension of GDT_SPN_*k*NN with respect to the original GDT_SPN algorithm, resides in the fact that we avoid using the whole training log as an input to construct the GDT_SPN model. In contrast, in GDT_SPN_*k*NN, only the k complete traces that are most similar to the to-be-predicted partial trace T are taken into account. The main motivation for selecting this subset is that, under the condition that the right features are selected, only looking at the k most similar traces will yield a predictive model that embodies less variation in the generalised density functions, likely leading to a final prediction closer to the real value. If tuned well, selecting only the k nearest instances thus omits irrelevant cases and outliers and yields a more representative and stable GDT_SPN model for the given partial trace.

The choice of the hyperparameter k to decide the number of nearest neighbors is not obvious. When k is taken too small, the event distributions built by the k neighbors may be inaccurate or have a high variance, which leads to volatile and most of the time worse results. When k is taken too large, the effect of looking at only the most similar traces may be minimal or even non-existent, given that the larger k, the more the outcomes will look like those of the original technique. By taking k too large, the variance may increase as well, as multiple trace variants are taken into account for building the model, which leads to a more complex Petri net. These different trace variants may also have a different underlying distribution to estimate the duration of the activities. Another drawback of setting k too large is that the computation time increases, as more traces need to be replayed to construct the Petri net and distributions. Nonetheless, despite the fact that hyperparameter k is key, it is computationally demanding to tune it. Based on initial explorations, we found that a value of 100 provided to be a good trade-off value. While in a practical application, it would be certainly beneficial to tune k carefully, in this paper, mainly due to a lack of time, we work with this fixed value of 100. In further research, we would like to investigate dedicated strategies to tune k, beyond an exhaustive search.

For applying k nearest neighbors, a proper featurization should be applied. While a conventional feature engineering would typically consider trace and event attributes, we propose a method that only relies on time information. More specifically, on a per event basis, we take into account the total time from the start of each trace t_0 until the occurrence of that individual event. This total duration between the start of the *trace* and the occurrence of that *event* will be referred to as the *time-to-occurrence* of an event in the remainder of this paper. In case of the presence of loops, the *time-to-occurrence* is determined based on the last repeated activity observed in a trace. The rationale behind using *time-to-occurrence* is that there is a likely correlation between traces that have a similar time-to-occurrence for events corresponding to the same activity type and hence, it might thus be valuable to sample them in order to make better remaining time predictions. This feature engineering is also more generalisable compared to matching neighbors based on attributes, as not all log files contain event and/or trace attributes. However, for certain processes, it might well be that taking into account other trace or event features is

beneficial. Our algorithm easily allows to incorporate this, when e.g. expert knowledge suggests correlations between these features and the remaining time. When we eventually build the GDT_SPN using the kNN, we still do n different simulations for which we take the average values as our eventual prediction. In this work, the value of $n = 500$ is chosen, adopted from [5], as taking this high enough will make the predictions more robust and less volatile.

Algorithm 1: Feature Construction of the times_to_occurrence vector for a trace

Result: Times_to_occurrence vector ψ^* for a trace ψ
ψ = Trace ;
Voc = [all activity types in training data];
Function *Trace_times_to_occurrence(ψ, Voc)*
  ψ_{start} = start_time(ψ);
  $\psi^* = [\psi_1^*, \psi_2^*, \dots, \psi_D^*]$ with $D = |\text{Voc}|$;
  for $i \in \{1, \dots, D\}$ **do**
    if $Voc_i \in activities(\psi)$ **then**
      event = event corresponding to the last occurring event of activity type Voc_i;
      endtime = timestamp(event);
      ψ_i^* = endtime - ψ_{start};
    else
      $\psi_i^* = -1$;
    end
  end
  return ψ^*
end
Function *activities(ψ)*
  Intitialize: $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_L]$ with $L = |\psi|$;
  for $i \in \{1, \dots, |\psi|\}$ **do**
    α_i = Activity type of event ψ_i
  end
  return α
end

The feature construction needed to predict the remaining time of a (partial) test trace T can be seen in Algorithm 1. We define an activity vocabulary *Voc* containing all possible activity types present in the process. Each trace ψ of the training log is formatted as follows. For each activity present in the activity vocabulary of the event log, the algorithm checks whether an event occurs in ψ corresponding to that activity type. If it does, then the difference between the time of this event in ψ and the beginning of this trace is calculated and stored in a formatted vector ψ^* corresponding to this trace ψ. If it does not, a negative value is assigned. We call this value the *time-to-occurrence* of that activity type (ψ_i^* in the algorithm above). For the to-be-predicted trace T a similar formatted vector is constructed. If a certain activity type is present in the trace multiple times, as is the case for e.g. loops, we only consider the last occurring event. We choose to take the last occurrence, because if a certain activity has to be performed multiple times (in one execution), we assume most often the final completion time of that activity (i.e. the timestamp of the last occurrence) provides the most useful information. However, this choice can be easily altered if needed.

The result of this procedure is that for each trace, we obtain a formatted counterpart in the form of a vector with a fixed length. The length corresponds to the number of activities in the

business process, i.e., all distinct activities observed in the training data. Each entry in this vector corresponds to a fixed activity. The entry is positive if an event corresponding with the completion of the activity appears in the trace, and is negative otherwise. For the remainder of this explanation, we call the formatted version of the to-be-predicted trace T^*. Only the distance between relevant events is measured. These are the events that appear in the partial test trace T and consequently have a positive value in T^*. We want traces from the training log that follow the same path, i.e. having the same executed activities, as the partial trace to have a higher impact, since these traces are more likely to be similar to the partial trace in question. Accordingly, a penalty, in the form of a maximal distance vector, is induced on those that do not follow the same route up to the point of prediction as the partial test trace. This means that the assigned formatted vector for such a trace will contain a large constant for each event, i.e., the maximum time-to-occurrence seen in the training set, making this trace very far off when selecting the nearest neighbors. A min-max normalisation is applied on the formatted versions of the training traces and on T^*, which gives all events equal influence in calculating the distance. This makes them suitable for the kNN algorithm. If k is higher than the number of traces present in the training set that follow the same route, the algorithm randomly selects the other traces up to k. In a future extension, this might be replaced by taking prefixes closest to the control-flow of prefix T, by some metric. We use the Euclidean distance between the formatted vectors as a distance function in the kNN algorithm. In summary the executed procedure has the following characteristics:

- Only the distance between relevant events is measured. These are the events that appear in the partial test trace T and consequently have a positive value in T^*. The formatted versions of the training traces can have positive values for other events as well, but these are not taken into consideration when selecting the nearest neighbors, as only the events present in T are used to calculate the distances.
- We want traces from the training log that share all relevant events with the partial trace to have a higher impact, since these traces are more likely to be similar to the partial trace in question. That is why a penalty is induced on those that do not follow the same route up to the point of prediction as the partial test trace. It should be noted that the rare situation may occur where the number of kNN is higher than the number of traces present in the training set that follow the same route. If this situation happens, the algorithm will randomly pick the other traces up to k. This is another motivation for proper parameter tuning and to keep the parameter k small enough.
- Because of the normalisation of the vectors, one should remark that all completed events have an equal influence while calculating the distance. In the case an outlier is present, the distance between normal traces will be rather small.
- An important consequence of taking the nearest neighbors to build our model is that we allow the process to be dynamic. The occurrence of a new trace variant in the business process will be fully taken into account in the model as soon as k training examples of this trace variant are recorded.

When the nearest neighbors are found, the full training traces corresponding to these neighbors are used to create a Petri net using Inductive Miner [6], which guarantees to produce sound

and fitting models and hence alleviating some of the shortcomings of the α-miner [25]. As mentioned in Section 3, this soundness is a necessity for the Rogge-Solti and Weske algorithm [4, 5]. Additionally, fitting models make it possible to replay the partial trace T, which is necessary for the algorithm to make a prediction [26]. Given this Petri net and k training traces, simulation can be performed to obtain a stochastic map to complement the Petri net. A stochastic map projects every activity on a probability distribution function. In this way, we obtain a GDT_SPN that can be used for prediction using the original simulation of Rogge-Solti and Weske. Here we have to set another hyperparameter n, which is equal to th number of simulations we perform for the partial trace. The actual prediction is then taken as the average of all these simulations. For the rest of this work we use n equal to 500, which should be high enough to average out outliers in the simulation.

We have translated the implementation of Rogge-Solti and Weske [4, 5], that was available in the open-source program ProM, to a standalone implementation in Python compatible with the *pm4py* framework [27]. It does not yet fully cover all features that are available in ProM, but it covers the core algorithm and achieves similar results. Our standalone implementation, including the addition of the kNN algorithm, can be found on Github[1]. For the kNN algorithm we use the implementation in Scikit-Learn [28].

Our approach is agnostic to the specific candidate selection-technique, given that kNN could be replaced by another technique. For instance, this might be an eager clustering technique where for each cluster a GDT_SPN model can be constructed during the training phase. Similar approaches have been explored in [9, 10, 11]. One advantage of taking such an approach is that, as opposed to the lazy learner kNN, this would yield a better performance in the deployment phase. However, there are also reasons why using kNN could result in more accurate predictions. Eager clustering techniques often yield large sized clusters in addition to some smaller clusters, which takes away the power of combining smart candidate selection with the approach of [4, 5]. Additionally, using kNN allows the process to be dynamic, as changes can be picked up quickly in the resulting GDT_SPN model, leading to more accurate predictions. Depending on the importance of stressing either performance or accuracy and flexibility, one can choose an appropriate kind of learner when adapting our approach.

5. Experimental Evaluation

5.1. Experimental setup

In this section, we will discuss the results of our approach on four real-life datasets. A schematic overview of the setup is given in Figure 1.

Each real-life dataset is split into a training and a test log. In order to compromise between a large enough test set and reasonable computing time, all experiments use a test log with approximately 500 traces, independent of the amount of traces in the training log. The size of the training log is always significantly larger than the size of the test log, ranging from 2500 to 8000 traces, depending on the size of the original dataset from which we took a subset. The train and test log split was done out-of-time, in order to avoid possible data leakage. The training

[1] https://github.com/JarneVDB/BP-Time-Prediction-using-KNN

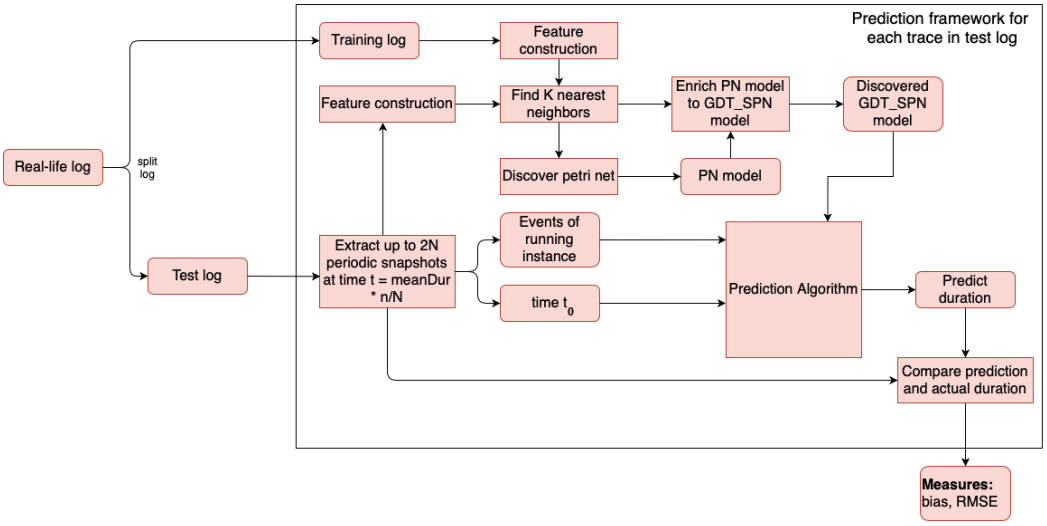

Figure 1: Evaluation setup

log used to obtain the k neighbors only contains finished traces upon to that point in time. All following steps are repeated x times, with x corresponding to the number of traces in the test log. For each trace in the test log, at most $2N$ periodic prediction iterations will be performed, where N is a hyperparameter that influences the number of prediction evaluation iterations. Corresponding to the methodology of [5], the Nth prediction will be performed *meanDur* after the start of the case, where *meanDur* corresponds to the average case duration of the training log. For all experiments, we set N equal to 20, leading to 40 prediction iterations. Each iteration thus refers to a point in time after the start of the trace, where we predict the remaining item of this execution. 2N can thus be regarded as the number of different points in time where we decide to predict the remaining lead time. Each time we do perform a prediction we provide the algorithm with the current traces up to t_0 (corresponding to this point in time). We call the specific point in time we are calculating the remaining times of all (still) ongoing cases, the prediction iteration. Hence, at most 40 moments of prediction are simulated for each trace, each corresponding to different stages in the execution of the process instance. When the execution has finished, at some iteration, no more predictions are done for this particular trace, thus most of the cases in the test log will only influence part of the prediction iterations (as they will be finished at some point). It should be noted that when the process has a low variance, the effective number of prediction iterations for each test trace will be close to N. When increasing the iteration, less and less traces are used to evaluate the time prediction, making the the results more volatile and statistically less significant.

For both the partial test trace T and the entire training log, feature construction is executed as described in Section 4, yielding the corresponding vector representations with the times-to-occurrence of all known activity types. In the transformed version of the training log, the 100 nearest neighbors of the formatted test trace T^* are found. The full traces corresponding to these neighbors are used to discover a Petri net, which is enriched to a GDT_SPN model

by means of stochastic information resulting from a simulation. Although the algorithm is flexible in the choice of distribution types, we decided to force a normal distribution for all non-immediate events. The choice of the most proper distribution is most liekly event log (process) dependent, but the normal distribution was chosen for every process in this paper due to time constraints. A further investigation on this topic, might clarify certain (possible) issues.

The results of the predictions are better when forcing normal distributions, given that the effect of the model vanishes when using memoryless exponential distributions and alternative distributions such as the uniform distribution are sensitive to outliers. Moreover, we assume that the normal distribution is the distribution type that is often a good estimation for the real underlying distribution function. This GDT_SPN model, together with the events of the running instance T and time t_0, serve as input for the prediction algorithm as described in [4, 5]. The number of different simulations performed by each GDT_SPN n, of which the purpose is explained above, is set to 500 for each experiment as this averages out most of the influence of outliers. The reported evaluation metrics will be the average error and the root mean square error (RMSE). These two metrics will allow us to respectively measure the bias and accuracy of our prediction method. Four different event logs are used in the experimentation. The first one, as a simple proof-of-concept, uses the BPI Challenge 2019 Event Log. However, instead of using the full event log, one single control-flow-variant is selected, which can be seen in Figure 2. The experiments on the other selected event logs, do use multiple different control-flow variants. These event logs are the *Hospital log*, depicting the billing process in a hospital, and the BPI Challenge 2020 event logs, depicting the travel expense declaration process of a university for domestic (*BPI2020d*) and international travel (*BPI2020i*). An overview of summary statistics regarding the different data sets can be found in Table 1.

Datasets	Cases	Events	Event classes	Max case length	Avg. case length	Max case time	Avg. case time
BPI2019[2] (1 variant)	7,460	44,760	6	6	6	356.21	94.54
Hospital[3]	7,847	33,450	7	6	4.26	867.54	156.95
BPI2020d[4]	7,820	40,281	7	6	5.15	290.89	10.52
BPI2020i [5]	2,361	23,726	14	12	10.05	463.04	80.17

Table 1
Descriptive statistics of event logs used for the training of the model. Time-related characteristics are reported in days.

[2]https://doi.org/10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1
[3]https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741
[4]https://doi.org/10.4121/uuid:3f422315-ed9d-4882-891f-e180b5b4feb5
[5]https://doi.org/10.4121/uuid:2bbf8f6a-fc50-48eb-aa9e-c4ea5ef7e8c5

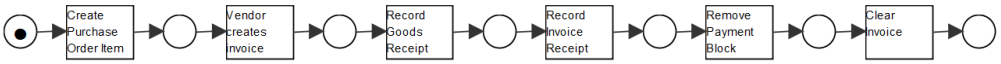

Figure 2: Petri net of the selected control-flow variant in the BPI Challenge 2019 dataset, illustrating a purchase order business process.

5.2. Experiment Results

GDT_SPN_*k*NN is tested against four benchmarks: the average duration of the full training set (Average), the average duration of the ten nearest neighbors of the candidate trace's prefix (Average 10 *k*NN), the average duration of the hundred nearest neighbors (Average 100 *k*NN), and the algorithm as proposed by Rogge-Solti and Weske (GDT_SPN) [4, 5]. One should note that for later iterations, only few to-be-predicted traces remain in the test set and that the results can thus be volatile and less informative towards the very end. The outcome of these experiments is visualised in Figure 3 and Figure 4, where the former reports the mean errors and the latter reports the root mean square errors. Note that in these figures our prediction algorithm is referred to as *GDT_SPN_kNN*. The x-axis represents the number of prediction iterations (as explained above), while the y-axis expresses the value of the corresponding metric in seconds.

In Figure 3 and Figure 4 no systematic under- or overestimation is observed. The bias compared to the benchmarks becomes smaller as more information about activities is known, i.e., as the prediction iteration goes up. In the vast majority of the prediction iterations, we achieve a higher accuracy than all benchmarks. The better predictions can be explained by two main factors. First, when there exists correlation between the times-to-occurrence of the different events, the algorithm can exploit this as soon as the first time-to-occurrence becomes available.

A second factor is that the Petri net for each cluster is more simple since it uses fewer trace variants, whereas the original GDT_SPN constructs a Petri net out of all training traces. This has a consequence in the further simulation of the Petri net as it is more likely that the ending is fixed, while in the more complex Petri net of [4, 5], multiple paths can be followed towards the final marking that actually belong to other trace variants. These trace variants might contain events portraying activity types not present in the test trace and might therefore yield worse predictions. Whenever the different trace variants have some matching activities, this factor becomes particularly important as the further simulation of the complete Petri net as in [4, 5] would be too volatile. It should be noted that although the results on the above data sets, on average, are showing significant improvements with regard to the stated benchmarks, there are some cases in which our approach does not yield better results. The predictions taken earlier on, at a lower iteration, show less difference between the tested methods as well. The benchmarks using simple averaging score not far off from the more elaborate GDT_SPN based methods. Later in the traces (for a higher iteration), the advantage of incorporating the time already passed on an activity, and possibly other information concerning the activities yet to come, result in more accurate predictions.

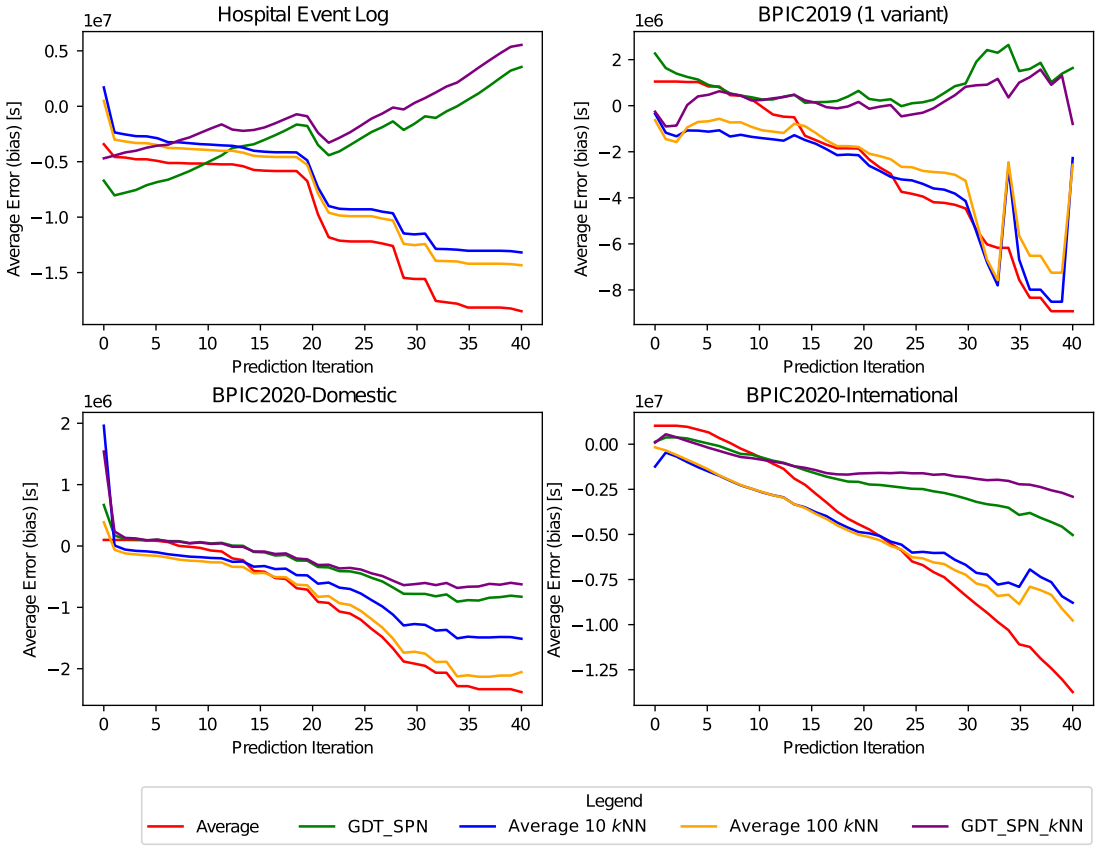

Figure 3: Mean errors of real-life experiments.

6. Conclusion and future work

In this paper, we constructed an approach for predicting the remaining time of a business process based on a combination of nearest neighbor selection and the GDT_SPN model as presented in [4, 5]. According to the four datasets we used to validate our model, we significantly outperform our four benchmarks, including the original method [4, 5]. The higher the correlation between the different times-to-occurrence and the more path variants with similar activities, the better our algorithm performs compared to the benchmarks. The GDT_SPN model itself is *white box*, and can therefore be used for explainability purposes. And while the kNN selection adds some complexity to the methodology, this does not obstruct explainability, and might even improve it when the discovered Petri nets are more simple.

Multiple assumptions were made in the experiments presented in this paper, such as putting the number of neighbors $k = 100$. The impact of these choices could be investigated further. Next to this, there are still multiple ways to build further on the prediction method presented in this paper, as both the choice of distance metric, clustering method and even the choice of using

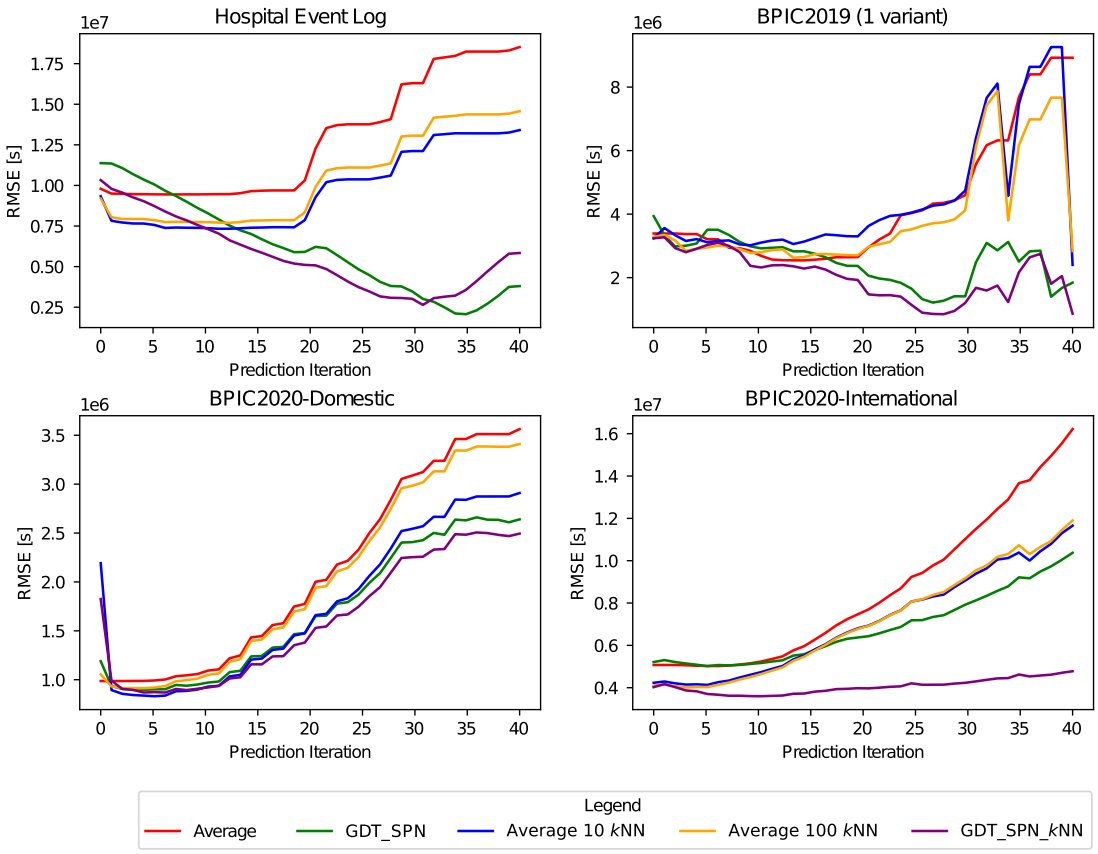

Figure 4: Root mean square errors of real-life experiments.

a GDT_SPN instead of something something else, is flexible and can easily be interchanged. One could develop a way to adjust the weighting of the activities by putting more weight on more relevant activities. This could potentially lead to better neighbor selection and hence better prediction. One could take into account trace and event variables while selecting nearest neighbors. A possible way of doing this is by a nested clustering approach, were one first clusters on either attributes or times-to-occurrence and afterwards reduces the number of neighbors by clustering on the other one. Furthermore, these extra attributes could be used to select the most relevant prefixes, when less than k prefixes can be found whose control-flow correspond to that of the case in question. In order to improve scalability, one could create fixed clusters and assign each test trace to the cluster it is most similar to. With such an eager clustering approach instead of the kNN algorithm, each cluster would have its own GDT_SPN model. These models can be built upfront and can directly be used to make a prediction from the moment the test trace is assigned to the corresponding cluster. While this may increase performance, the accuracy may drop, especially when the process is dynamic. Moreover more

experimentation into the impact and sensitivity of the results with different parameter values when setting up the inductive miner and different ways of matching neighbors, could provide some interesting insights. Moreover, investigating the true duration distributions might be interesting, as in this work we assumed Normal distributions. If the true distribution is not normal, learning different kinds of parametric (or even non-parametric) models might increase the prediction accuracy.

References

[1] M. Polato, A. Sperduti, A. Burattin, M. de Leoni, Time and activity sequence prediction of business process instances, Computing 100 (2018) 1005–1031.

[2] W. M. P. van der Aalst, Process Mining, Communications of the ACM. 55 (2012) 76–83.

[3] A. E. Marquez-Chamorro, M. Resinas, A. Ruiz-Cortes, Predictive monitoring of business processes: A survey, IEEE Transactions on Services Computing 11 (2018) 962–977.

[4] A. Rogge-Solti, M. Weske, Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 8274 LNCS (2013) 389–403.

[5] A. Rogge-Solti, M. Weske, Prediction of business process durations using non-Markovian stochastic Petri nets, Information Systems 54 (2015) 1–14.

[6] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Discovering block-structured process models from incomplete event logs, in: G. Ciardo, E. Kindler (Eds.), Application and Theory of Petri Nets and Concurrency, Springer International Publishing, Cham, 2014, pp. 91–110.

[7] I. Verenich, M. Dumas, M. La Rosa, F. M. Maggi, I. Teinemaa, Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring, ACM Transactions on Intelligent Systems and Technology 10 (2019).

[8] W. M. P. van der Aalst, M. H. Schonenberg, M. Song, Time prediction based on process mining, Information Systems 36 (2011) 450–475.

[9] F. Folino, M. Guarascio, L. Pontieri, Discovering context-aware models for predicting business process performances, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7565 LNCS (2012) 287–304.

[10] F. Folino, M. Guarascio, L. Pontieri, Discovering high-level performance models for ticket resolution processes, in: R. Meersman, H. Panetto, T. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. De Leenheer, D. Dou (Eds.), On the Move to Meaningful Internet Systems: OTM 2013 Conferences, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 275–282.

[11] C. Di Francescomarino, M. Dumas, F. M. Maggi, I. Teinemaa, Clustering-Based Predictive Process Monitoring, IEEE Transactions on Services Computing 12 (2019) 896–909.

[12] M. Polato, A. Sperduti, A. Burattin, M. de Leoni, Data-aware remaining time prediction of business process instances, Proceedings of the International Joint Conference on Neural Networks (2014) 816–823.

[13] M. de Leoni, W. M. van der Aalst, M. Dees, A general process mining framework for

correlating, predicting and clustering dynamic behavior based on event logs, Information Systems 56 (2016) 235–257. URL: https://www.sciencedirect.com/science/article/pii/S0306437915001313. doi:https://doi.org/10.1016/j.is.2015.07.003

[14] A. Senderovich, C. Di Francescomarino, C. Ghidini, K. Jorbina, F. M. Maggi, Intra and inter-case features in predictive process monitoring: A tale of two dimensions, Lecture Notes in Computer Science 10445 LNCS (2017) 306–323.

[15] S. V. D. Spoel, M. V. Keulen, C. Amrit, LNBIP 162 - Process Prediction in Noisy Data Sets: A Case Study in a Dutch Hospital, International Symposium on Data-Driven Process Discovery and Analysis (2013) 60–83.

[16] M. Ajmone Marsan, G. Conte, G. Balbo, A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems, ACM Transactions on Computer Systems (TOCS) 2 (1984) 93–122.

[17] M. Camargo, M. Dumas, O. González-Rojas, Learning Accurate LSTM Models of Business Processes, Lecture Notes in Computer Science 11675 LNCS (2019) 286–302.

[18] J. Evermann, J.-R. Rehse, P. Fettke, Predicting process behaviour using deep learning, Decision Support Systems 100 (2017) 129–140.

[19] N. Tax, I. Verenich, M. La Rosa, M. Dumas, Predictive business process monitoring with lstm neural networks, Lecture Notes in Computer Science (2017) 477–492.

[20] N. Mehdiyev, J. Evermann, P. Fettke, A multi-stage deep learning approach for business process event prediction, in: 2017 IEEE 19th Conference on Business Informatics (CBI), volume 01, 2017, pp. 119–128. doi:10.1109/CBI.2017..46

[21] L. Lin, L. Wen, J. Wang, MM-Pred: A Deep Predictive Model for Multi-attribute Event Sequence, 2019, pp. 118–126. doi:10.1137/1.9781611975673.14

[22] V. Pasquadibisceglie, A. Appice, G. Castellano, D. Malerba, Using convolutional neural networks for predictive process analytics, in: 2019 International Conference on Process Mining (ICPM), 2019, pp. 129–136. doi:10.1109/ICPM.2019.00028

[23] F. Taymouri, M. L. Rosa, S. Erfani, Z. D. Bozorgi, I. Verenich, Predictive business process monitoring via generative adversarial nets: The case of next event prediction, in: D. Fahland, C. Ghidini, J. Becker, M. Dumas (Eds.), Business Process Management, Springer International Publishing, Cham, 2020, pp. 237–256.

[24] Z. A. Bukhsh, A. Saeed, R. M. Dikman, Processtransformer: Predictive business process monitoring with transformer network, CoRR abs/2104.00721 (2021). URL: https://arxiv.org/abs/2104.00721. arXiv:2104.00721

[25] S. J. Leemans, D. Fahland, W. M. Van Der Aalst, Discovering block-structured process models from event logs - A constructive approach, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7927 LNCS (2013) 311–329.

[26] J. C. Buijs, B. F. Van Dongen, W. M. P. Van Der Aalst, Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity, International Journal of Cooperative Information Systems 23 (2014) 1–18.

[27] A. Berti, S. J. van Zelst, W. M. P. van der Aalst, Process mining for python (pm4py): Bridging the gap between process- and data science, CoRR abs/1905.06169 (2019). URL: http://arxiv.org/abs/1905.06169. arXiv:1905.06169

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel,

P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

Taking on Infrequent Behavior in Event Logs using Hypothesis Tests

Patrizia Schalk[1], Lisa Petrak[1]

[1]University of Augsburg, Universitätsstraße 6a, 86159 Augsburg

Abstract

When an event log is generated based on the real-life data of an existing process, there is a high probability that apart from events that happen frequently and therefore should be represented in the process model, infrequent behavior is featured in the event log that would make a model generated from this log hardly readable. There are many process discovery algorithms that work with such outliers by filtering infrequent behavior just before or during the creation of an appropriate process model. Less common methods make use of statistical tests to detect infrequent behavior in a preprocessing step. This paper aims to simplify the use of hypothesis tests introduced by Petrak et al. and proposes an approach to delete infrequent behavior without generating unwanted side effects for the process model. Furthermore, we solve a problem regarding the use of hypothesis tests for a process that contains loops and compare the results of hypothesis tests to well-known discovery techniques.

Keywords

Process Mining, Process Discovery, Filtering Outliers, Detection of Infrequent Behavior, Hypothesis Test, Directly Follows Graph, Loop Shortening

1. Introduction

Finding mathematical models for real-life processes is a highly relevant task, since such models can easily be analyzed in a way that shows exactly where the process has its weaknesses and where it can be optimized. However, constructing such a process model by hand leads to results that describe what the process should look like instead of how it actually looks. Therefore, it is highly desired to create such models automatically and analyze them afterwards, so there is no bias present in the model. To automatically construct such a model, a record of the behavior of the process, e.g. in the form of an event log, is needed. However, there are several challenges that make it hard to construct a model from an event log, especially if such a model has to meet some criteria like being readable.

One example of such a challenge is the existence of outliers in the event log – data that is present in the log but is so infrequent that it does not belong to the important parts of a process. Related to the existence of outliers is the existence of noise – data in the event log that cannot occur in the process. There are several causes for noise, such as human error ("I meant to log a different task than I actually did") or technical failure ("A part of the system shut down and didn't log a status during a period of time"). While noise is erroneous behavior and should not be featured in the event log, outliers can very well be part of the actual behavior of the process.

Algorithms & Theories for the Analysis of Event Data (ATAED) 2022

patrizia.schalk@informatik.uni-augsburg.de (P. Schalk); lisa.petrak@informatik.uni-augsburg.de (L. Petrak)

However, removing edge cases of the event log promises to make the remaining behavior and therefore a suitable process model more clear and readable. Finding such a reasonable model despite the existence of outliers in the event log is the task of process discovery. Most such techniques handle noise by using heuristics or by setting thresholds for the maximum amount of behavior that may be deleted [1, 2, 3, 4, 5, 6]. Fewer discovery techniques make use of statistical theory to detect infrequent behavior in event logs [7].

As mentioned before, a discovered model needs to meet some quality measures. For example, we want a process model that is able to replay most of the behavior present in the event log (but not everything, since infrequent behavior should be excluded) – this trait is measured by the *fitness* of a process model. However, fitness alone is not enough for a good process model, since we can accomplish perfect fitness by simply allowing every behavior, even behavior that has nothing to do with the process. Therefore, another quality measure is the *precision*, which ensures that the process model cannot replay way more than is present in the event log. The notion of *generalization* describes how open the model is for reasonable extensions, i.e. behavior that was not recorded in the log but can happen in the process. Finally, a process model should be simple and easy to read, so that a human can understand and analyze it – this trait is measured by the *simplicity* of the discovered process model. More information on quality measures for process models can be found in [8].

In this paper, we continue research on the question of how to discover a "good" process model for the main behavior of an event log by filtering out infrequent behavior. The idea to use hypothesis tests to find such outliers in an event log was presented in [7], where one-sided hypothesis tests are used to decide whether the direct neighborhood of two events should be classified as main or infrequent behavior. However, the approach of said paper creates unwanted side effects by deleting single events out of a trace and therefore creating new direct neighborhood relations that might be simply wrong. We present a new approach without this problem that is graphical, therefore easy to follow, and gives the user more power over the result he wishes to get. We first lay the foundation for our approach by defining needed notions in Section 2 and revisit the general idea to use hypothesis tests to find infrequent behavior in Section 3 by simplifying the used techniques described in [7] and investigating a running example that is small and easy to follow. In Section 4 we propose to use a Directly Follows Graph to represent the directly follows relations between events after we detected infrequent direct neighbors using hypothesis tests and show how we can delete infrequent behavior without risking side effects. We address a problem concerning loops that feature many iterations in the event log in Section 5 and propose a simple solution that is based on the well-studied Chinese Postman Problem [9]. Section 6 evaluates the results by comparing the accomplished fitness, precision, generalization and simplicity with those of well-known discovery techniques, namely the Inductive Miner infrequent [6] and the Directly-Follows Miner [5], with which our approach shares similar ideas. Section 7 concludes the paper.

2. Basic Definitions

We denote the set of natural numbers $\{1, 2, 3, \dots\}$ by $\mathbb{N}$ and define $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. For an arbitrary set T we call $m : T \to \mathbb{N}_0$ a *multiset* over T. For any $a \in T$, $m(a)$ is the number of occurences of the element a in the multiset m. We write $a \in m$ if this number is greater than zero, i.e. $a \in m :\Leftrightarrow m(a) > 0$. We also write a finite multiset m over T with elements $a_1, \dots, a_n$ in the form $[a_1^{m(a_1)}, \dots, a_n^{m(a_n)}]$.

Definition 1 (Event, Trace, Event log [10]). *Let T be a set of activity names. A sequence of activities $\sigma = \langle t_1, \dots, t_n \rangle \in T^*$ is called a* trace. *An activity $a \in T$ is* contained in *σ if it occurs in σ at any time, i.e. $a \in \sigma :\Leftrightarrow \sigma = \langle t_1, \dots, t_n \rangle \wedge \exists i \in \{1, \dots, n\} : t_i = a$. For an arbitrary trace $\sigma = \langle t_1, \dots, t_n \rangle \in T^*$ with $n \geq 1$ we define $first(\sigma) := t_1$ and $last(\sigma) := t_n$.*
An event log *$L : T^* \to \mathbb{N}_0$ over T is a multiset of traces. The occurrence of an activitiy $a \in T$ in L is called an* event.

Regarding a real-life process, we give every event that takes place a name, which can be an artificial one (like $a, b, c, \dots$) or a descriptive one (like $take\_order$). A trace then describes a sequence of events that happened for a special *case*, e.g. for a customer or a general object.

Definition 2 (Ordering relations [10]). *Let L be an event log over a set of activities T. For any $a, b \in T$ we introduce the following binary causal relations on T:*

- *$a >_L b$ if and only if a is directly followed by b somewhere in a trace σ in L, i.e. if a trace $\sigma = \langle t_1, \dots, t_n \rangle \in L$ and a number $i \in \{1, \dots, n-1\}$ exist, with $t_i = a$ and $t_{i+1} = b$*
- *$a \to_L b$ if and only if $a >_L b$ and $b \not>_L a$*
- *$a \leftarrow_L b$ if and only if $a \not>_L b$ and $b >_L a$*
- *$a \#_L b$ if and only if $a \not>_L b$ and $b \not>_L a$*
- *$a \|_L b$ if and only if $a >_L b$ and $b >_L a$.*

From these ordering relations $>_L$ will be of great interest for us, since we want to investigate the neighborhood of two events and decide whether two events directly following each other happen often (i.e. main behavior) or rarely (i.e. infrequent behavior). To do so, we need to know how often two events follow each other, which we can store in the *Correlation Matrix*:

Definition 3 (Correlation matrix). *Let L be an event log over T with $Start, End \notin T$. Further, take $T_S := T \cup \{Start\}$, $T_E := T \cup \{End\}$ and $\lambda := \langle\rangle$, the empty trace. We denote the number of times $a \in T_S$ is directly followed by $b \in T_E$ in all traces contained in L by $|(a, b)|_{>_L}$. The* correlation matrix *for an event log L is defined as the square matrix $C^L : T_S \times T_E \to \mathbb{N}_0$ with*

$$C^L_{a,b} = \begin{cases} |(a,b)|_{>_L} & \text{if } a, b \in T \\ \sum_{\sigma \in L \setminus \{\lambda\}, first(\sigma) = b} L(\sigma) & \text{if } a = Start, b \in T \\ \sum_{\sigma \in L \setminus \{\lambda\}, last(\sigma) = a} L(\sigma) & \text{if } a \in T, b = End \\ L(\lambda) & \text{if } a = Start, b = End. \end{cases}$$

Recall that L is a multiset and hence, for any $\sigma \in L$, $L(\sigma)$ denotes the number of times the trace σ is contained in L.

Apart from the Correlation Matrix, we can store the information related to which events are directly followed by one another in the *Directly Follows Graph*, which is a simple directed graph containing all events as vertices. Two vertices u and v are connected by an edge (u, v) if somewhere in the event log u is directly followed by v. The weight on the edge shows how often this happens in the event log.

Definition 4 (Directly Follows Graph). *Let L be an event log over a set of activities T. The* Directly Follows Graph *(DFG) for L is a weighted directed graph $G = (V, E, w)$ with*

- $V := T \cup \{Start, End\}$,
- $E := >_L \cup \{(Start, t) \mid \exists \sigma \in L : t = first(\sigma)\}$
  $\cup \{(t, End) \mid \exists \sigma \in L : t = last(\sigma)\}$ *and*
- $w : E \to \mathbb{N}_0$, $w((a, b)) := C^L_{a,b}$.

An example for a Directly Follows Graph will be given in Figure 1 in Section 4, where we first need the DFG. A useful property of the DFG is that it can easily be translated to a language-equivalent Petri-net. Leemans et al. [5] showed that if the DFG is *sound*, the respective language-equivalent Petri-net is also sound.

Definition 5 (Soundness of the Directly Follows Graph [5]). *Let $G = (V, E, w)$ be a DFG for an event log L. G is* sound *if every node $v \in V$ is on a path from $Start$ to End, i.e.*
$\forall v \in V : \exists u_1, \ldots, u_n : u_1 = Start \wedge$
$u_n = End \wedge$
$\exists j \in \{1, \ldots n\} : u_j = v \wedge$
$\forall i \in \{1, \ldots, n-1\} : (u_i, u_{i+1}) \in E$

As a running example for this paper we take a set of activities $T = \{a, b, c, d\}$ and an artificial event log L which is defined as

$$L = [\langle a, b, c, b\rangle^{100}, \langle a, c, b\rangle^{50}, \langle d, b, d\rangle^{100}, \langle b, e\rangle^{1000}, \langle d, e\rangle^{1000}, \langle f, g, f, g, f, g\rangle^{100}].$$

To construct the Correlation Matrix, we simply count how often an event $e_1 \in T$ is followed by an event e_2 in the event log and add this quantity to the Correlation Matrix in row e_1 and column e_2.

	End	a	b	c	d	e	f	g
$Start$	0	150	1000	0	1100	0	100	0
a	0	0	100	50	0	0	0	0
b	150	0	0	100	100	1000	0	0
c	0	0	150	0	0	0	0	0
d	100	0	100	0	0	1000	0	0
e	2000	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	300
g	100	0	0	0	0	0	200	0

The artificial event $Start$ (End) is used to record how often an event $e \in T$ is the first (last) event in a trace of L. The Correlation Matrix is very handy to calculate values needed for the execution of hypothesis tests, which are described in the next section.

3. Hypothesis Tests

Rather recently, Petrak et al. [7] showed in their article that hypothesis tests can be used to determine whether the direct neighborhood of two events that was observed in the event log should be classified as infrequent behavior, i.e. *noise*, or not. We shortly revisit this idea, simplify it slightly, and give some simple examples that show how this method differs from simply considering the direct neighborhood that is the least frequent in the event log as noise.

Generally speaking, hypothesis tests can be used to check whether a certain hypothesis is valid with a "sufficiently high" probability. Such a hypothesis makes a statement about a certain property of the objects in a population. As a simple example, take the set of all researchers in the field of process mining as the population and the statement *"At least 80% of researchers in the field of process mining like process discovery"*. Since this is a statement that can't be proved or falsified without asking every single researcher in the population (which would be a rather costly operation) one could easily doubt the correctness of this statement. So next to the already stated *Null-Hypothesis* H_0, we formulate an *Alternative Hypothesis* H_1 stating the contrary: *"Less than 80% of researchers in the field of process mining like process discovery"*. To check which of the two hypotheses is correct, we take a small sample of the population and check how many researchers of this population like process discovery. We then perform a left-sided hypothesis test to check whether this sample implies that the Null-Hypothesis H_0 or the Alternative Hypothesis H_1 is true.

In general, let $p \in [0, 1]$ be an unknown probability (in our example the actual percentile of researchers who love process discovery) and $p_0 \in [0, 1]$ a parameter given by the user (in our example $p_0 = 0.8$). Using data from a sample of the population, a left-sided hypothesis test can be used to decide whether $p \in [0, p_0[$ or $p \in [p_0, 1]$ is true, i.e. whether the *Null Hypothesis* H_0: $p \geq p_0$ or the *Alternative Hypothesis* H_1: $p < p_0$ is true. Since a hypothesis test comes to this decision based on a sample of the full data, there is a certain probability that the wrong decision will be made. The probability that this happens can be estimated, making it possible to formulate rather reliable statements about which of the two hypotheses holds.

The probability that we chose H_1 when in reality H_0 is true is called the α-error; the probability that we chose H_0 when in reality H_1 is true, is called β-error. In the context of this paper, we are especially interested in the α-error, which can be bounded by a given α when using hypothesis tests. We then want to find a value k such that for a random variable X (which can be understood as the number of elements in a random sample that fulfill the property defined in H_0) $P(X \leq k \mid H_0 \text{ is true}) \leq \alpha$ is true. The values of k for which this inequality holds can be determined by using the density-function of the given probability distribution. However, since this computation is quite time-consuming, we use an approximation for k if said probability distribution is a binomial distribution and the standard-deviation $\sigma_n := \sqrt{n \cdot p_0 \cdot (1 - p_0)}$ satisfies $\sigma_n > 3$. In this case, we approximate k by

$$k = \lceil np_0 - \sigma_n \cdot u_{1-\alpha} \rceil, \tag{1}$$

where $u_{1-\alpha}$ is the $(1 - \alpha)$-quantile of the standard normal distribution. A more formal description of hypothesis tests can be found in [11].

To detect noise in an event log, Petrak et al. [7] understand the neighborhood relation between two events as a binomial distribution and use hypothesis tests to check whether the sighting of two events $e_1, e_2 \in T$ directly following each other is main behavior (H_0 is true) or infrequent behavior (H_1 is true). To achieve this, they define the population as the pairs $(e, e') \in T$ that can follow each other in the process and where $e = e_1$ or $e' = e_2$ and view the event log as a sample where some of these direct neighborhoods were randomly drawn. Hence, the population is defined as:

$$P_{(e_1,e_2)} := \{(x, y) \in T_S \times T_E \mid (x = e_1 \vee y = e_2) \wedge |(x, y)|_{>L} > 0\},$$

which defines the sample size n as the number of events that can follow e_1 or can be followed by e_2 in the process:

$$n := |P_{(e_1,e_2)}| = \sum_{e \in T_E} |(e_1, e)|_{>L} + \sum_{e \in T_S} |(e, e_2)|_{>L} - |(e_1, e_2)|_{>L}.$$

With the user-given constants $1 - p_0$ and α they then execute a right-sided hypothesis test, which means, for a pair of events (e_1, e_2), they estimate k by $\lceil np_0 - \sigma_n \cdot u_{1-\alpha} \rceil$ and decide for H_0 when $n - |(e_1, e_2)|_{>L} < k$. Since this is equivalent to estimate k by the formula in Equation 1 and deciding for H_0 when $|(e_1, e_2)|_{>L} > k$, we can instead perform a left-sided hypothesis test with the probability p_0 as described at the start of this section.

With this idea we can calculate the critical value k for which $P(X \leq k \mid H_0 \text{ is true}) \leq \alpha$ holds and check afterwards whether $|(e_1, e_2)|_{>L} > k$. If this is true, we accept the Null Hypothesis H_0 and consider e_2 directly following e_1 as main behavior. Otherwise, we reject the Null Hypothesis, which means we accept the Alternative Hypothesis H_1 and consider e_2 directly following e_1 as infrequent behavior. However, if $\sigma_n \leq 3$, we need to calculate the critical value by evaluating the binomial distribution and finding a k for which the integral of the density function from 0 to k is less than $1 - \alpha$. For a more detailed description of this idea, see [7] and [11]. Using these ideas leads to the procedure shown in algorithm 1.

For our running example, we set $p_0 = 0.05$ (so we consider a pair of direct neighbors as infrequent if this behavior affects less than 5% of the population) and $\alpha = 0.05$ (so we accept that the probability for classifying main behavior as infrequent is at most 5%). We show the decision for whether a pair of direct neighbors is main or infrequent behavior on the following two examples:

Check $(a, c) \in >_L$:

	End	a	b	c	d	e	f	g
Start	0	150	1000	0	1100	0	100	0
a	0	0	100	50	0	0	0	0
b	150	0	0	100	100	1000	0	0
c	0	0	150	0	0	0	0	0
d	100	0	100	0	0	1000	0	0
e	2000	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	300
g	100	0	0	0	0	0	200	0

$n = 250$

$\sigma_n \approx 3.4 > 3$

$\Rightarrow k = 7$

$|(a, c)|_{>L} = 50 > 7 = k$

$\Rightarrow (a, c) \in >_L$ is main behavior.

Algorithm 1 Detecting main and infrequent neighbors

Input: $p_0 \in [0,1]$, $\alpha \in [0,1]$, $L : T^* \to \mathbb{N}_0$
Output: $main, infreq \subseteq (T \cup \{Start, End\})^2$

```
main ← ∅
infreq ← ∅
for (e1, e2) ∈ T ∪ {Start, End} do
    n ← Σ_{e∈T_E} |(e1, e)|_{>L} + Σ_{e∈T_S} |(e, e2)|_{>L} − |(e1, e2)|_{>L}
    σ_n ← sqrt(n · p0 · (1 − p0))
    if σ_n > 3 then
        k ← ⌈n p0 − σ_n · u_{1−α}⌉
    else
        sum = 0
        while sum < 1 − α do
            sum ← sum + C(n, k) · p0^k · (1 − p0)^{n−k}
            k ← k + 1
        end while
    end if
    if |(e1, e2)|_{>L} > k then                ▷ k > 0, so |(e1, e2)|_{>L} > 0.
        main ← main ∪ {(e1, e2)}
    else
        infreq ← infreq ∪ {(e1, e2)}
    end if
end for
return main, infreq
```

It is notable that in this example the direct neighborhood of the events a and c has the lowest frequency, but the result of the hypothesis test implies that a being directly followed by c is main behavior. This is due to the fact that a is only followed by b or c and c is only preceded by a or b. The number of sightings where b precedes c or b follows after a is rather small, so that the neighborhood between a and c can be considered as main behavior.

Check $(b,d) \in >_L$:

	End	a	b	c	d	e	f	g
$Start$	0	150	1000	0	1100	0	100	0
a	0	0	100	50	0	0	0	0
b	150	0	0	100	100	1000	0	0
c	0	0	150	0	0	0	0	0
d	100	0	100	0	0	1000	0	0
e	2000	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	300
g	100	0	0	0	0	0	200	0

$n = 2450$
$\sigma_n \approx 10.7 > 3$
$\Rightarrow k = 105$
$|(b,d)|_{>_L} = 100 \leq 105 = k$

$\Rightarrow (b,d) \in >_L$ is infrequent behavior.

Executing the hypothesis test for every pair of events leads to the result that the pairs (b, d), (b, End), (d, b), (d, End) and $(Start, f)$, of direct neighbors are infrequent. Petrak et al. propose to simply delete these direct neighborhood relations from the footprint and use the footprint that was constructed in this way to mine a process model.

4. Using the results to construct a DFG

We use the approach presented by Petrak et al., but construct a Directly Follows Graph (see Definition 4) instead of a footprint. Gaining a Petri-net from a DFG is rather simple, and we can easily check the soundness of a DFG by performing two Depth First Searches, which gives us the possibility to easily construct a DFG that is sound.

By applying the hypothesis tests to the event log with Algorithm 1 we immediately get a set $M \subseteq (T \cup \{Start, End\})^2$ of direct neighbors that are part of the main behavior of the event log and a set $I \subseteq (T \cup \{Start, End\})^2$ of direct neighbors that are infrequent behavior. Obviously, no direct neighborhood can be both main and infrequent behavior, so $M \cap I = \emptyset$. Let $G = (V, E)$ be the DFG for our event log. Since we only add behavior to M or I if we have seen the direct neighborhood at least once in the event log, $M \subseteq E$ and $I \subseteq E$ hold. On the other hand, due to the definition of E, there is no pair of events in $M \cup I$ that is not present in E, since E contains every pair of direct neighbors that was observed in the event log. Therefore, we can conclude that $M \cup I = E$. The DFG for our running example is shown in Figure 1, the black edges are those of the set M and the highlighted edges are those of the set I.

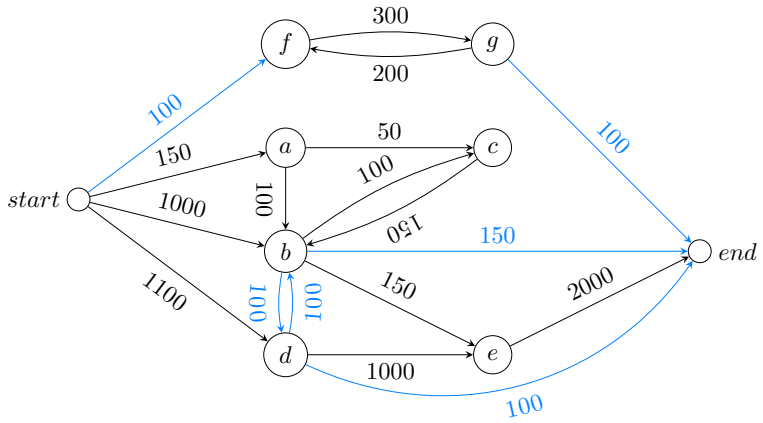

Figure 1: The DFG for the event log of our running example L. Blue edges were classified as infrequent behavior, black edges were classified as main behavior by the hypothesis tests.

When deleting edges from the DFG, we need to be careful, since the deletion of the whole set I may lead to a result that is not sound according to Definition 5. This is also the case for the DFG in Figure 1, since without all the highlighted edges, the events f and g do not lie on a path

from $Start$ to End. A process model constructed directly from such a DFG is not sound, so the goal is to delete as many infrequent edges as possible without sacrificing the soundness of the DFG. Deleting every highlighted edge except the edges $(Start, f)$ and (g, End) leads to the result shown in Figure 2.

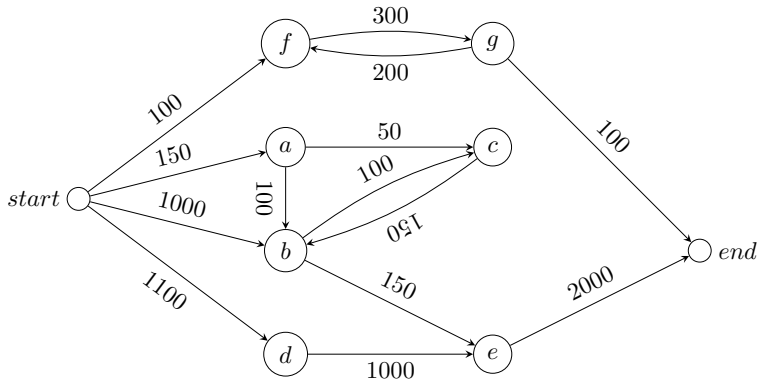

Figure 2: The modified DFG for our running example log L, where infrequent edges that were not necessary for the soundness were deleted.

In general, it is not always clear which set $J \subseteq I$ should be deleted from the DFG, since the deletion of an edge e could lead to a situation where other edges are needed for soundness, whereas said edges would not be needed if e wasn't deleted. Finding the "best" subset of I to delete from the DFG is a difficult problem, since it is not clear when one subset is better than another. If we define a function $f : \mathcal{P}(I) \to \mathbb{N}_0$ which maps every subset of I to a numerical value, where a high value indicates that the subset is a better candidate for deletion than another candidate with a lower value, we have an optimization problem at hand: Maximize $f(J)$ where $G' = (V, E \setminus J)$ is a sound DFG and $J \subseteq I$. As a default for f we propose $f(J) := |J|$, so that edge sets that contain more infrequent edges are preferred for deletion. A naive implementation that solves this problem can be found in Algorithm 2.

Algorithm 2 Deleting infrequent edges from the DFG

Input: $G = (V, E)$, $M, I \subseteq E$ with $M \cup I = E$ and $M \cap I = \emptyset$, $f : \mathcal{P}(I) \to \mathbb{N}_0$
Output: $G' = (V, E')$

```
J_opt ← ∅
for each J ⊆ I do
    G'' ← (V, E \ J)
    if f(J) > f(J_opt) and G'' is sound then
        J_opt ← J
    end if
end for
return (V, E \ J_opt)
```

This algorithm needs $O(2^{|I|} \cdot (|V| + |E|))$ time and is therefore exponential in time, but since

the number of infrequent edges, and therefore the cardinality $|I|$ of I, is rather small in the most cases, we accept this naive implementation, since it allows for an easy exchange of the user-given function f that should be optimized, and is easy to read. In our running example, this algorithm leads to the sound DFG shown in Figure 2.

5. Handling of Loops

As can be seen in the example of Figure 1, loops in the DFG can have a high impact on the population for our hypothesis tests. In this example, executing the loop between f and g some times artificially increases the number of times when f was preceded by another event than $Start$ and the number of times when after g followed another event than End. Even though the edges $(Start, f)$ and (g, End) are only encountered in the same trace as the edges (f, g) and (g, f), the repetition of this loop leads to the effect that $(Start, f)$ and (g, End) are classified as infrequent behavior. In the example of Figure 1 this is not a problem, since these two infrequent edges are not deleted for the sake of soundness. However, the following simple event log shows that we are not always in such a lucky situation:

$$L_{\circlearrowleft} = [\langle a, \underbrace{b, \ldots, b}_{51 \text{ times}}, d\rangle^{10}, \langle a, b, c, d\rangle^{40}, \langle a, c, d\rangle^{100}]$$

Executing hypothesis tests on this log leads to the classification of the direct neighborhood between b and d as infrequent.

Check $(b, d) \in >_{L_{\circlearrowleft}}$:

	End	a	b	c	d
$Start$	0	150	0	0	0
a	0	0	50	100	0
b	0	0	500	40	10
c	0	0	0	0	140
d	150	0	0	0	0

$n = 690$

$\sigma_n \approx 5.7 > 3$

$\Rightarrow k = 26$

$|(b, d)|_{>_L} = 10 \leq 26 = k$

$\Rightarrow (b, d) \in >_{L_{\circlearrowleft}}$ is infrequent behavior.

Every other direct neighborhood is classified as main behavior by the hypothesis tests, so we get the DFG for $\mathcal{L}$ that is shown in Figure 3.

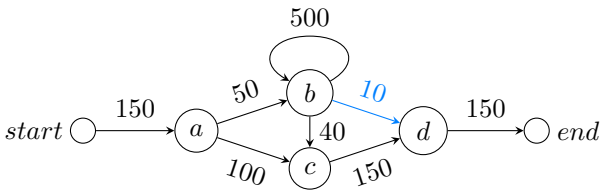

Figure 3: The DFG for the event log $L_{\circlearrowleft}$ with the only infrequent edge highlighted in blue.

Deletion of the highlighted edge would not violate the soundness of the DFG, but doing so would remove the entire behavior of a being followed by one or more bs, which is followed by a

single d. This is a problem, since the neighborhood (b, d) would not be classified as infrequent behavior if the event log would feature fewer iterations of the b-loop:

Check $(b, d) \in >_{L_\circlearrowleft}$:

	End	a	b	c	d
$Start$	0	150	0	0	0
a	0	0	50	100	0
b	0	0	10	40	10
c	0	0	0	0	140
d	150	0	0	0	0

$n = 200$
$\sigma_n \approx 3.08 > 3$
$\Rightarrow k = 5$
$|(b, d)|_{>_L} = 10 > 5 = k$

$\Rightarrow (b, d) \in >_{L_\circlearrowleft}$ is main behavior.

To eliminate this problem, we propose a preprocessing step for the hypothesis tests to reduce the number of loop iterations if there is a loop present in the process. To do so, we alter the traces of our event log that contain such a loop for the hypothesis tests. The idea is to construct a DFG for this single trace and find a path from $Start$ to End that cycles less frequently through the loop in the trace, if this is possible. To do so, we firstly introduce the notion of a *Trace DFG*:

Definition 6 (Trace DFG). *Let $\sigma = \langle t_1, \ldots, t_n \rangle$ be a trace. The* Trace DFG *for σ is a directed multigraph $G = (V, E)$ with*

- $V := \{Start, End\} \cup \{t_1, \ldots, t_n\}$ *and*
- $E : (T \cup \{Start, End\})^2 \to \mathbb{N}_0$,

$$E((t_1, t_2)) = \begin{cases} 1 & \text{if } t_1 = Start \text{ or } t_2 = End \\ |\{i \mid t_i = t_1 \wedge t_{i+1} = t_2\}| & \text{otherwise} \end{cases}$$

For a trace σ, the Trace DFG is therefore the DFG of the log $[\sigma]$, where the edge weights of said DFG are interpreted as multiple edges. The Trace DFG for the trace $\langle a, b, c, d \rangle$ for example is a simple path, the Trace DFG for the trace $\langle a, \underbrace{b, \ldots, b}_{51 \text{ times}}, d \rangle$ is shown in Figure 4.

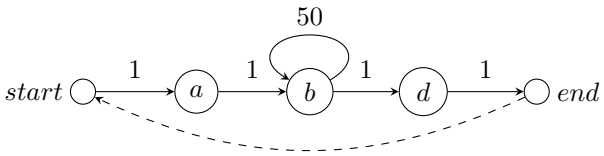

Figure 4: The Trace DFG for $\langle a, b, \ldots, b, d \rangle \in \mathcal{L}$. Edge weights denote the multiplicity of edges, the artificial dashed edge (which is not part of the Trace DFG) makes the DFG eulerian.

To reduce the number of loop iterations in the Trace DFG we first introduce a new, artificial edge from End to $Start$, as shown by the dashed dart in Figure 4. The resulting multigraph G is eulerian, which means there is a eulerian cycle in G.

Theorem 1. *Let $\sigma \in T^*$ be a trace and $G = (V, E)$ the Trace DFG of σ. Then, the graph $G' = (V, E \cup \{(End, Start)\}$ is eulerian, i.e. contains a eulerian cycle.*

Proof. By construction of the Trace DFG, σ is a path from $Start$ to End that uses every edge in G exactly once. With the artificial edge $(End, Start)$ in G' this path becomes a cycle that uses every edge in G' exactly once, i.e. a eulerian cycle.

Since this extended Trace DFG G' is eulerian, we can find the shortest cycle through G' that uses every type of edge in G' at least once, but no more than the multiplicity of the edges allow. The problem of finding this shortest cycle is known as the Chinese Postman Problem, which was introduced by Edmonds et al. [9] and can be solved in $O(|V|^3)$ time. In the example for $L_{\circlearrowleft}$, the shortest cycle is $(start, a, b, b, d, end, start)$, which cycles only once through the loop at the vertex b. From this cycle we can easily construct the shortened trace $\langle a, b, b, d\rangle$ that features less loop iterations than the original trace and hence does not disrupt the hypothesis tests. We therefore use this trace instead of the original trace to execute the hypothesis tests and find that the neighborhood (b, d) is main behavior. However, when constructing the DFG for $L_{\circlearrowleft}$, we calculate the multiplicities based on the original (i.e. not shortened) traces.

6. Evaluation

We implemented the algorithm described in this paper in `python`, version `3.8`, using the process mining library `pm4py` [12]. The latter provides functions to calculate the fitness, precision, generalization and simplicity for a given Petri net. To calculate these quality-measures, we used the following techniques provided by `pm4py`:

- **Fitness:** The alignment-based fitness is computed and the percentage of completely fit traces is returned, as well as the average fitness value of the single traces,
- **Precision:** The Align-ETConformance is computed and returned, see [13],
- **Generalization:** Computed as described in [14],
- **Simplicity:** Computed as described in [15].

To check the Petri net for soundness, the external tool WOFLAN [16] is used. Aside from `pm4py` there are other tools, like Entropia [17], that are also interesting for efficient and robust conformance checking. We plan to investigate these tools in the future and use them to further evaluate our results.

We chose some real-life event logs that are often used to evaluate new process discovery techniques, to test and compare our implementation to other mining-techniques. We used

- `BPI_Challenge_2012.xes` (**BPI12**), which can be found at [18],
- `DomesticDeclarations.xes` (**DD**),
- `InternationalDeclarations.xes` (**ID**),
- `PermitLog.xes` (**PL**),
- `PrepaidTravelCost.xes` (**PTC**),
- `RequestForPayment.xes` (**RFP**), all of which can be found at [19].

Table 1
Characteristics of the event logs that were chosen for the evaluation.

	BPI12	DD	ID	PL	PTC	RFP
Number of unique traces	4 366	99	753	1 478	202	89
Total number of traces	13 087	10 500	6 449	7 065	2 099	6 886
Number of cycles	43 566	121	585	3 842	113	80
Longest Trace Length	175	24	27	90	21	20
Shortest Trace Length	3	1	3	3	1	1
Average Trace Length	20.04	5.37	11.19	12.25	8.69	5.34

Table 1 gives an overview over some important characteristics of these event logs.

First, we compared our approach with the fixed parameters $p_0 = 0.05$ and $\alpha = 0.05$ with the following well-known mining approaches that consider outliers in the event log:

- The *Inductive Miner infrequent* (IMi) [6] creates a process tree out of the DFG for an event log and creates a Petri net from that process tree. In a preprocessing step, all edges that are "too infrequent" are removed, by going through all events e and deleting all outgoing edges of e with a frequency less than k times the frequency of the strongest outgoing edge of e, where k is a user-chosen parameter between 0 and 1. We chose 0.2 as the parameter for IMi.
- The *Directly Follows Miner* (DFM) [5] creates a DFG and scans it for infrequent behavior by determining the edges with the least frequency. Then all traces that feature the direct neighborhood modeled with an infrequent edge are removed from a copy of the event log. This step is repeated until at most a fraction of t traces were removed, where t is a user-chosen threshold between 0 and 1. Then, the DFG for the new event log is computed (or the old DFG is adjusted accordingly) and a Petri net is constructed from the DFG. We chose 0.1 as the parameter for DFM.

We also compared our approach to the technique presented in [7], where hypothesis tests are executed, and all infrequent edges are deleted from the DFG without the goal to maintain soundness. For this approach, we chose $p_0 = 0.95$ and $\alpha = 0.05$, so the hypothesis tests are performed exactly like with the new technique.

Since the DFM and our approach result in a DFG instead of a Petri net, we also need to convert the DFG into a Petri net to compute the quality measures as implemented in `pm4py`. To do so we constructed a labeled Petri net $N = (P, T, F, l)$ from a DFG $G = (V, E, w)$ as follows:

- For each node $v \in V$ a place $v \in P$ is added.
- For each edge $(u, v) \in V \times (V \setminus \{End\})$, a transition $t_{u,v} \in T$ with $l(t_{u,v}) = v$ is added.
- For each edge $(u, End) \in V \times \{End\}$, we also add an edge $t_{u,End} \in T$. However, this is a silent transition, i.e. $l(t_{u,End}) = \tau$.
- Finally, for each transition $t_{u,v} \in T$ two arcs are added: $(u, t_{u,v}) \in F$ and $(t_{u,v}, v) \in F$.

The results of executing the algorithms and the conformance checking methods on the resulting Petri nets can be found in Figure 5. For all evaluated methods and logs, Fitness and Generalization were similarly good. Simplicity also showed consistently good values, except for

a few very good results from the DFM. The percentage of fitting traces was consistently very good for DFM, while the other techniques achieved variously good values for different logs. The largest differences between the methods were observed for the Precision and the F-Score. Here, the results for different logs vary greatly for the DFM, while IMi achieves rather low results. Both of our new approaches achieve very good values most of the time in these two categories.

It is notable that the approach of [7] (which we call HT) always leads to a better precision than our new approach, which is not surprising considering that we leave some infrequent edges in the graph to maintain soundness. For example, if there is a single trace featuring much infrequent behavior, from which almost everything is deleted from the DFG except for an edge e, it is possible to use this edge e in the resulting Petri Net, even though it does not fit to any behavior seen in the event log. Due to the higher precision, the F-Score is also better than in our approach. Further, the simplicity of the net mined by HT is often much higher than in the new approach, simply because more edges are deleted. However, contrary to our new approach, HT never led to a sound net for any of the tested event logs.

IMi scores similary to our approach, except for precision and therefore F-Score. This is due to the fact that IMi classifies edges as infrequent with a rather *local* argument – the frequency of an edge is compared to the frequencies of all other outgoing edges, but not to the frequencies in the full graph. Hence, IMi may often delete single edges from many traces. For example, for the very simple event log $[\langle a, b, c\rangle^l]$, it can easily be forced to delete the edge (b, c) by introducing a trace $\langle x, b, y\rangle^{\frac{l}{k}+1}$ to the event log. Then, the edge (b, c) in the DFG becomes infrequent compared to the other outgoing edge (b, y) and is therefore deleted. Then, the trace $\langle a, b, c\rangle$ can not be replayed, which leads to a worse fitness, but the trace $\langle a, b, y\rangle$ can be replayed, even though it is not present in the event log.

With the user-chosen threshold that affects the behavior of the DFM, we have direct control over the fitness of the mined model, so this metric features always very good values. However, since DFM deletes only traces and not single edges, it seems odd that the precision of DFM is as low as it is for some event logs. This may be associated with the fact that in the DFG there is exactly one vertex for every event present in the event log, and therefore some traces that share an event name but have nothing else in common lead to a Petri net where traces can be replayed that are not part of the event log. However, our approach shares the same problem concerning the DFG, which makes it unclear why the precision of our approaches have a much higher precision most of the time. To investigate this behavior further could be an interesting task for future work.

When comparing the simpler version of our method to the one with loop reduction, it can be seen that both versions achieve similar results, which can be explained by the structure of the chosen event logs. It may be interesting to investigate further how the two variants differ on event logs with many loops.

During the execution of the different algorithms, the times required to construct a model in each case were measured. These do not include the calculation of the metrics that were afterwards calculated on it. In Table 2, all execution times are given in seconds. Hence, for all event logs and algorithms evaluated, a result could be calculated within a few seconds.

Table 2
The execution times in seconds for each investigated approach and each chosen event log.

	BPI12	DD	ID	PL	PTC	RFP
HTs	00.87397 s	00.09229 s	00.20281 s	02.73899 s	00.07453 s	00.06550 s
HTl	02.69569 s	00.10818 s	00.36392 s	01.00158 s	00.10343 s	00.07757 s
HT	05.493242 s	00.09222 s	00.18860 s	00.34547 s	00.05466 s	00.06677 s
IMi	12.15239 s	01.67628 s	03.43188 s	07.16169 s	01.54955 s	01.86956 s
DFM	01.99909 s	00.14701 s	00.31106 s	00.68258 s	00.05291 s	00.09879 s

7. Conclusion

In this paper, we revisited the use of hypothesis tests of Petrak et al. [7] to detect infrequent neighborhood relations in a given event log. We proposed a simpler variant for these tests in the form of a left-sided hypothesis test and changed the output to a Directly Follows Graph instead of a footprint, in order to be able to selectively remove infrequent behavior without destroying the soundness of the resulting DFG. Furthermore, we showed theoretically that high loop iterations in an event log can lead to undesired results of the hypothesis tests and solved this problem by implementing a preprocessing step that reduces the number of loop iterations to an amount small enough that the sample size used for the hypothesis tests is not bloated by the loop. The proposed techniques are very easy to understand and implement. We compared both the hypothesis tests without loop-shortening and the hypothesis tests with loop-shortening with the results of the old technique using hypothesis tests [7], the Inductive Miner infrequent [6] and the Directly Follows Miner [5] and found that our results for fitness, precision, generalization, simplicity and F-score are rather consistent and high – also compared to these techniques.

Further research should include the evaluation of logs with high loop iterations to verify that the shortening of loops is not only reasonable in theory, but also in practice. It is interesting how often the loop-shortening is relevant in real-life logs and under what circumstances (concerning the underlying process) the necessity of loop-shortening is probable. Also, there are some more techniques that outliers before mining a process model [20, 21, 22], some of which also use statistical ideas. It would be very interesting to see how our approach competes to these techniques. Since the DFG loses information on concurrency, a preprocessing step to detect concurrency may be a topic of further research – as well as the investigation on the existence of event logs that lead to bad results with our approach due to concurrency.

References

[1] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour, in: Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers, volume 171 of *Lecture Notes in Business Information Processing*, 2013, pp. 66–78.

[2] C. W. Günther, W. M. P. van der Aalst, Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics, in: Business Process Management, 2007, pp. 328–343.

[3] A. Weijters, W. Aalst, van der, A. Alves De Medeiros, Process mining with the HeuristicsMiner algorithm, BETA publicatie : working papers, Technische Universiteit Eindhoven, 2006.

[4] A. J. M. M. Weijters, J. T. S. Ribeiro, Flexible Heuristics Miner (FHM), in: 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2011, pp. 310–317.

[5] S. J. J. Leemans, E. Poppe, M. T. Wynn, Directly Follows-Based Process Mining: Exploration & a Case Study, in: International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24-26, 2019, IEEE, 2019, pp. 25–32.

[6] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Discovering Block-Structured Process Models from Incomplete Event Logs, in: G. Ciardo, E. Kindler (Eds.), Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings, volume 8489 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 91–110.

[7] L. Petrak, R. Lorenz, Detecting Infrequent Behavior in Event Logs using Statistical Inference, in: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2020, June 24, 2020, volume 2625 of *CEUR Workshop Proceedings*, 2020, pp. 33–48.

[8] J. Carmona, B. F. van Dongen, A. Solti, M. Weidlich, Conformance Checking - Relating Processes and Models, Springer, 2018. URL: https://doi.org/10.1007/978-3-319-99414-7. doi:10.1007/978-3-319-99414-7.

[9] J. R. Edmonds, E. L. Johnson, Matching, Euler tours and the Chinese postman, Math. Program. 5 (1973) 88–124.

[10] W. M. van der Aalst, B. F. van Dongen, Discovering Petri Nets from Event Logs, in: Transactions on Petri Nets and Other Models of Concurrency VII, Springer, 2013, pp. 372–422.

[11] G. Hornsteiner, Daten und Statistik, Eine praktische Einführung für den Bachelor in Psychologie und Sozialwissenschaften. Berlin/Heidelberg (2012).

[12] A. Berti, S. J. Van Zelst, W. van der Aalst, Process Mining for Python (pm4py): bridging the gap between process-and data science, arXiv preprint arXiv:1905.06169 (2019).

[13] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. Dongen, W. Aalst, Measuring precision of modeled behavior, Information Systems and e-Business Management 13 (2014). doi:10.1007/s10257-014-0234-7.

[14] J. Buijs, B. Dongen, W. Aalst, Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity, International Journal of Cooperative Information Systems 23 (2014) 1440001. doi:10.1142/S0218843014400012.

[15] F. R. Blum, Metrics in process discovery, 2015.
[16] W. M. P. van der Aalst, Woflan: A Petri-net-based Workflow Analyzer, Systems analysis modelling simulation 35 (1999) 345–357. URL: https://publications.rwth-aachen.de/record/714622.
[17] A. Polyvyanyy, H. Alkhammash, C. D. Ciccio, L. García-Bañuelos, A. A. Kalenkova, S. J. J. Leemans, J. Mendling, A. Moffat, M. Weidlich, Entropia: A Family of Entropy-Based Conformance Checking Measures for Process Mining, CoRR abs/2008.09558 (2020). URL: https://arxiv.org/abs/2008.09558. arXiv:2008.09558.
[18] B. van Dongen, BPI Challenge 2012, 2012. URL: https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204/1. doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.
[19] B. van Dongen, BPI Challenge 2020, 2020. URL: https://data.4tu.nl/collections/BPI_Challenge_2020/5065541/1. doi:10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51.
[20] R. Conforti, M. L. Rosa, A. H. t. Hofstede, Filtering Out Infrequent Behavior from Business Process Event Logs, IEEE Transactions on Knowledge and Data Engineering 29 (2017) 300–314. doi:10.1109/TKDE.2016.2614680.
[21] X. Lu, D. Fahland, W. M. van der Aalst, Interactively Exploring Logs and Mining Models with Clustering, Filtering, and Relabeling, in: BPM, 2016.
[22] D. Brons, R. Scheepens, D. Fahland, Striking a new Balance in Accuracy and Simplicity with the Probabilistic Inductive Miner, in: 2021 3rd International Conference on Process Mining (ICPM), 2021, pp. 32–39. doi:10.1109/ICPM53251.2021.9576864.

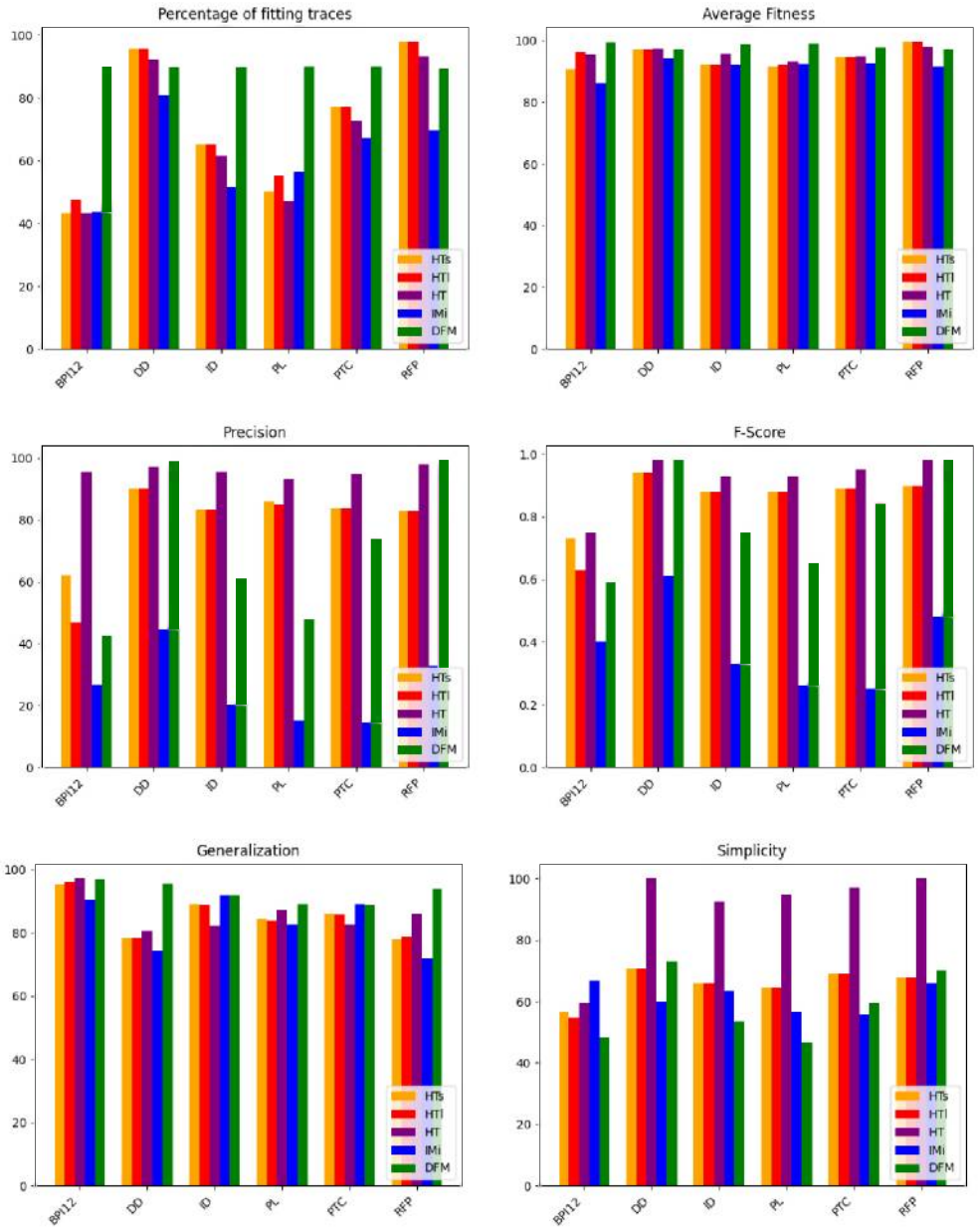

Figure 5: Comparison of the resulting quality-metrics for the approach described in this paper, using hypothesis tests and maintaining soundness (HTs), the same approach using loop-reduction (HTl) the old technique using hypothesis tests (HT), the Inductive Miner infrequent (IMi) and the Directly Follows Miner (DFM).

Minimising the Synthesised ENL-systems

Aishah Ahmed[1], Marta Pietkiewicz-Koutny[1,*]

[1]School of Computing, Newcastle University, Newcastle upon Tyne NE4 5TG, United Kingdom

Abstract

Elementary Net Systems with Localities (ENL-systems) is a class of Petri nets introduced to model GALS (globally asynchronous locally synchronous) systems, where some of the components might be considered as logically or physically close and acting synchronously, while others might be considered as loosely connected or residing at distant locations and communicating with the rest of the system in an asynchronous way. The specification of the behaviour of a GALS system comes very often in the form of a transition system. The automated synthesis, based on regions, is an approach that allows to construct Petri net models from their transition system specifications. In our previous papers we developed algorithms and tool support for the synthesis of ENL-systems from step transition systems, where arcs are labelled by steps (sets) of executed actions. In this paper we focus on the minimisation of the synthesised nets. In particular, we discuss the properties of minimal, companion, and complementary regions, and their role in the process of minimisation of ENL-systems. Furthermore, we propose a strategy to eliminate redundant regions. Our theoretical results are backed by experiments (the algorithms for the minimisation are implemented within the WORKCRAFT framework).

Keywords

theory of concurrency, Petri nets, localities, analysis and synthesis, step sequence semantics, theory of regions, transition systems, WORKCRAFT framework

1. Introduction

A number of computational systems exhibit behaviour that follows the 'globally asynchronous locally (maximally) synchronous' paradigm. Examples can be found in hardware design, where a VLSI chip may contain multiple clocks responsible for synchronising different subsets of gates [1], and in biologically inspired membrane systems representing cells within which biochemical reactions happen in synchronised pulses [2]. To formalise such systems, [3] introduced *Place/Transition-nets with localities* (PTL-nets), where each locality defines a distinct set of events which must be executed synchronously, i.e., in a maximally concurrent manner (often called *local maximal concurrency*).

An attractive way of constructing complex computing systems is their automated synthesis from behavioural specifications given in terms of suitable transition systems. In such a case, the synthesis procedure is often based on the regions of a transition system, a notion introduced in [4], and later used to solve the synthesis problem for different classes of Petri nets [5, 6, 7, 8, 9, 10]. A comprehensive survey of the synthesis problem and region theory is presented in [11].

ATAED'22: Algorithms & Theories for the Analysis of Event Data, June 20–21, 2022, Bergen, Norway

*Corresponding author.

a.ahmad@ncl.ac.uk (A. Ahmed); marta.koutny@ncl.ac.uk (M. Pietkiewicz-Koutny)

The vast majority of results in the area of synthesis of Petri nets use the standard transition systems, where the arcs are labelled with single events/actions, as initial specifications of systems' behaviour. In this paper, however, we follow the approach, used in [12, 13, 14, 15, 10], employing step transition systems instead, where arcs are labelled with sets of executed events/actions.

The nets with localities, as already mentioned, were first introduced in [3] using as a base a class of Place/Transition nets. The idea of actions' localities was later adapted to Elementary Net Systems (EN-systems) in [12], where a solution to the synthesis problem for ENL-systems was presented. Further advances in the area of synthesising nets with localities from step transition systems are the subjects of [13, 14, 15]. The last of them, [15], concentrated on finding the rules for reducing the number of regions that are essential to synthesise ENL-systems.

In our previous papers, [16, 17], we developed algorithms and tool support for the synthesis of ENL-systems from step transition systems. In this paper we continue the work started in [15] and focus on the minimisation of ENL-systems. The nets obtained from the synthesis procedure, called saturated nets, contain many conditions that are redundant from their behaviour point of view. Removing such conditions is important to get more manageable and readable solutions to the synthesis problem. The approaches to remove redundant conditions from nets were investigated in the literature and implemented in several tools [18, 19]. Many synthesis procedures concentrate on returning smaller solutions based on so-called minimal regions [20, 21, 22, 23, 24]. In our approach, minimal regions, as defined for our class of step transition systems, are also important for building smaller solutions to the synthesis problem. Furthermore, in this paper, following [6], we are interested in the role of minimal regions in defining state-machine components of the synthesised and minimised ENL-systems. Our theoretical results are backed by experiments (the algorithms for the minimisation are implemented within the WORKCRAFT framework [25, 26]).

To explain the basic idea behind ENL-systems, let us consider the net in Figure 1 modelling two co-located consumers and one producer residing in a remote location. In the initial state, the net can execute the singleton step $\{c_4\}$. Another enabled step is $\{p_2\}$ which removes the token from b_1 and puts two tokens, into b_0 and b_2. In this new state, there are three enabled steps, viz. $\{p_1\}$, $\{c_1, c_4\}$ and $\{p_1, c_1, c_4\}$. The last one, $\{p_1, c_1, c_4\}$, corresponds to what is usually called *maximal concurrency* as no more activities can be added to it without violating the constrains imposed by the available resources (represented by tokens). However, the previously enabled step $\{c_4\}$ which is still *resource (or token) enabled* is disallowed by the control mechanism of ENL-systems. It rejects a resource enabled step like $\{c_4\}$ since we can add to it c_1 co-located with c_4 obtaining a step which is resource enabled. In other words, the control mechanism employed by ENL-systems (and PTL-nets) is that of *local maximal concurrency* as indeed postulated by the GALS systems execution rule.

The paper is organised as follows. The next section recalls some basic notions concerning step transition systems as well as ENL-systems and their synthesis. Section 3 recalls, from [15], three reduction rules that can be used to safely eliminate redundant regions/conditions from the synthesised nets. Section 4 defines minimal regions for the class of ENLST-systems and Section 5 discusses the properties of different kinds of regions and their roles in the process of minimisation of ENL-systems. Section 6 presents a strategy to eliminate redundant regions from the synthesised nets. The paper ends with a conclusion that includes some directions for future work. Proofs are omitted due to the page limit.

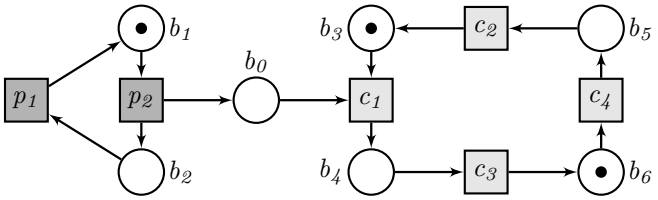

Figure 1: An ENL-system modelling a system comprising one producer and two co-located consumers (the shading of boxes indicates the co-location of events they represent).

2. Preliminaries

In this section, we recall suitably adapted notions and results from [12, 14, 15].

Throughout the paper, E is a fixed finite nonempty set of *events*. A *step* is a nonempty set of events, and a *co-location relation* $\simeq$ is any equivalence relation over E. For every event $e \in E$, $[e]_{\simeq}$ is the equivalence class of $\simeq$ to which e belongs (i.e., the *locality* of e). For an event e and a step U, we denote $e \simeq U$ whenever there is at least one event $f \in U$ satisfying $e \simeq f$.

Definition 1. *A* step transition system *(or* ST*-system) is a triple* $\mathfrak{ts} = (Q, A, q_0)$*, where* Q *is a nonempty finite set of* states*,* $A \subseteq Q \times (\mathbb{P}(E) \setminus \{\varnothing\}) \times Q$ *is a set of* transitions (arcs)*, and* $q_0 \in Q$ *is the* initial state. ♢

In diagrams, ST-systems are represented as labelled directed graphs, and singleton steps annotating transitions are denoted without brackets.

To ease the presentation, we assume that each event of E occurs in at least one of the steps labelling the transitions of $\mathfrak{ts}$.

The set of all steps labelling transitions outgoing from q will be denoted by $allSteps_q$. For a transition $\mathfrak{t} = (q, U, q') \in A$, we have therefore $U \in allSteps_q$, and respectively call q and q' the *source* and *target* of $\mathfrak{t}$. Furthermore, $\mathfrak{t}$ is *thick* if $|U| \geq 2$.

$\mathfrak{ts}$ is called *thin* if, for every event $e \in E$, there is $(q, \{e\}, q') \in A$.

A sequence of transitions $(q_1, U_1, q_2)(q_2, U_2, q_3) \ldots (q_k, U_k, q_{k+1})$ is a *path* from q_1 to q_{k+1}. A state q is *reachable* if there is a path from q_0 to q.

Two ST-systems, $\mathfrak{ts} = (Q, A, q_0)$ and $\mathfrak{ts}' = (Q', A', q_0')$, are *isomorphic* if there is a bijection $f : Q \to Q'$ such that $f(q_0) = q_0'$ and $A' = \{(f(q), U, f(q')) \mid (q, U, q') \in A\}$. We denote this by $\mathfrak{ts} \cong \mathfrak{ts}'$.

Definition 2. *An* elementary net system with localities w.r.t. a co-location relation $\simeq$ *(or* ENL$_{\simeq}$*-system) is a tuple* $\mathfrak{enl} = (B, E, F, \simeq, c_0)$*, where* B *is a finite set of* conditions *such that* $B \cap E = \varnothing$*,* $F \subseteq (B \times E) \cup (E \times B)$ *is the* flow relation*, and* $c_0 \subseteq B$ *is the* initial case *(in general, any* $c \subseteq B$ *is a* case*).*
We will also say that $\mathfrak{enl}$ *is an elementary net system with localities (or* ENL*-system) if mentioning* $\simeq$ *is not important.* ♢

In diagrams, conditions (local states) are represented by circles, events (actions) by boxes, the flow relation by directed arcs, and each case (global state) by tokens (small black dots) placed

inside those conditions which belong to this case. Moreover, boxes representing co-located events are shaded in the same way (see Figure 1).

For every event e, its *pre-conditions* and *post-conditions* are given respectively by ${}^\bullet e = \{b \mid (b,e) \in F\}$ and $e^\bullet = \{b \mid (e,b) \in F\}$, and both sets are assumed to be nonempty and disjoint. The dot-notation extends to sets of events in the usual way, e.g., ${}^\bullet U = \bigcup\{{}^\bullet e \mid e \in U\}$. Two distinct events, e and f, are *in conflict* (or conflicting) if they share a pre-condition, or share a post-condition. Furthermore, we assume that there are no isolated conditions in $\mathfrak{enl}$.

The semantics of $\mathfrak{enl}$ is based on steps of simultaneously executed events, and can be understood as *local maximal concurrency*. We first define *potential steps of* $\mathfrak{enl}$ as all nonempty sets of mutually non-conflicting events. A potential step U is then *resource enabled* at a case c if ${}^\bullet U \subseteq c$ and $U^\bullet \cap c = \varnothing$, and *control enabled* if, in addition, there is no event $e \notin U$ such that $e \nharpoonup U$ and the step $U \cup \{e\}$ is resource enabled at c. We denote these respectively by $U \in \mathit{resenabled}(c)$ and $U \in \mathit{enabled}(c)$. A control enabled step $U \in \mathit{enabled}(c)$ can be *executed* leading from c to the case $c' = (c \setminus {}^\bullet U) \cup U^\bullet$. We denote this by $c[U\rangle c'$.

The set of *reachable* cases of $\mathfrak{enl}$, denoted $\mathit{reach}_{\mathfrak{enl}}$, is the least set of cases containing c_0 such that if $c \in \mathit{reach}_{\mathfrak{enl}}$ and $c[U\rangle c'$, then $c' \in \mathit{reach}_{\mathfrak{enl}}$.

The ST-system *generated* by $\mathfrak{enl}$ is $\mathfrak{ts}_{\mathfrak{enl}} = (\mathit{reach}_{\mathfrak{enl}}, A, c_0)$, where $A = \{(c,U,c') \mid c \in \mathit{reach}_{\mathfrak{enl}} \wedge c[U\rangle c'\}$.

$\mathfrak{enl}$ is a *net realisation* of an ST-system $\mathfrak{ts}$ if $\mathfrak{ts}_{\mathfrak{enl}} \cong \mathfrak{ts}$.

To ease the presentation, we assume that $\mathfrak{enl}$ does not have *dead events*, i.e., for each event e, there are $c \in \mathit{reach}_{\mathfrak{enl}}$ and $U \in \mathit{enabled}(c)$ such that $e \in U$.

Definition 3. *Let $\mathfrak{enl} = (B, E, F, \nharpoonup, c_0)$ be an ENL$_\nharpoonup$-system. We say that $\mathfrak{enl}$ is a* state machine ENL$_\nharpoonup$*-system iff:*

1. $\forall e \in E : \ |{}^\bullet e| = 1 = |e^\bullet|$;
2. $|c_0| = 1$. ♢

Definition 4. *Let $\mathfrak{enl} = (B, E, F, \nharpoonup, c_0)$ be an ENL$_\nharpoonup$-system. A* subsystem *of $\mathfrak{enl}$ is an ENL$_\nharpoonup$-system $\mathfrak{enl}' = (B', E', F', \nharpoonup', c_0')$ such that the following conditions hold:*

1. *$B' \subseteq B$ and $E' \subseteq E$;*
2. *$\forall b \in B' \, \forall e \in E : \ ((b,e) \in F \vee (e,b) \in F) \implies e \in E'$;*
3. *$F' = F \cap ((B' \times E') \cup (E' \times B'))$; $\nharpoonup' = \nharpoonup \ \cap E' \times E'$ and $c_0' = c_0 \cap B'$.*

A subsystem $\mathfrak{enl}'$ is connected *if the graph $(B' \cup E', F')$ is connected. Also, point 2 above says that the subsystem $\mathfrak{enl}'$ is* generated by the subset of conditions *B' of the ENL$_\nharpoonup$-system $\mathfrak{enl}$.* ♢

Definition 5. *Let $\mathfrak{enl} = (B, E, F, \nharpoonup, c_0)$ be an ENL$_\nharpoonup$-system. We say that $\mathfrak{enl}$ is a* state machine decomposable *ENL$_\nharpoonup$-system iff there exists a set of connected subsystems of $\mathfrak{enl}$, $\mathfrak{enl}_i = (B_i, E_i, F_i, \nharpoonup_i, c_0^i)$ ($i = 1, \ldots, m$), satisfying the following:*

1. *$\forall i \in \{1, \ldots, m\}$: $\mathfrak{enl}_i$ is a state machine ENL$_{\nharpoonup_i}$-system;*

2. $B = \bigcup_i B_i,\ E = \bigcup_i E_i$ *and* $F = \bigcup_i F_i$.

The $\mathfrak{enl}_i$ *are called* state machine (or sequential) components *of* $\mathfrak{enl}$. ♢

The general synthesis problem we consider can be formulated thus:

Problem 1. *Given an* ST*-system* $\mathfrak{ts}$ *and a co-location relation* $\mathrel{\hat{-}}$*, find an effective way of checking whether there is an* ENL$_{\mathrel{\hat{-}}}$*-system which is a net realisation of* $\mathfrak{ts}$*. If the answer is positive construct such an* ENL$_{\mathrel{\hat{-}}}$*-system.* ♢

The above problem can be approached by considering a link between the nodes (global states) of an ST-system with the conditions (local states) of a hypothetical ENL-system realising it, captured by the notion of regions with explicit input and output events.

Definition 6. *A* region (with explicit input and output events) *of an* ST*-system* $\mathfrak{ts} = (Q, A, q_0)$ *is a triple* $\mathfrak{r} = (in, r, out) \in \mathbb{P}(E) \times \mathbb{P}(Q) \times \mathbb{P}(E)$*, such that* $in = out = \varnothing$ *implies* $r = Q$ *or* $r = \varnothing$ *and, for every transition* (q, U, q') *of* $\mathfrak{ts}$*, the following hold:*

R1 *If* $q \in r$ *and* $q' \notin r$ *then* $|U \cap out| = 1$.

R2 *If* $q \notin r$ *and* $q' \in r$ *then* $|U \cap in| = 1$.

R3 *If* $U \cap out \neq \varnothing$ *then* $q \in r$ *and* $q' \notin r$.

R4 *If* $U \cap in \neq \varnothing$ *then* $q \notin r$ *and* $q' \in r$. ♢

In a region $\mathfrak{r} = (in, r, out)$, the set *in* comprises events responsible for entering the set of states r, and *out* comprises events responsible for leaving r. Note that $\bar{\mathfrak{r}} = (out, Q \setminus r, in)$ is also a region (the *complement* of region $\mathfrak{r}$).

In general, a region $\mathfrak{r}$ cannot be identified only by its set of states r. However, if $\mathfrak{ts}$ is *thin*, then its different regions are based on different sets of states.

There are exactly two *trivial* regions satisfying $r = \varnothing$ or $r = Q$, viz. $(\varnothing, \varnothing, \varnothing)$ and $(\varnothing, Q, \varnothing)$.

The set of all non-trivial regions of $\mathfrak{ts}$ will be denoted by $\mathfrak{R}_{\mathfrak{ts}}$ and, for every state q, $\mathfrak{R}_q = \{\mathfrak{r} \in \mathfrak{R}_{\mathfrak{ts}} \mid q \in r\}$ is the set of all non-trivial regions (in, r, out) containing q.

The sets of *pre-regions* and *post-regions* of an event e, ${}^\circ e$ and e°, comprise all the non-trivial regions (in, r, out) respectively satisfying $e \in out$ and $e \in in$, viz. ${}^\circ e = \{\mathfrak{r} \in \mathfrak{R}_{\mathfrak{ts}} \mid e \in out\}$ and $e^\circ = \{\mathfrak{r} \in \mathfrak{R}_{\mathfrak{ts}} \mid e \in in\}$. This extends in the usual way to sets of events, for example, ${}^\circ U = \bigcup\{{}^\circ e \mid e \in U\}$. Also, we will write $e \in \mathfrak{r}^\circ \iff \mathfrak{r} \in {}^\circ e$ and $e \in {}^\circ\mathfrak{r} \iff \mathfrak{r} \in e^\circ$.

The set of *potential steps* of $\mathfrak{ts}$ comprises all nonempty sets U of events such that ${}^\circ e \cap {}^\circ f = e^\circ \cap f^\circ = \varnothing$, for each pair of distinct events $e, f \in U$. A potential step U is then *region enabled* at $q \in Q$ if ${}^\circ U \subseteq \mathfrak{R}_q$ and $U^\circ \cap \mathfrak{R}_q = \varnothing$. We denote this by $U \in \mathit{regenabled}(q)$.

Definition 7. *An* ST*-system* $\mathfrak{ts}$ *is an* ENL step transition system *w.r.t. a co-location relation* $\mathrel{\hat{-}}$ *(or* ENLST$_{\mathrel{\hat{-}}}$*-system) if the following hold:*

A1 *Each state is reachable.*

A2 *For every event e, both $^\circ e$ and e° are nonempty.*

A3 *For all distinct states q and q', $\mathfrak{R}_q \neq \mathfrak{R}_{q'}$.*

A4 *For every state q and step U, $U \in allSteps_q$ iff $U \in regenabled(q)$ and there is no event $e \notin U$ such that $e \mathrel{\hat{-}} U$ and $U \cup \{e\} \in regenabled(q)$.*

We also say that $\mathfrak{ts}$ is an ENLST-system (if mentioning $\mathrel{\hat{-}}$ is not important). ♢

One can show that the ST-system generated by an ENL$_{\mathrel{\hat{-}}}$-system is an ENLST$_{\mathrel{\hat{-}}}$-system. The converse is also true, and a suitable translation is based on the regions of ST-systems.

Definition 8. *Let $\mathfrak{ts}$ be an ENLST$_{\mathrel{\hat{-}}}$-system. The tuple* associated *with $\mathfrak{ts}$ is defined by $\mathfrak{enl}^{\mathrel{\hat{-}}}_{\mathfrak{ts}} = (\mathfrak{R}_{\mathfrak{ts}}, E, F_{\mathfrak{ts}}, \mathrel{\hat{-}}, \mathfrak{R}_{q_0})$, where q_0 is the initial state of $\mathfrak{ts}$ and*

$$F_{\mathfrak{ts}} = \{(\mathfrak{r}, e) \in \mathfrak{R}_{\mathfrak{ts}} \times E \mid \mathfrak{r} \in {}^\circ e\} \cup \{(e, \mathfrak{r}) \in E \times \mathfrak{R}_{\mathfrak{ts}} \mid \mathfrak{r} \in e^\circ\}.$$

♢

The above construction always produces an ENL$_{\mathrel{\hat{-}}}$-system which generates an ST-system isomorphic to $\mathfrak{ts}$ (see [12]).

Theorem 1. *Let $\mathfrak{ts}$ be an ENLST$_{\mathrel{\hat{-}}}$-system. Then $\mathfrak{enl}^{\mathrel{\hat{-}}}_{\mathfrak{ts}}$ is an ENL$_{\mathrel{\hat{-}}}$-system such that $\mathfrak{ts} \cong \mathfrak{ts}_{\mathfrak{enl}^{\mathrel{\hat{-}}}_{\mathfrak{ts}}}$. Moreover, the unique isomorphism ψ between $\mathfrak{ts}$ and $\mathfrak{ts}_{\mathfrak{enl}^{\mathrel{\hat{-}}}_{\mathfrak{ts}}}$ is given by $\psi(q) = \mathfrak{R}_q$, for every state q of $\mathfrak{ts}$.*

3. Optimising solutions to the synthesis problem

In this section we recall some notions and results from [15].

The ENL-system $\mathfrak{enl}^{\mathrel{\hat{-}}}_{\mathfrak{ts}}$ obtained from the synthesis of the ENLST-system $\mathfrak{ts}$ may contain many conditions which are *redundant* from the point of view of its behaviour, *i.e.*, deletion of such conditions (and their adjacent arcs) would lead to a net that generates the ST-system, which is still isomorphic to $\mathfrak{ts}$.

Suppose that we have reduced $\mathfrak{enl}^{\mathrel{\hat{-}}}_{\mathfrak{ts}}$ in this way obtaining a sub-ENL-system $\mathfrak{enl}$. We would like to reduce $\mathfrak{enl}$ further by deleting a condition/region $\mathfrak{r}$ (and its adjacent arcs) without, as before, violating the property of it being an ENL-system [1] and making sure that the resultant net still generates the ST-system isomorphic to $\mathfrak{ts}$. We denote the ENL-system after such one step reduction: $\mathfrak{enl}_{\mathfrak{r}}$.

In [15], it was proved that complement regions are very often redundant:

Reduction Rule 1. *If $\mathfrak{r} = (in, r, out)$ and $\bar{\mathfrak{r}} = (out, Q \setminus r, in)$ are two conditions in $\mathfrak{enl}$ and deleting $\bar{\mathfrak{r}}$ leads to an ENL-system, then the ST-systems generated by $\mathfrak{enl}$ and $\mathfrak{enl}_{\bar{\mathfrak{r}}}$ are isomorphic and $\bar{\mathfrak{r}}$ is redundant.* ♢

[1] Every ENL-system $\mathfrak{enl} = (B, E, F, \mathrel{\hat{-}}, c_0)$ should satisfy: $\forall e \in E\ ({}^\bullet e \neq \varnothing \ \wedge\ e^\bullet \neq \varnothing \ \wedge\ {}^\bullet e \cap e^\bullet = \varnothing)$.

Another source of redundancy among conditions/regions in the synthesised net are "big" regions that are compositions of smaller regions. To define a composition operator, [15] introduces the concept of *compatible* regions.

Definition 9. *A region (in, r, out) is compatible with another region (in', r', out') iff the following three conditions hold:*

1. $r \cap r' = \varnothing$.
2. *For every $e \in out$ exactly one of the following holds:*
   - *For all the transitions (q, U, q') such that $e \in U$ we have $q' \in r'$.*
   - *For all the transitions (q, U, q') such that $e \in U$ we have $q' \notin r'$.*
3. *For every $e \in in$ exactly one of the following holds:*
   - *For all the transitions (q, U, q') such that $e \in U$ we have $q \in r'$.*
   - *For all the transitions (q, U, q') such that $e \in U$ we have $q \notin r'$.*

If region $\mathfrak{r}$ is compatible with region $\mathfrak{r}'$ and region $\mathfrak{r}'$ is compatible with $\mathfrak{r}$ we say that the two regions are compatible. ♢

In [15], it was proved that the *composition of two compatible regions* (defined below) is also a region. If $\mathfrak{r} = (in, r, out)$ and $\mathfrak{r}' = (in', r', out')$ are two non-trivial compatible regions of an ENLST-system $\mathfrak{ts}$, then the following is a (possibly trivial) region of $\mathfrak{ts}$:

$$\mathfrak{r} \oplus \mathfrak{r}' \stackrel{\text{df}}{=} (in \cup in' \setminus H, r \cup r', out \cup out' \setminus H),$$

where H is a set of events that belong only to steps labelling transitions hidden/buried in $r \cup r'$ (with its source in r and its target in r' or the other way round). The region $\mathfrak{r} \oplus \mathfrak{r}'$ is called the composition of $\mathfrak{r}$ and $\mathfrak{r}'$. In [15], the following reduction rule was proved:

Reduction Rule 2. *If $\mathfrak{r} = (in, r, out)$, $\mathfrak{r}' = (in', r', out')$ and $\mathfrak{r} \oplus \mathfrak{r}'$ are three conditions/regions in $\mathfrak{enl}$, then the ST-systems generated by $\mathfrak{enl}$ and $\mathfrak{enl}_{\mathfrak{r} \oplus \mathfrak{r}'}$ are isomorphic and $\mathfrak{r} \oplus \mathfrak{r}'$ is redundant.* ♢

The third reduction rule considers regions of $\mathfrak{ts} = (Q, A, q_0)$ based on the same set of states. We will call such regions *companion* regions. For a given set of states r, they will belong to the set denoted by $\mathfrak{R}^r_{\mathfrak{ts}}$.

In [15], it was proved that if the events contained in the set in (out) of a region $\mathfrak{r} = (in, r, out)$ can be found in the in (out) sets of other companion regions then $\mathfrak{r}$ is redundant and can be deleted. Formally:

Reduction Rule 3. *Let $\mathfrak{r} = (in, r, out)$ be a condition/region of $\mathfrak{enl}$ such that:*

$$in \subseteq \bigcup \{in' \mid (in', r, out') \text{ is condition in } \mathfrak{enl} \text{ different from } \mathfrak{r}\} \tag{1}$$

$$out \subseteq \bigcup \{out' \mid (in', r, out') \text{ is condition in } \mathfrak{enl} \text{ different from } \mathfrak{r}\} \tag{2}$$

Then the ST-systems generated by $\mathfrak{enl}$ and $\mathfrak{enl}_{\mathfrak{r}}$ are isomorphic and $\mathfrak{r}$ is redundant. ♢

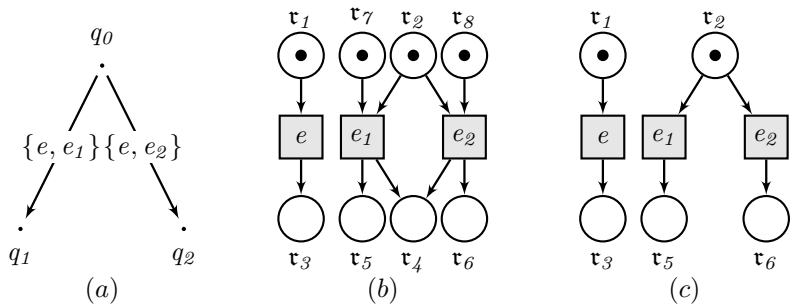

Figure 2: An ENLST-system with three co-located events e, e_1 and e_2 (a); the ENL-system resulting from its synthesis (b); and the reduced ENL-system solution for (a) that uses only minimal regions (c).

4. Minimal regions

For many classes of Petri nets, for which the synthesis problem was investigated, a region was defined as a subset of states of a transition system. For such classes of nets and their transition systems a minimal region was defined $w.r.t.$ the set inclusion $\subset$ [6, 7, 23, 27]. Also, composition of regions (as sets) was defined by using the set union operator ($\cup$), which is both commutative and associative.

The regions of ENLST-systems are triples of the form: $\mathfrak{r} = (in, r, out)$. The minimal regions in this class of (step) transition systems are defined $w.r.t.$ the strict pre-order $\prec$ on the set of regions, that utilises the idea of regions' composition by means of $\oplus$:

$$\mathfrak{r} \prec \mathfrak{r}' \text{ iff there is a non-trivial region } \mathfrak{r}'' \text{ such that } \mathfrak{r} \oplus \mathfrak{r}'' = \mathfrak{r}' \text{ [15].}$$

Formally, we have the following definition of a *minimal region*:

Definition 10. *A region* $\mathfrak{r} \in \mathfrak{R}_{\mathfrak{ts}}$ *is* minimal *iff* $\forall\, \widehat{\mathfrak{r}} \in \mathfrak{R}_{\mathfrak{ts}} : \widehat{\mathfrak{r}} \not\prec \mathfrak{r}$. $\diamondsuit$

The set of minimal regions of $\mathfrak{ts}$ w.r.t. $\prec$ will be denoted by $\mathfrak{R}^{min}_{\mathfrak{ts}}$.

We observe that if a non-trivial region is *non-minimal* then it can be represented as a composition of two other non-trivial regions. This follows from the definition of the relation $\prec$ and the fact that the composition operator $\oplus$ is commutative, which, in turn, follows immediately from the definition of $\oplus$.

As an example, consider the ENLST-system in Figure 2(a). Its non-trivial regions are:

$$\begin{array}{lcllcl}
\mathfrak{r}_1 & = & (\varnothing, \{q_0\}, \{e\}) & \mathfrak{r}_3 & = & \bar{\mathfrak{r}}_1 = (\{e\}, \{q_1, q_2\}, \varnothing) \\
\mathfrak{r}_2 & = & (\varnothing, \{q_0\}, \{e_1, e_2\}) & \mathfrak{r}_4 & = & \bar{\mathfrak{r}}_2 = (\{e_1, e_2\}, \{q_1, q_2\}, \varnothing) \\
\mathfrak{r}_5 & = & (\{e_1\}, \{q_1\}, \varnothing) & \mathfrak{r}_7 & = & \bar{\mathfrak{r}}_5 = (\varnothing, \{q_0, q_2\}, \{e_1\}) \\
\mathfrak{r}_6 & = & (\{e_2\}, \{q_2\}, \varnothing) & \mathfrak{r}_8 & = & \bar{\mathfrak{r}}_6 = (\varnothing, \{q_0, q_1\}, \{e_2\})
\end{array}$$

The minimal regions of the ENLST-system in Figure 2(a) are: $\mathfrak{r}_1$, $\mathfrak{r}_2$, $\mathfrak{r}_3$, $\mathfrak{r}_5$ and $\mathfrak{r}_6$. The remaining regions are non-minimal (their set is denoted by $\mathfrak{R}^{\oplus}_{\mathfrak{ts}}$): $\mathfrak{r}_4 = \mathfrak{r}_5 \oplus \mathfrak{r}_6$ $(H = \varnothing)$; $\mathfrak{r}_7 = \mathfrak{r}_2 \oplus \mathfrak{r}_6$ $(H = \{e_2\})$; $\mathfrak{r}_8 = \mathfrak{r}_2 \oplus \mathfrak{r}_5$ $(H = \{e_1\})$.

The reduced ENL-system solution for the ENLST-system in Figure 2(a) that uses only regions minimal $w.r.t.$ $\prec$ is shown in Figure 2(c).

Note that the operator $\oplus$ is not associative as can be shown by using, again, the example of the ENLST-system in Figure 2(a). We can observe that: $(\mathfrak{r}_5 \oplus \mathfrak{r}_6) \oplus \mathfrak{r}_1 \neq \mathfrak{r}_5 \oplus (\mathfrak{r}_6 \oplus \mathfrak{r}_1)$. While $\mathfrak{r}_5$ and $\mathfrak{r}_6$ are compatible and their composition produces $\mathfrak{r}_4$ ($\mathfrak{r}_4 = \mathfrak{r}_5 \oplus \mathfrak{r}_6$), regions $\mathfrak{r}_6$ and $\mathfrak{r}_1$ are not compatible, because $\mathfrak{r}_1$ is not compatible with $\mathfrak{r}_6$, and they cannot be composed.

Also notice that there might be two companion regions (regions based on the same set of states) such that one of them is a minimal region and the second one is a non-minimal region. See, for example, regions $\mathfrak{r}_3 = (\{e\}, \{q_1, q_2\}, \varnothing)$ and $\mathfrak{r}_4 = (\{e_1, e_2\}, \{q_1, q_2\}, \varnothing)$ of the ENLST-system in Figure 2(a), where $\mathfrak{r}_3$ is minimal and $\mathfrak{r}_4$ is non-minimal. So, the minimality of a region cannot be decided by looking at its set of states only.

The following result, about the representation of non-trivial regions, is similar to the results proved for other classes of nets (and their transition systems) that can be found in the literature: Elementary Net Systems [6], pure and bounded Place/Transition Nets [7], Safe Nets [23] or Elementary Net Systems with Inhibitor Arcs (ENI-systems) [27].

Theorem 2. *Every $\mathfrak{r} = (in, r, out) \in \mathfrak{R}_{\mathfrak{ts}}$ can be represented as a composition of minimal regions, where for each pair of different minimal regions in this representation, $\widehat{\mathfrak{r}} = (\widehat{in}, \widehat{r}, \widehat{out})$ and $\widetilde{\mathfrak{r}} = (\widetilde{in}, \widetilde{r}, \widetilde{out})$, $\widehat{r} \cap \widetilde{r} = \varnothing$.*

Theorem 2 and Reduction Rule 2 imply that one can construct a solution to the synthesis problem based on minimal regions $w.r.t.$ the strict pre-order $\prec$. The consequence of the fact that the operator $\oplus$ is not associative, is that we cannot drop the brackets, when we represent a non-trivial region of an ENLST-system as a composition of its minimal regions (for example, $\mathfrak{r} = \mathfrak{r}_1 \oplus (\mathfrak{r}_2 \oplus \ldots (\mathfrak{r}_{n-2} \oplus (\mathfrak{r}_{n-1} \oplus \mathfrak{r}_n)) \ldots)$, where $\mathfrak{r}_i$ $(i = 1, \ldots, n)$ are minimal regions in this representation of $\mathfrak{r}$).

5. Properties of regions

In this section we gather facts regarding relationships of complementary, compatible, companion and minimal regions of an ENLST-system $\mathfrak{ts} = (Q, A, q_0)$.

Fact 1. *Any pair of complementary regions of $\mathfrak{ts}$, $\mathfrak{r}$ and $\bar{\mathfrak{r}}$, form a pair of compatible regions and $\mathfrak{r} \oplus \bar{\mathfrak{r}} = (\varnothing, Q, \varnothing)$.* ♢

From Fact 1 it follows that if $\mathfrak{ts}$ has only minimal regions among non-trivial regions, then only the pairs of complementary regions can be composed resulting in a trivial region.

Fact 2. *Let $\mathfrak{r}_1 = (in_1, r_1, out_1)$ and $\mathfrak{r}_2 = (in_2, r_2, out_2)$ be two non-trivial compatible regions of an ENLST-system $\mathfrak{ts}$. Then $in_1 \cap in_2 = \varnothing$ and $out_1 \cap out_2 = \varnothing$.*

Now, we introduce a notion of *strong compatibility* of regions.

Definition 11. *A region (in, r, out) is strongly compatible with another region (in', r', out') iff the following three conditions hold:*

1. $r \cap r' = \varnothing$.

2. *For every* $e \in out$ *exactly one of the following holds:*
   - $e \in in'$.
   - *For all the transitions* (q, U, q') *such that* $e \in U$ *we have* $q' \notin r'$.
3. *For every* $e \in in$ *exactly one of the following holds:*
   - $e \in out'$.
   - *For all the transitions* (q, U, q') *such that* $e \in U$ *we have* $q \notin r'$.

If region $\mathfrak{r}$ *is strongly compatible with region* $\mathfrak{r}'$ *and region* $\mathfrak{r}'$ *is strongly compatible with* $\mathfrak{r}$ *we say that the two regions are strongly compatible.* ♢

Fact 3. *Two regions, which are strongly compatible are compatible.*

The composition operator defined for the strongly compatible regions (rather than compatible regions) will be denoted by $\oplus_s$. The strict pre-order relation for the set of regions that utilises operator $\oplus_s$ instead of $\oplus$ will be denoted by $\prec_s$. The set of minimal regions of $\mathfrak{ts}$ w.r.t. $\prec_s$ will be denoted by $\mathfrak{R}^{min,s}_{\mathfrak{ts}}$.

The implication of Fact 3 is that Reduction Rule 2 works with strongly compatible regions (we can replace operator $\oplus$ by $\oplus_s$ in that rule). Also, we can strengthen Fact 1 to:

Fact 4. *Any pair of complementary regions of* $\mathfrak{ts}$, $\mathfrak{r}$ *and* $\bar{\mathfrak{r}}$, *form a pair of strongly compatible regions and* $\mathfrak{r} \oplus_s \bar{\mathfrak{r}} = (\varnothing, Q, \varnothing)$. ♢

We will give examples of compatible and strongly compatible pairs of regions using the ENLST-system in Figure 2(a). The pairs of regions $\{\mathfrak{r}_1, \mathfrak{r}_4\}$ and $\{\mathfrak{r}_2, \mathfrak{r}_3\}$ are compatible, but not strongly compatible. However, the pairs $\{\mathfrak{r}_1, \mathfrak{r}_3\}$, $\{\mathfrak{r}_2, \mathfrak{r}_4\}$, $\{\mathfrak{r}_2, \mathfrak{r}_5\}$, $\{\mathfrak{r}_2, \mathfrak{r}_6\}$, $\{\mathfrak{r}_5, \mathfrak{r}_6\}$, $\{\mathfrak{r}_5, \mathfrak{r}_7\}$ and $\{\mathfrak{r}_6, \mathfrak{r}_8\}$ are strongly compatible pairs of regions. Observe also that the pairs $\{\mathfrak{r}_1, \mathfrak{r}_5\}$ and $\{\mathfrak{r}_1, \mathfrak{r}_6\}$ are <u>not</u> compatible, because $\mathfrak{r}_1$ is not compatible with neither $\mathfrak{r}_5$ nor $\mathfrak{r}_6$.

The next proposition shows that unlike $\oplus$ operator, $\oplus_s$ is associative.

Proposition 1. *The operator* $\oplus_s$ *is associative.*

Proposition 2. *Let* $\mathfrak{r}_1 = (in_1, r_1, out_1)$ *and* $\mathfrak{r}_2 = (in_2, r_2, out_2)$ *be compatible regions of an* ST*-system* $\mathfrak{ts} = (Q, A, q_0)$, *which do not satisfy the conditions to be strongly compatible regions of* $\mathfrak{ts}$. *Then there exists a companion region of* $\mathfrak{r}_1$, $\mathfrak{r}'_1 \in \mathfrak{R}^{r_1}_{\mathfrak{ts}}$, *and a companion region of* $\mathfrak{r}_2$, $\mathfrak{r}'_2 \in \mathfrak{R}^{r_2}_{\mathfrak{ts}}$, *such that* $\mathfrak{r}_1$ *and* $\mathfrak{r}'_2$ *are strongly compatible and* $\mathfrak{r}'_1$ *and* $\mathfrak{r}_2$ *are strongly compatible. Furthermore,* $\mathfrak{r}_1 \oplus_s \mathfrak{r}'_2 = \mathfrak{r}'_1 \oplus_s \mathfrak{r}_2 = \mathfrak{r}_1 \oplus \mathfrak{r}_2$.

To illustrate the result of Proposition 2, we can use again the ENLST-system in Figure 2(a). A pair of its regions, $\mathfrak{r}_4$ and $\mathfrak{r}_1$, are compatible, but not strongly compatible regions. However, there are regions $\mathfrak{r}_3 \in \mathfrak{R}^{r_4}_{\mathfrak{ts}}$ and $\mathfrak{r}_2 \in \mathfrak{R}^{r_1}_{\mathfrak{ts}}$, such that $\mathfrak{r}_4$ and $\mathfrak{r}_2$ are strongly compatible and $\mathfrak{r}_3$ and $\mathfrak{r}_1$ are strongly compatible. Also, we have $\mathfrak{r}_4 \oplus_s \mathfrak{r}_2 = \mathfrak{r}_3 \oplus_s \mathfrak{r}_1 = \mathfrak{r}_4 \oplus \mathfrak{r}_1 = (\varnothing, Q, \varnothing)$.

Corollary 1. *Let* $\mathfrak{ts} = (Q, A, q_0)$ *be an* ENLST*-system. Then*

1. $\mathfrak{R}^{min}_{\mathfrak{ts}} = \mathfrak{R}^{min,s}_{\mathfrak{ts}}$.

2. *Every* $\mathfrak{r} \in \mathfrak{R}_{\mathfrak{ts}}$ *can be represented as a composition of minimal regions, where each pair of different minimal regions in this representation,* $\mathfrak{r} = (in, r, out)$ *and* $\mathfrak{r}' = (in', r', out')$*, is a pair of strongly compatible regions.*

The next result, about special families of non-trivial regions of $\mathfrak{ts}$, is inspired by a result proved for the class of Elementary Net Systems in [6]. We have adapted this result here to the context of ENL-systems by changing one of the original conditions that a family of regions should satisfy, but the implied result is the same: a family of regions that satisfy the conditions of Theorem 3, treated as a set of conditions of the synthesised net, would generate a state machine component of this net. Points 2 and 3 of the consequent of Theorem 3 guarantee the satisfaction of Definition 3(1) and the point 1 of the consequent of Theorem 3 guarantees the satisfaction of Definition 3(2).

Theorem 3. *Let* $\mathfrak{R} = \{\mathfrak{r}_1, \mathfrak{r}_2, \ldots, \mathfrak{r}_n\}$ *be a family of non-trivial regions of* $\mathfrak{ts} = (Q, A, q_0)$*, where* $\mathfrak{r}_i = (in_i, r_i, out_i)$*,* $i \in \{1, \ldots, n\}$*, satisfy the following:*

1. *Every two different regions* $\mathfrak{r}_i, \mathfrak{r}_j \in \mathfrak{R}$ *are strongly compatible regions.*
2. $\forall\, \widehat{\mathfrak{r}} = (\widehat{in}, \widehat{r}, \widehat{out}) \in \mathfrak{R}_{\mathfrak{ts}} : \widehat{\mathfrak{r}} \notin \mathfrak{R} \implies (\exists \mathfrak{r}_i \in \mathfrak{R} : \widehat{r} \cap r_i \neq \varnothing)$.

Then:

1. $\bigcup r_i = Q$*;*
2. $\forall e \in E : |{}^\circ e \cap \mathfrak{R}| \leq 1$ *and* $|e^\circ \cap \mathfrak{R}| \leq 1$*;*
3. $\forall e \in E : e \in {}^\circ \mathfrak{r}_i \iff \exists j : e \in \mathfrak{r}_j{}^\circ$.

The saturated ENL-system that is a solution to Problem 1 for a given ENLST-system $\mathfrak{ts}$, $\mathfrak{enl} = \mathfrak{enl}^{\widehat{\ }}_{\mathfrak{ts}}$, and is based on all non-trivial regions, is state machine decomposable (see Definition 5), as due to Fact 4 every pair of complementary regions satisfies the conditions of Theorem 3 and would form a state machine component of $\mathfrak{enl}$. Furthermore, from Corollary 1 it follows that, similarly as for the class of Elementary Net Systems (see [6]), the ENL-system obtained from $\mathfrak{enl}$ by deleting all non-minimal regions following Reduction Rule 2 is also state machine decomposable as every region can be represented as a composition of minimal regions (w.r.t. $\prec_s$) and selected subsets of $\mathfrak{R}^{min}_{\mathfrak{ts}} = \mathfrak{R}^{min,s}_{\mathfrak{ts}}$ would satisfy the conditions of Theorem 3.

As an example we can take the saturated ENL-system synthesised from the ENLST-system in Figure 2(a), shown in Figure 2(b), and its minimised version shown in Figure 2(c). The state machine components of the former ENL-system are generated by the following subsets of conditions/regions:

$$\mathfrak{r}_1 \oplus_s \mathfrak{r}_3 = \mathfrak{r}_2 \oplus_s \mathfrak{r}_5 \oplus_s \mathfrak{r}_6 = \mathfrak{r}_2 \oplus_s \mathfrak{r}_4 = \mathfrak{r}_7 \oplus_s \mathfrak{r}_5 = \mathfrak{r}_8 \oplus_s \mathfrak{r}_6 = (\varnothing, Q, \varnothing).$$

The minimised ENL-system in Figure 2(c) has the first two state machine components from the components listed above.

6. A strategy to eliminate redundant regions

The three reduction rules give conditions for deleting one of the redundant regions at a time. Therefore, we need a *strategy* to delete as many redundant regions as possible to obtain a net, where all (or almost all) remaining regions are needed (essential). The regions are redundant or essential only in the context of other regions. Different strategies lead to different sets of essential (or nearly essential) regions. Such sets of regions were called in [28] *admissible*.

As an example we can take again the ENLST-system in Figure 2(a). We observe that region $\mathfrak{r}_4 = \bar{\mathfrak{r}}_2$ can be deleted according to Reduction Rule 1 (as the complement of region $\mathfrak{r}_2$) or according to Reduction Rule 2 (as a non-minimal region: $\mathfrak{r}_4 = \mathfrak{r}_5 \oplus \mathfrak{r}_6$).

When looking for a strategy for deleting redundant regions, we will take into consideration the following criteria:

- Limiting as much as possible the non-determinism in the process of computing admissible regions.
- Effectiveness of the strategy gauged in terms of the number of the removed regions.
- Efficiency of the strategy gauged in terms of time needed to compute a set of admissible regions.

Our first attempt at formulating a strategy will be based on the first criterion listed above. Reduction Rule 2 showed that all non-minimal regions are redundant [2], so we can eliminate first the non-minimal regions. After this step, for a given ENLST-system $\mathfrak{ts}$, we obtain from the unique set of regions, $\mathfrak{R}_{\mathfrak{ts}}$, the unique set of minimal regions: $\mathfrak{R}^{min}_{\mathfrak{ts}}$. The application of Reduction Rule 1 and Reduction Rule 3 might not lead to a unique resultant set of regions. We might decide to keep certain companion regions and delete other companion regions in case of Reduction Rule 3. Similarly, we can keep both or one (random one) out of two complementary regions. As, in general, the Reduction Rule 3, leads to fewer possible choices of regions to delete, and might be even irrelevant in the case of thin step transition systems, where there are no companion regions, we might decide that this rule should be applied before the Reduction Rule 1, which can lead to many possible combinations of regions to keep/delete. This strategy, called *Strategy (2,3,1)*, can be defined as follows:

1. Use Reduction Rule 2 to delete all non-minimal regions.
2. Use Reduction Rule 3 to delete any redundant companion regions that might be present among the minimal regions.
3. Use Reduction Rule 1 to delete any redundant complementary regions that might be present after the first two steps of the strategy.

To check how good this strategy is from the second criterion point of view we consider a set of ENLST-systems generated by nets composed of several sequential subsystems, where all the events are co-located. Such systems have a lot of companion regions. We will call them $\mathfrak{ts}^{co-loc}_{i,j}$, where the index i denotes the number of sequential subsystems, and the index j denotes the

[2] Reduction Rule 2 uses operator $\oplus$ and it was proved in [15] for this operator, but from Corollary 1(1) we have $\mathfrak{R}^{min}_{\mathfrak{ts}} = \mathfrak{R}^{min,s}_{\mathfrak{ts}}$, so it does not matter whether we use $\prec$ and $\oplus$, or $\prec_s$ and $\oplus_s$, to define the set of minimal regions.

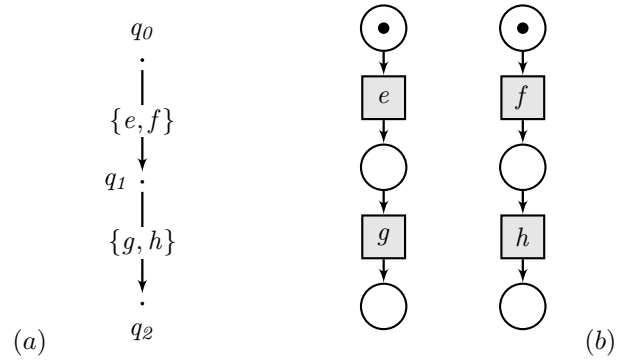

Figure 3: An ENLST-system $\mathfrak{ts}_{2,2}^{co-loc}$ with co-located events e, f, g and h (a), and one of the possible ENL-systems generating it (b).

Table 1
Comparison between the effectiveness of Strategy (3,2,1) and Strategy (2,3,1), where x - y - z in the last two columns reports the number of remaining regions after the first (x), the second (y) and the third (z) stage of the strategies.

$\mathfrak{ts}$	$\lvert\mathbf{Q}\rvert$	$\lvert\mathbf{E}\rvert$	$\lvert\mathfrak{R}_{\mathfrak{ts}}\rvert$	Strategy (3,2,1)	Strategy (2,3,1)
$\mathfrak{ts}_{2,2}^{co-loc}$	3	4	16	12 - 6 - 6	8 - 6 - 6
$\mathfrak{ts}_{2,3}^{co-loc}$	4	6	52	28 - 10 - 10	12 - 8 - 8
$\mathfrak{ts}_{2,4}^{co-loc}$	5	8	160	60 - 12 - 12	16 - 10 - 10
$\mathfrak{ts}_{2,5}^{co-loc}$	6	10	484	124 - 26 - 26	20 - 12 - 12
$\mathfrak{ts}_{3,2}^{co-loc}$	3	6	30	22 - 11 - 11	15 - 11 - 11
$\mathfrak{ts}_{3,3}^{co-loc}$	4	9	126	66 - 20 - 20	24 - 16 - 16
$\mathfrak{ts}_{3,4}^{co-loc}$	5	12	510	190 - 52 - 51	33 - 21 - 21
$\mathfrak{ts}_{3,5}^{co-loc}$	6	15	2046	546 - 147 - 143	42 - 26 - 26
$\mathfrak{ts}_{4,2}^{co-loc}$	3	8	48	36 - 18 - 18	24 - 18 - 18
$\mathfrak{ts}_{4,3}^{co-loc}$	4	12	248	140 - 48 - 47	40 - 28 - 28
$\mathfrak{ts}_{4,4}^{co-loc}$	5	16	1248	540 - 165 - 159	56 - 38 - 38
$\mathfrak{ts}_{4,5}^{co-loc}$	6	20	6248	2108 - 532 - 508	72 - 48 - 48

number of events in each of the line-like sequential subsystem. As an example of such a step transition system we can see an ENLST-system $\mathfrak{ts}_{2,2}^{co-loc}$ in Figure 3(a).

Using the set of step transition systems $\mathfrak{ts}_{i,j}^{co-loc}$ $(i = 2, \ldots, 4;\ j = 2, \ldots, 5)$, we compare the effectiveness of region removal of Strategy (2,3,1) and Strategy (3,2,1) (Strategy (2,3,1) with the first two steps reversed). The result of this comparison is presented in Table 1.

The results in Table 1 are not so surprising. The removal of non-minimal regions makes only sense in the context of all non-trivial regions (as the first step of the strategy). When removing companion regions, the algorithm processes groups of companion regions (each based on a shared set of states) separately from each other. From each group some subset of regions may be removed, at random, according to Reduction Rule 3. Once some of the minimal companion

regions are removed (if we remove companion regions first), some of the regions that were previously non-minimal would become minimal as it won't be possible to represent them as compositions of minimal regions using the remaining minimal regions. Therefore, they won't be deleted by the Reduction Rule 2, if it is applied after Reduction Rule 3. The results in Table 1 show how great would be the loss of effectiveness if we used Strategy (3,2,1) for ENLST-systems with a big number of companion regions.

While the Reduction Rule 2, applied first in our strategy, can be considered as a method for eliminating non-minimal regions, non-minimal from the 'state information' point of view, the Reduction Rule 3 can be understood as a method for eliminating regions that are redundant from the 'event information' point of view. However, some subsets of companion regions will remain after the application of Reduction Rule 3, because they are essential as shown below:

Fact 5. *Let $\mathfrak{r} = (in, r, out)$ be a non-trivial region of $\mathfrak{ts} = (Q, A, q_0)$ and let $\mathfrak{R}^r \subseteq \mathfrak{R}^r_{\mathfrak{ts}}$ be a set of its companion regions (including $\mathfrak{r}$) that were left after the application of the Reduction Rule 3 and let $|\mathfrak{R}^r| \geq 2$. Then all the regions of $\mathfrak{R}^r$ do not satisfy the same conditions of the Reduction Rule 3 (all do not satisfy condition (1) or all do not satisfy condition (2) or all do not satisfy both conditions: (1) and (2)).*

Corollary 2. *Let $\mathfrak{R}^r = \{\mathfrak{r}_1, \mathfrak{r}_2, \ldots, \mathfrak{r}_n\} \subseteq \mathfrak{R}^r_{\mathfrak{ts}}$ be a set of companion regions based on r that were left after the application of the Reduction Rule 3. Then, for every region $\mathfrak{r}_i = (in_i, r, out_i)$ of $\mathfrak{R}^r$ that does not satisfy condition (1) (respectively (2)) of Reduction Rule 3 there exists a unique $E_i \subseteq in_i$ (respectively $E'_i \subseteq out_i$) with events that are not present in the in (respectively out) sets of other regions from $\mathfrak{R}^r$. So, companion regions of $\mathfrak{R}^r$ are 'indexed' by the unique subsets of events of their in (respectively out) sets. We will call these subsets of events* in-indices *(respectively* out-indices*) of r for the regions of $\mathfrak{R}^r$.* ♢

Notice that sets $\mathfrak{R}^r$ in Corollary 2 (and in Fact 5) might be equal to $\mathfrak{R}^r_{\mathfrak{ts}}$. For example, for $\mathfrak{ts}$ in Figure 2(a), we have $\mathfrak{R}^{\{q_0\}} = \mathfrak{R}^{\{q_0\}}_{\mathfrak{ts}} = \{\mathfrak{r}_1, \mathfrak{r}_2\}$. Also, the indexing sets of events do not need to be singleton sets (as, for example, set $\{e_1, e_2\}$ for $\{q_0\}$ of $\mathfrak{r}_2$ of $\mathfrak{ts}$ in Figure 2(a)).

We will further illustrate the above results using the ENLST-system in Figure 3(a). Its non-trivial regions are listed below:

$$
\begin{array}{lcllcl}
\mathfrak{r}_1 & = & (\varnothing, \{q_0\}, \{e\}) & \bar{\mathfrak{r}}_1 & = & (\{e\}, \{q_1, q_2\}, \varnothing) \\
\mathfrak{r}_2 & = & (\{e\}, \{q_1\}, \{g\}) & \bar{\mathfrak{r}}_2 & = & (\{g\}, \{q_0, q_2\}, \{e\}) \\
\mathfrak{r}_3 & = & (\{g\}, \{q_2\}, \varnothing) & \bar{\mathfrak{r}}_3 & = & (\varnothing, \{q_0, q_1\}, \{g\}) \\
\mathfrak{r}_4 & = & (\{e\}, \{q_1\}, \{h\}) & \bar{\mathfrak{r}}_4 & = & (\{h\}, \{q_0, q_2\}, \{e\}) \\
\mathfrak{r}_5 & = & (\{h\}, \{q_2\}, \varnothing) & \bar{\mathfrak{r}}_5 & = & (\varnothing, \{q_0, q_1\}, \{h\}) \\
\mathfrak{r}_6 & = & (\varnothing, \{q_0\}, \{f\}) & \bar{\mathfrak{r}}_6 & = & (\{f\}, \{q_1, q_2\}, \varnothing) \\
\mathfrak{r}_7 & = & (\{f\}, \{q_1\}, \{g\}) & \bar{\mathfrak{r}}_7 & = & (\{g\}, \{q_0, q_2\}, \{f\}) \\
\mathfrak{r}_8 & = & (\{f\}, \{q_1\}, \{h\}) & \bar{\mathfrak{r}}_8 & = & (\{h\}, \{q_0, q_2\}, \{f\})
\end{array}
$$

The implemented tool, after applying Reduction Rule 2, will delete 8 out of 16 regions, leaving the minimal regions ($\mathfrak{r}_1$ - $\mathfrak{r}_8$). The set of minimal regions will be the same whether they are defined w.r.t. $\prec$ or $\prec_s$ strict pre-order (see Corollary 1(1)) as every non-minimal region that can be expressed as a composition of compatible regions can be also expressed as a composition of

strongly compatible regions (see Proposition 2). For example, $\bar{\mathfrak{r}}_3 = \mathfrak{r}_1 \oplus_s \mathfrak{r}_2 = \mathfrak{r}_1 \oplus \mathfrak{r}_7$, where the first two regions are strongly compatible, but the second two regions are only compatible, but not strongly compatible. The algorithm that implements Reduction Rule 3, when applied to this example, would delete two out of four regions based on the set of states $\{q_1\}$ leaving either $\mathfrak{r}_2$ and $\mathfrak{r}_8$ or $\mathfrak{r}_4$ and $\mathfrak{r}_7$. The remaining pairs of companion regions, based on sets of states $\{q_0\}$,$\{q_1\}$ and $\{q_2\}$, will remain as they are essential (having different in-indices or/and out-indices for the shared sets of states; see Corollary 2). The Reduction Rule 1, the last to be used in Strategy (2,3,1), is not applicable to this example as all the complementary regions of $\mathfrak{r}_1$ - $\mathfrak{r}_8$ were already deleted as non-minimal regions (see the results for $\mathfrak{ts}_{2,2}^{co-loc}$ in Table 1).

7. Conclusions

In this paper we discussed the minimisation of the synthesised ENL-systems and the strategy to eliminate redundant regions that involves three reduction rules. Also, we investigated the properties of minimal regions that play a crucial role in the minimisation process. We showed that synthesised and minimised nets that are based on all minimal regions (after the application of the Reduction Rule 2) do not lose the property of the saturated ENL-systems of being state machine decomposable. We believe [3] that after the application of the Reduction Rule 3 this property still holds for the resultant net. However, after applying Reduction Rule 1, some synthesised ENL-systems are no longer decomposable. As an example, we can take the ENLST-system generated by the ENL-system in Figure 1. The synthesis procedure for this example will produce a saturated net that has only minimal regions (12 regions). Some of them are redundant and can be deleted according to Reduction Rule 1. Figure 1 shows one of the possible minimised versions of this net that is not state machine decomposable.

In the future work, we plan to investigate the relationship between the split of ENL-systems into state machine components (based on conditions) and the split into localities (based on events). Also, we want to develop an improved algorithm implementing Reduction Rule 1, which would allow to target certain regions for deletion from the pairs of complementary regions.

Acknowledgement

We are grateful to the anonymous reviewers for their constructive comments, which have helped us to improve the presentation of the paper and to clarify our ideas.

The first author is grateful to the National Transitional Council of Libya for funding her PhD studentship and research.

References

[1] S. Dasgupta, D. Potop-Butucaru, B. Caillaud, A. Yakovlev, Moving from weakly endochronous systems to delay-insensitive circuits, Electronic Notes in Theoretical Compututer Science 146 (2006) 81–103.

[3] This still needs to be formally proved.

[2] G. Păun, Membrane Computing: An Introduction, Natural Computing Series, Springer, 2002.

[3] J. Kleijn, M. Koutny, G. Rozenberg, Towards a Petri net semantics for membrane systems, in: R. Freund, G. Paun, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing, 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers, volume 3850 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 292–309.

[4] A. Ehrenfeucht, G. Rozenberg, Partial (set) 2-structures. Part I: Basic notions and the representation problem. Part II: State spaces of concurrent systems, Acta Informatica 27 (1990) 315–368.

[5] É. Badouel, P. Darondeau, Theory of regions, in: W. Reisig, G. Rozenberg (Eds.), Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, volume 1491 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 529–586.

[6] L. Bernardinello, Synthesis of net systems, in: M. A. Marsan (Ed.), 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*, Springer, 1993, pp. 89–105.

[7] L. Bernardinello, G. D. Michelis, K. Petruni, S. Vigna, On the synchronic structure of transition systems, in: J. Desel (Ed.), Proceedings of the International Workshop on Structures in Concurrency Theory, STRICT 1995, Berlin, Germany, May 11-13, 1995, Workshops in Computing, Springer, 1995, pp. 69–84.

[8] M. Mukund, Petri nets and step transition systems, International Journal of Foundations of Computer Science 3 (1992) 443–478.

[9] M. Nielsen, G. Rozenberg, P. S. Thiagarajan, Elementary transition systems, Theoretical Computer Science 96 (1992) 3–33.

[10] M. Pietkiewicz-Koutny, The synthesis problem for elementary net systems with inhibitor arcs, Fundamenta Informaticae 40 (1999) 251–283.

[11] É. Badouel, L. Bernardinello, P. Darondeau, Petri Net Synthesis, Texts in Theoretical Computer Science. An EATCS Series, Springer, 2015.

[12] M. Koutny, M. Pietkiewicz-Koutny, Transition systems of elementary net systems with localities, in: C. Baier, H. Hermanns (Eds.), Concurrency Theory 2006, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006. Proceedings, volume 4137 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 173–187.

[13] M. Koutny, M. Pietkiewicz-Koutny, Synthesis of elementary net systems with context arcs and localities, Fundamenta Informaticae 88 (2008) 307–328.

[14] M. Koutny, M. Pietkiewicz-Koutny, Synthesis of Petri nets with localities, Scientific Annals of Computer Science 19 (2009) 1–23.

[15] M. Koutny, M. Pietkiewicz-Koutny, Minimal regions of ENL-transition systems, Fundamenta Informaticae 101 (2010) 45–58.

[16] A. Ahmed, M. Pietkiewicz-Koutny, Algorithms for the synthesis of elementary net systems with localities, in: M. Köhler-Bußmeier, E. Kindler, H. Rölke (Eds.), Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE 2020, Paris, France, June 24, 2020 (due to COVID-19: virtual workshop), volume 2651 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 86–107.

[17] A. Ahmed, M. Koutny, M. Pietkiewicz-Koutny, Synthesising elementary net systems with

localities, Theoretical Computer Science 908 (2022) 123–140.
[18] W. M. P. van der Aalst, B. F. van Dongen, C. W. Günther, A. Rozinat, E. Verbeek, T. Weijters, Prom: The process mining toolkit, in: A. K. A. de Medeiros, B. Weber (Eds.), Proceedings of the Business Process Management Demonstration Track (BPMDemos 2009), Ulm, Germany, September 8, 2009, volume 489 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2009.
[19] R. Bergenthum, J. Desel, R. Lorenz, S. Mauser, Synthesis of Petri nets from scenarios with viptool, in: K. M. van Hee, R. Valk (Eds.), Applications and Theory of Petri Nets, 29th International Conference, PETRI NETS 2008, Xi'an, China, June 23-27, 2008. Proceedings, volume 5062 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 388–398.
[20] R. Bergenthum, Prime miner - process discovery using prime event structures, in: International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24-26, 2019, IEEE, 2019, pp. 41–48.
[21] J. Carmona, J. Cortadella, M. Kishinevsky, Genet: A tool for the synthesis and mining of Petri nets, in: Application of Concurrency to System Design 2009, 9th International Conference, ACSD 2009, Augsburg, Germany, 1-3 July 2009, IEEE Computer Society, 2009, pp. 181–185.
[22] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers, IEICE Transactions on Information and Systems E80-D (1997) 315–325.
[23] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, Logic Synthesis of Asynchronous Controllers and Interfaces, Springer Berlin Heidelberg, 2002.
[24] L. L. Mannel, R. Bergenthum, W. M. P. van der Aalst, Removing implicit places using regions for process discovery, in: W. M. P. van der Aalst, R. Bergenthum, J. Carmona (Eds.), Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2020, ATAED 2020, Paris, France, June 24, 2020 (due to COVID-19: virtual workshop), volume 2625 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 20–32.
[25] I. Poliakov, V. Khomenko, A. Yakovlev, Workcraft - A framework for interpreted graph models, in: G. Franceschinis, K. Wolf (Eds.), Applications and Theory of Petri Nets, 30th International Conference, PETRI NETS 2009, Paris, France, June 22-26, 2009. Proceedings, volume 5606 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 333–342.
[26] Workcraft, https://workcraft.org/, 2018.
[27] M. Pietkiewicz-Koutny, Synthesis of ENI-systems using minimal regions, in: D. Sangiorgi, R. de Simone (Eds.), Concurrency Theory 1998, 9th International Conference, CONCUR 1998: Nice, France, September 8-11, 1998. Proceedings, volume 1466 of *Lecture Notes in Computer Science*, Springer, 1998, pp. 565–580.
[28] J. Desel, W. Reisig, The synthesis problem of Petri nets, Acta Informatica 33 (1996) 297–315.

A First Glimpse at Petri Net Regions

Robin Bergenthum[1] and Jakub Kovář[2]

[1] *Fakultät für Mathematik und Informatik, FernUni in Hagen, Germany*
[2] *Lehrgebiet Programmiersysteme, FernUni in Hagen, Germany*

Abstract

Synthesis is automatically producing a process model from specified behavior. If the desired process model is a Petri net, synthesis is tackled by so-called region theory. Region-based synthesis has been extensively explored for cases where the specification language is a transition system, a language or even a partially ordered language. Although the ideas of region-based synthesis are the same for every kind of specification, every type of specification has its own region definition and uses different representations of the set of all regions to synthesize a finite result. Up to this point, state-based and language-based regions are just two different concepts.

In this paper, we introduce a new region definition we call Petri net regions and reason that every state-based and every language-based region is a Petri net region as well. Thus, there is no need to distinguish language-based and state-based regions anymore. With the help of Petri net regions every concept from one of the older region definitions can be directly applied to the other. Using Petri net regions, we introduce an implementation of a synthesis algorithm that handles state-based as well as language-based input. Furthermore, this algorithm can synthesize a Petri net from a set of labeled Petri nets.

Keywords

Petri Nets, Synthesis, Region Theory, State-Based Regions, Language-Based Regions

1. Introduction

Complex systems are often modeled by Petri nets [1, 2, 3, 4]. Petri nets have formal semantics, an intuitive graphical representation, and can express concurrency among the occurrences of actions of a system. However, constructing a Petri net model for a real-world process is a costly and error-prone task [3, 5]. Fortunately, whenever we model a system, there are often some associated descriptions or even specifications of the desired process behavior. There may be log-files of recorded behavior, example runs, and product specifications describing use cases. We can model these specifications by a language, a transition system, or a partial language. If a specification reflects the desired behavior faithfully, we can automatically synthesize the best fitting process model. The synthesis problem is to compute a process model so that: (A) the specification is behavior of the generated model and (B) the generated model has minimal additional behavior.

Looking at the literature, the theory behind Petri net synthesis is called region theory [6, 7]. Region theory has been extensively explored for transition systems, languages, and even partial languages. There are many non-trivial theoretical results, notions, case studies, as well as tool support by tools like for example ProM [8], Genet [9], APT [10], and Viptool [11].

Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2022, Bergen, Norway
EMAIL: robin.bergenthum@fernuni-hagen.de; jakub.kovar@fernuni-hagen.de
ORCID: 0000-0003-0464-8843 (Bergenthum); 0000-0002-7775-3698 (Kovář)


Today, region theory has two main branches. If the input is state-based, a transition system for example, we apply the theory of state-based regions [6, 7,12]. Here, a region is a multi-set of states, and we construct a set of minimal regions to generate a finite set of valid places for the resulting Petri net. If the input is language-based, an event log, a language, or a partial language for example, we apply the theory of language-based regions [13, 14]. Here, a region is a multi-set of tokens produced by prefixes of the language and we calculate valid places by solving a related integer linear programming problem (ILP). To generate a finite result, we calculate a basis of the ILP or we use the concept of wrong continuations [15]. Both branches of region theory follow the same ideas, but in detail use different techniques, definitions, and algorithms. To just give one very illustrative example, we can compare the two prominent process discovery algorithms based on region theory. The ILP-miner [16] is language-based, the region-miner [17] uses state-based techniques.

This workshop paper presents a glimpse at a new and very intuitive definition we call Petri net regions. This definition can handle state-based and language-based input at once. We define Petri net regions for transition systems, languages, partial languages, branching processes, and labeled Petri nets. We get our new definition simply by lifting the notion of compact tokenflow regions [14] from partial languages to labeled Petri nets. We lift the notion of state-based regions [6, 7,12] from transition systems to labeled Petri nets as well. We show that if we lift both notions, they will match perfectly. Furthermore, we will argue that Petri net regions can handle any given labeled Petri net as an input, even if this net is neither a transition system nor a (partial) language.

2. Preliminaries

In the following preliminaries, we present compact token flow regions, state-based regions, and Petri nets by instructive examples and refer the reader to the literature for formal definitions. We assume the reader is familiar with the basic concepts of region theory.

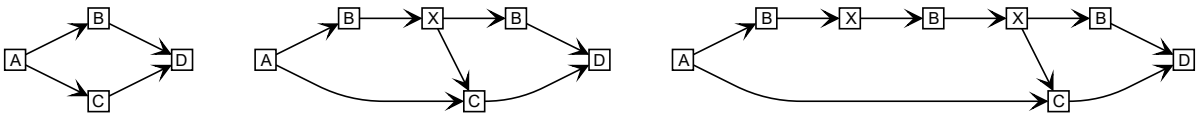

Figure 1: A partial language.

Figure 1 depicts a set of three labeled Hasse diagrams, i.e., a partial language. Every diagram models a run of the intended system's behavior. In every run, every node is called an event and models the occurrence of a transition of the Petri net to be synthesized. The set of arcs defines a later-than relation on the set of events. Thus, a partial language can directly express concurrency.

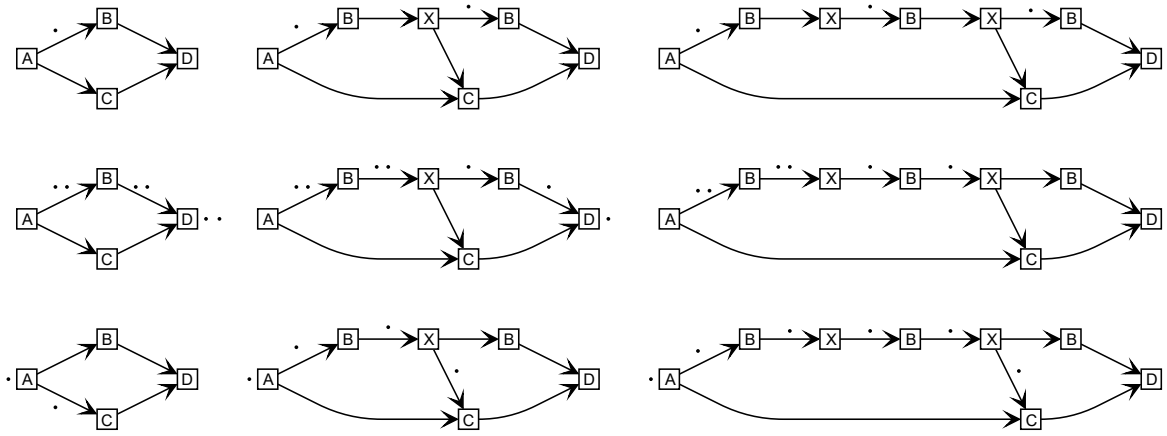

Figure 2: Three compact tokenflow regions.

To synthesize a Petri net model from a partial language we use the concept of compact tokenflow regions. Every compact tokenflow region is a distribution of tokens on the arcs of the specification and defines a valid place for the synthesis result. That means, every region describes the production, passing, and consumption of tokens between events in the related valid place and thus, adding this place to the synthesis result will generate a net which is still able to execute the specified behavior. Figure 2 depicts three copies of the specification of Figure 1 and three different compact tokenflow regions. The first region defines the valid place p2 of Figure 3. The second region defines the valid place p3. The third region defines the valid place p5.

A compact tokenflow based synthesis algorithm defines an ILP, so that every solution of the ILP is a compact tokenflow region. The solution-space of this ILP is not finite, but the set of all places related to the finite set of integer basis solutions will solve the synthesis problem. A second approach to receive a finite result from the ILP, is to use the concept of wrong continuations. Roughly speaking, the set of wrong continuations is the border between the specified and all other behavior. The set of wrong continuations is finite if the specification is finite as well. Figure 3 depicts the Petri net synthesized from the specification depicted in Figure 1 using compact tokenflow regions, an ILP synthesis algorithm, the concept of wrong continuations and deleting some implicit places as a last step.

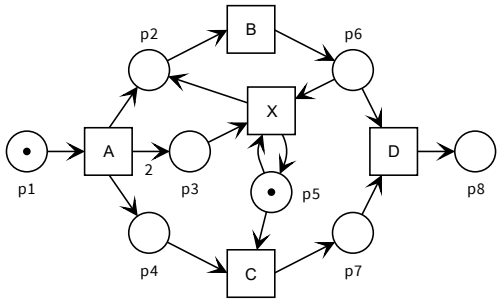

Figure 3: A Petri net.

Figure 4 depicts a transition system. A transition system is a set of states connected by a set of labeled arcs. Every arc models an event changing the state of the desired Petri net model. Partial languages can model concurrency, but they need to add extra runs whenever there is conflict. Transition systems can model conflict but are not able to model concurrency. Thus, depending on the specific application using one specific specification language can be more adequate than using the other.

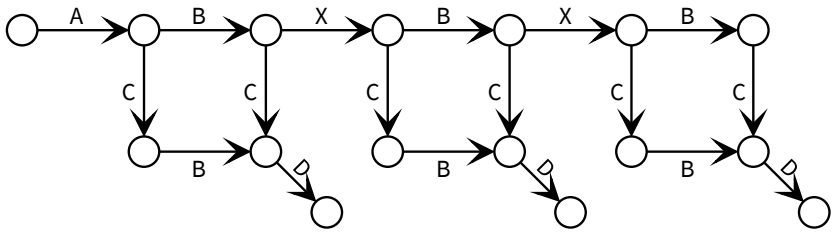

Figure 4: A transition system.

To synthesize a Petri net model from a transition system we use the concept of state-based regions. Such region is a multi-set of states so that every pair of equally labeled events has the same gradient. That means, every label is entering, not-crossing, or exiting the region with the same weight. Thus, every event has a constant effect on the marking of the synthesized result. We add the related places to construct a Petri net so that its reachability graph will be isomorphic to the input if such a net exists.

Figure 5 depicts three copies of the specification of Figure 4 and three different state-based regions. The first region is the set of grey states and defines the valid place p2 of Figure 3. The second region is twice the set of black states plus the grey states and defines the valid place p3. The third region is the

set of grey states and defines the valid place p5 without the short-loop to X. State-based regions usually do not generate short-loops because transition systems can not specify concurrency.

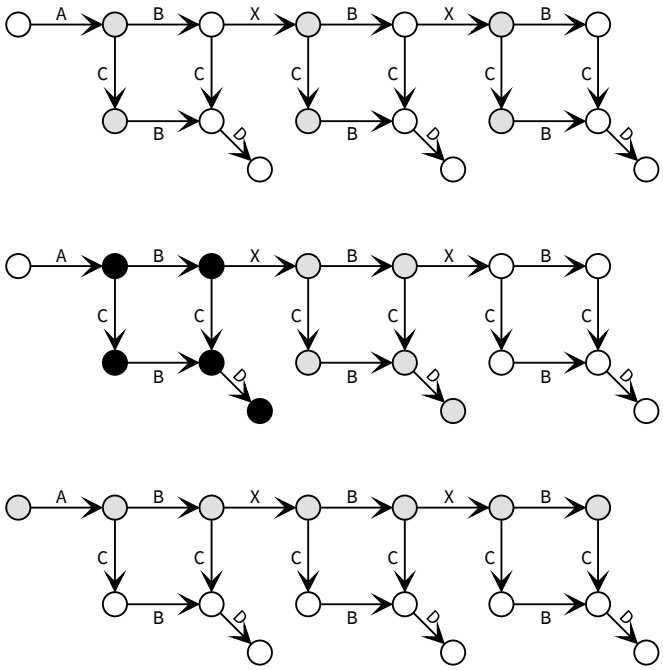

Figure 5: Three state-based regions.

Most state-based region synthesis algorithms construct a set of so-called minimal regions. They start by a set of minimal candidate regions and repair these regions by adding states until labels have a constant gradient to get a finite Petri net result.

3. Petri Net Regions

This section introduces Petri net regions. A Petri net region is a marking of a set of labeled Petri nets so that: (I) For every pair of equally labeled transitions, the difference between the sum of tokens in the post-set and the sum of tokens in the pre-set is the same. (II) For every pair of labeled Petri nets, the sum of tokens of all places with an empty preset is the same. We highlight the idea of this definition, by translating the specification of Figure 1 into three labeled Petri nets. Every event becomes a labeled transition, and we add a place to every relation. Figure 6 depicts three copies of the translated result, and we add markings related to the three different compact tokenflow regions of Figure 2.

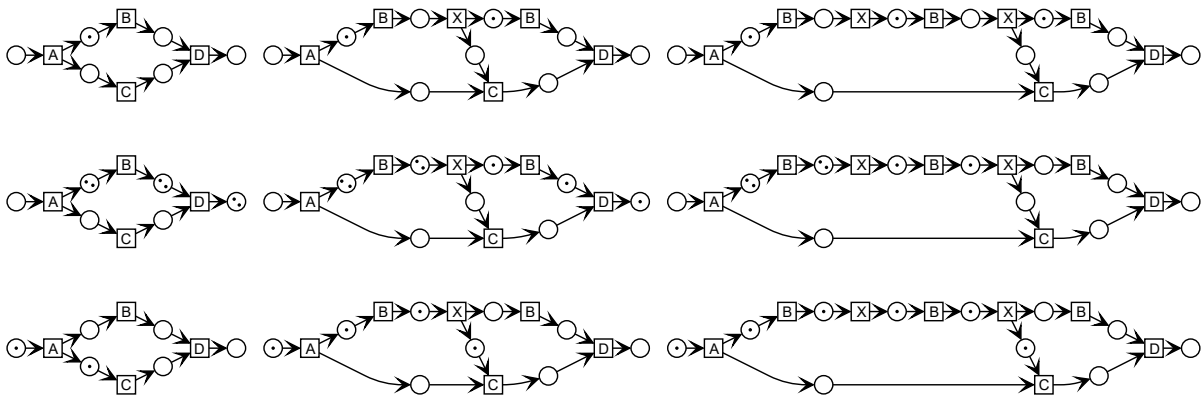

Figure 6: Three Petri net regions, equivalent to Figure 2.

It is easy to see that there is a one-to-one correspondence between a compact tokenflow region on the set of Hasse diagrams and a Petri net region on the labeled Petri nets. Thus, we propose that the following conjecture holds.

Conjecture 1. Let S be a set of labeled Petri nets and S be equivalent to a partial language L. There is a compact tokenflow region for L defining the valid place p iff there is a Petri net region for S defining the valid place p. We can solve the synthesis problem for partial languages using Petri net regions.

We translate the specification of Figure 4 into a labeled Petri net. Every state becomes a place, and every event becomes a labeled transition. Figure 7 depicts three copies of the translated result, and we add markings related to the three different state-based regions of Figure 5.

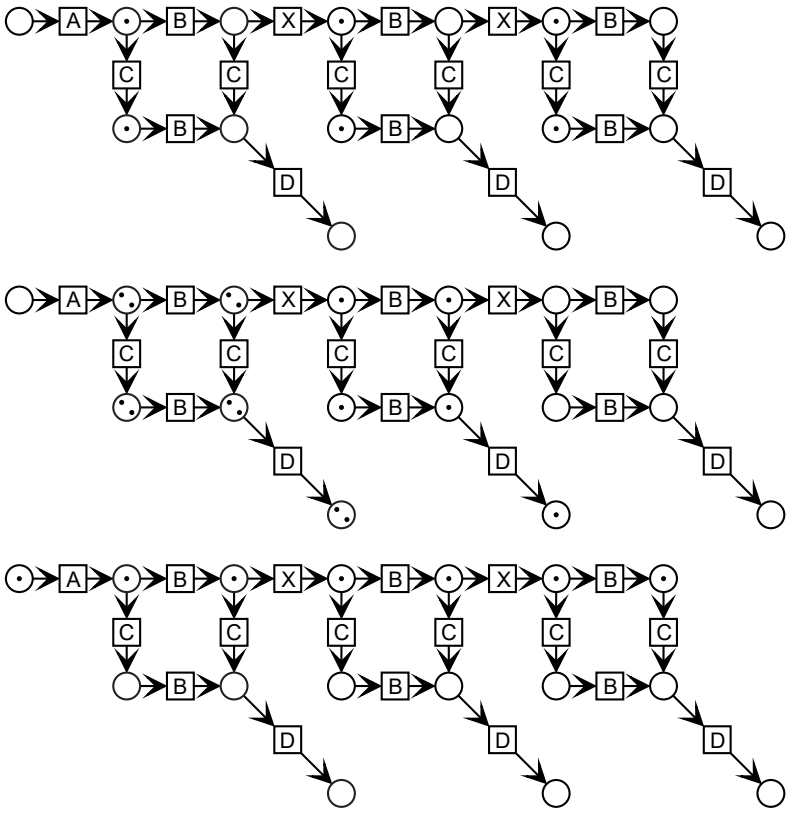

Figure 7: Three Petri net regions, equivalent to Figure 5.

Again, it is easy to see that there is a one-to-one correspondence between a state-based region on a transition system and a Petri net region on the labeled Petri net. Thus, we propose that the following conjecture holds as well.

Conjecture 2. Let S be a labeled Petri net and S be equivalent to a transition system L. There is a state-based region for L defining the valid place p iff there is a Petri net region for S defining the valid place p. We can solve the synthesis problem for transition systems using Petri net regions.

Conditions (I) and (II) are very intuitive and easy to check/implement. To get a running synthesis algorithm we can construct Petri net regions by repairing minimal candidate regions or by solving a related ILP. To get a finite result we can use the set of minimal regions, a basis of the ILP, or use the concept of wrong continuations. Petri net regions can handle state-based and language-based input. There is no need to distinguish the two concepts anymore.

Petri net regions are not only unifying previous region definitions but have very interesting application if the input is neither a transition system nor a partial language. Remark, Petri net regions can handle general labeled Petri nets as an input. Note that, we have to be careful with condition (A) of the synthesis problem in this part of the paper because of a missing semantic defining when a net is in the language of another. But we will get the idea ;)

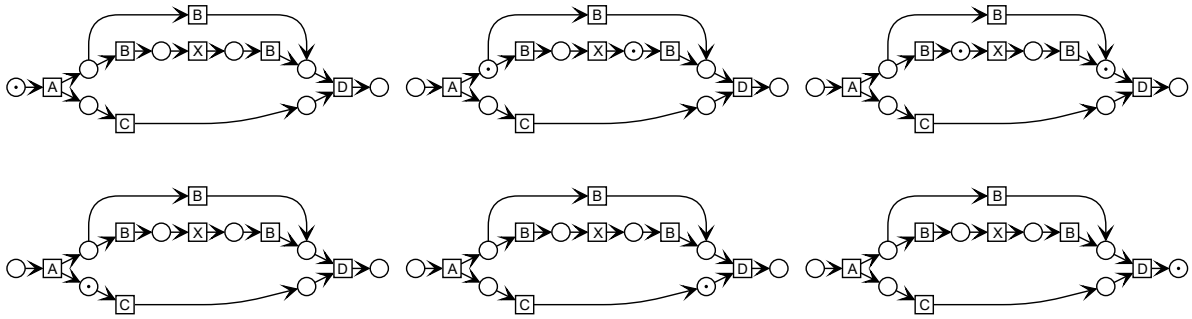

Figure 8: Six minimal Petri net regions.

Figure 8 depicts six copies of a labeled Petri net, and we use this Petri net as the specification. This Petri net is neither a transition system nor a partial language. Every depicted marking is a minimal Petri net region. The related synthesis result is depicted in Figure 9. Specifying a shared place after no and one iteration of the loop started by X, leads to a result where counting the number of loops is not possible. Thus, Figure 9 is the intended result.

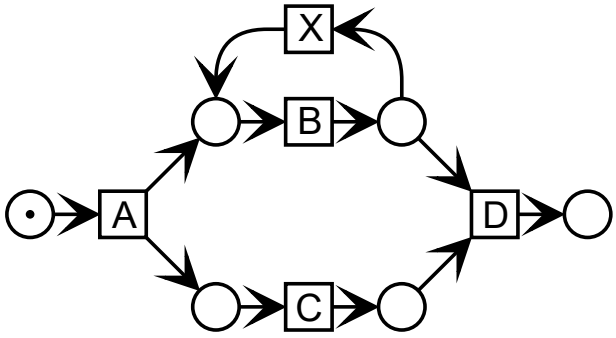

Figure 9: Synthesis result using minimal Petri net regions.

The left part of Figure 10 depicts another labeled Petri net where the state after the occurrence of the loop is not shared. The depicted marking is a minimal Petri net region. Taking the left-hand side as an input, the right-hand side of Figure 10 depicts the synthesis result. This time, the occurrence of X and the occurrence of B is restricted by the additional place.

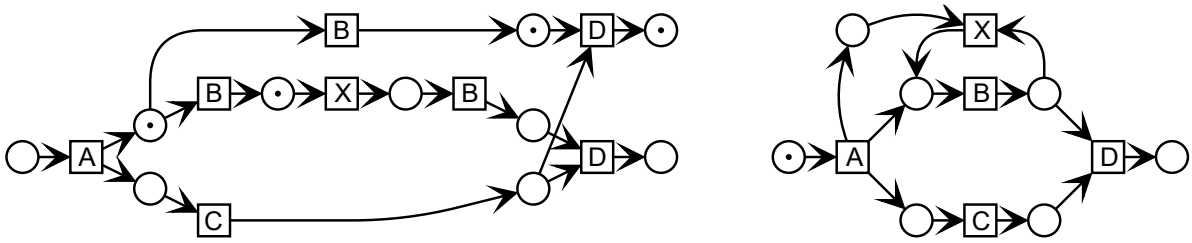

Figure 10: Petri net region for a branching process.

4. Synthesis Algorithm

In this section, we use the new concept of Petri net regions to implement a synthesis algorithm. Using regions, the region definition ensures that the output of every synthesis can execute the input. But if we implement a full-on synthesis algorithm, we must choose (a) the net class of the output, (b) the kind of regions we calculate, and (c) how regions are calculated. For the net class we can calculate nets with or without arc-weights or short-loops. We can calculate regions that are one-bounded and have at most one token in every place. To get a finite result, we can calculate a basis, a set of regions related to wrong continuations, or a set of minimal regions. Furthermore, we can construct regions by simulation, construction, or by implementing an ILP.

In this paper, to highlight the power of Petri net regions, we implement a synthesis algorithm using the new concept. The input to this algorithm is a set of labeled Petri nets. The algorithm uses an ILP to enforce conditions (I) and (II) of the Petri net region definition. Calculating the set of minimal Petri net regions, the algorithm will synthesize a Petri net without short-loops. We can toggle if regions will be one-bound or not. The synthesis algorithm is available at the I ♥ Petri Nets web toolkit (https://www.fernuni-hagen.de/ilovepetrinets/).

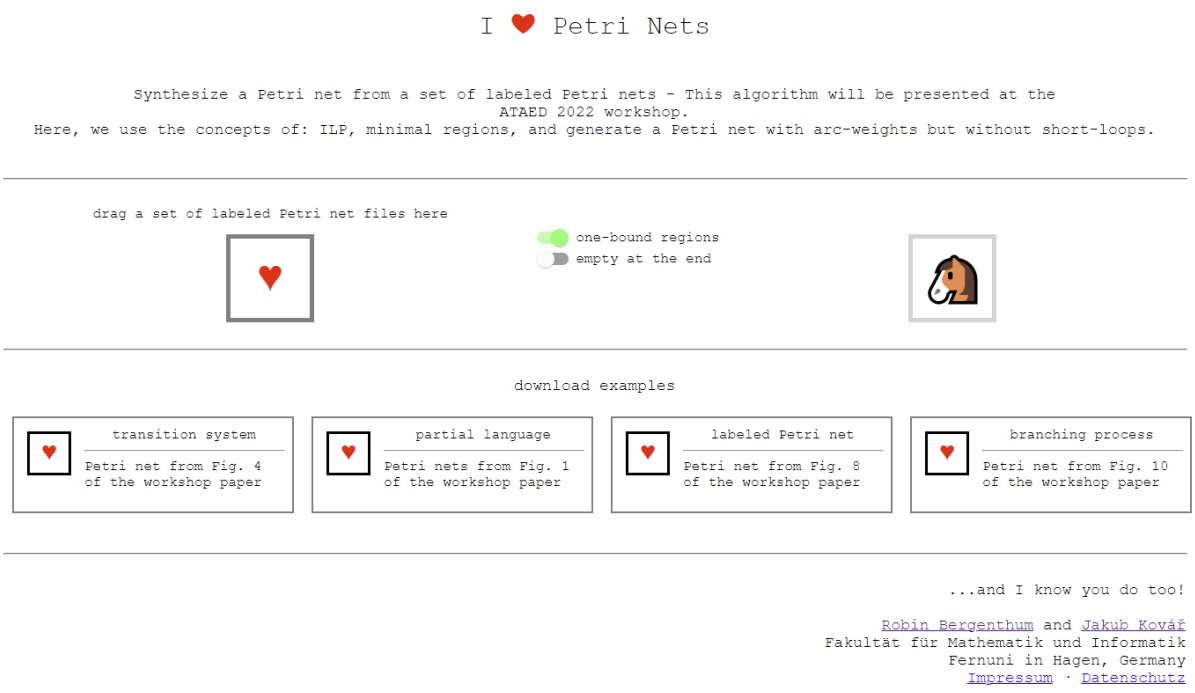

Figure 11: Synthesis using Petri net regions in the I ♥ Petri Nets web toolkit.

Figure 11 depicts a screenshot of the synthesis algorithm. Input to the algorithm is a set of text-files describing labeled Petri nets. We can use any text-file editor to specify the input, use an editor from the I ♥ Petri Nets web toolkit, or simply download example nets from the synthesis web page. Synthesis starts as we drag-and-drop files to the big ♥ button. As soon as the synthesis algorithm is done, we can download the synthesized result via the 🐴 button. We can read the files using a text-file editor or use the "show a labeled Petri net" tool from the toolkit.

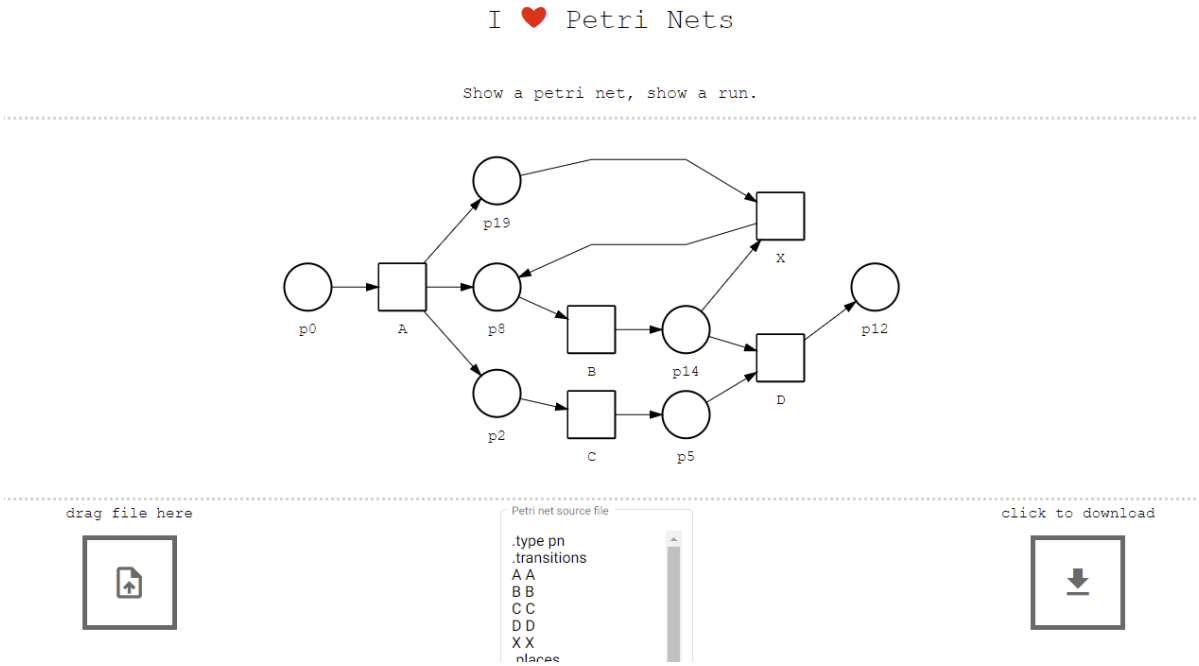

Figure 12: Displaying a labeled Petri net in the I ♥ Petri Nets web toolkit.

Figure 12 depicts the synthesis result using the example net of Figure 10 as an input for the synthesis algorithm calculating one-safe regions displayed by the show a labeled Petri net editor of the web toolkit.

5. Conclusion

Petri net regions unify state-based and language-based region theory. They can handle transition systems, languages, partial languages, branching processes (see Figure 10), and general labeled Petri nets (see Figure 9). We present a synthesis algorithm using Petri net regions in the I ♥ Petri Nets web toolkit. Here, we use an ILP to generate a set of minimal Petri net regions and calculate results without short-loops. Obviously, this workshop paper only presents a first glance at Petri net regions, the formal definitions and proofs must be submitted in future work.

References

[1] Peterson, J. L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall, 1981.
[2] Desel, J.; Juhas, G.: "What is a Petri Net?". Advances in Petri Nets, LNCS 2128, Springer, 2001.
[3] van der Aalst, W. M. P.; van Dongen, B. F.: Discovering Petri Nets from Event Logs. ToPNoC VII, LNCS 7480, Springer, 2013, 372–422.
[4] Reisig, W.: Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies. Springer, 2013.
[5] Mayr, H. C.; Kop, C.; Esberger, D.: Business Process Modeling and Requirements Modeling. ICDS 2007, Computer Society, IEEE, 2007, 8—14.
[6] Ehrenfeucht, A.; Rozenberg, G.: Partial (Set) 2-Structures. Part I: Basic Notions and the Representation Problem, Part II: State Spaces of Concurrent Systems. Acta Inf. 27(4), 1990.
[7] Badouel, E.; Bernardinello, L.; Darondeau, P.: Petri Net Synthesis. Texts in Theoretical Computer Science, Springer, 2015.

[8] van Dongen, B. F.; de Medeiros, A.; Verbeek, H. M. W.; Weijters, A. J. M. M.; van der Aalst, W. M. P.: The ProM Framework: A New Era in Process Mining Tool Support. Petri Nets 2005, LNCS 3536, Springer, 2005, 381–390.
[9] Carmona, J.; Cortadella, J.; Kishinevsky, M.: Genet: a Tool for the Synthesis and Mining of Petri Nets. Application of Concurrency to System Design 2009, 2009, 181–185.
[10] Borde, D., Dierkes, S., Ferrari, R., Gieseking, M., Göbel, V., Grunwald, R., von der Linde, B., Lückehe, D., Schlachter, U., Schierholz, C., Schwammberger, M., Spreckels, V.: APT: analysis of Petri nets and labeled transition systems. https://github.com/CvO-Theory/apt.
[11] Bergenthum, R.; Desel, J.; Juhas, G.; Lorenz, R.: Synthesis of Petri Nets from Scenarios with VipTool. Petri Nets 2008, LNCS 5062, Springer, 2008, 388–398.
[12] Bernardinello, L.; De Michelis. G.; Petruni, K.; Vigna, S.: On The Synchronic Structure of Transition Systems. Structures in Concurrency Theory, Springer, 1995, 69–84.
[13] Bergenthum, R.; Desel, J.; Lorenz, R.; Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. Fundamenta Informaticae 88, IOS Press, 2008, 437–468.
[14] Bergenthum, R.: Synthesizing Petri Nets from Hasse Diagrams. Proceedings of Business Process Management 2017, LNCS 10445, Springer, 2017, 22-39.
[15] Bergenthum, R.; Desel, J.; Mauser, S.: Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language. ToPNoC III, LNCS 5800, Springer, 2009, 216–243.
[16] van der Werf, J.M.E.M.; van Dongen, B.F.; Hurkens, C.A.J.; Serebrenik, A.: Process Discovery Using Integer Linear Programming. Proceedings of PETRI NETS 2008, LNCS 5062, Springer, 2008.
[17] Carmona, J.; Cortadella, J.; Kishinevsky, M.: A Region-Based Algorithm for Discovering Petri Nets from Event Logs. Proceedings of BPM 2008. LNCS 5240, Springer, 2008.

Implementable strategies for a two-player asynchronous game on Petri nets

Federica Adobbati[1], Luca Bernardinello[1], Lucia Pomello[1] and Riccardo Stramare[1]

[1]*Università degli studi di Milano–Bicocca, DISCo*

Abstract

We consider a two-player game on Petri nets, in which each player controls a subset of transitions. The players are called 'user' and 'environment'; we assume that the environment must guarantee progress on its transitions. A play of this game is a run in the unfolding of the net, satisfying the progress assumption. In general, we define a strategy for the user as a map from the set of 'observations' to subsets of transitions owned by the user. Different restrictions on strategies can be used to encode observability assumptions. We say that a given strategy is implementable if the net can be endowed with new places so that the runs of the new net coincide with the plays of the original net, complying with the strategy. We propose an algorithm based on the search of regions to decide whether a strategy is implementable.

1. The game

We recall the definition of a two-player, asynchronous game on 1-safe Petri nets (see [1]), define a notion of implementable strategy for one of the players, and give a simple algorithm to decide whether a given, general, strategy is implementable.

Let us introduce the main notions related to the game through an example. Consider the net in Fig. 1. Two players interact on it, the *user*, by controlling the light grey transitions, and the *environment*, by controlling the white ones. The game is asynchronous: the players can concurrently fire their transitions. The game is asymmetric: the user has the right to keep his transitions blocked when they are enabled, whereas the environment must guarantee progress of its transitions. In this example, we suppose that the user has full knowledge of the current marking, and that he has the goal of marking place q infinitely often. In order to win, the user must wait for the environment to choose between firing t_1 or t_2. He can do it, since the environment cannot delay this choice forever. In the former case, the user chooses u_1, otherwise u_2. The environment is then forced to fire either v_1 or v_2, with the effect of marking q, and then to fire z, reproducing the initial marking.

Formally, a Petri net is a tuple $\Sigma = (P, T, F, m_0)$, where P is the set of places, T the set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ the flow relation, $m_0 : P \to \mathbf{N}$ the initial marking. A marking is a map $m : P \to \mathbf{N}$. We suppose the reader knows the definition of the firing rule, denoted $m[t\rangle m'$, and how to compute the set of reachable markings. Let M be the set of reachable markings in Σ; then Σ is 1-safe iff $\forall m \in M, \forall p \in P, m(p) \leq 1$. We assume that the game is played on a 1-safe Petri net. In a 1-safe Petri net a marking can be interpreted as the characteristic function of a subset of places. Hence, in the rest of the paper we will denote

ATAED'22, June 20-21, 2022, Bergen, Norway

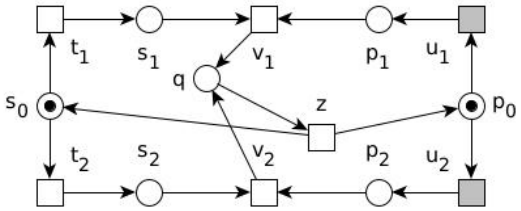

Figure 1: A game net

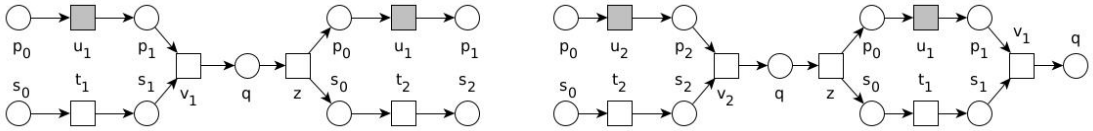

Figure 2: A full play (left) and a partial play (right)

markings as sets of places. We denote with $\mathsf{MG}(\Sigma) = (M, T, A, m_0)$ the *sequential marking graph* of Σ, where $A = \{(m, t, m') : m, m' \in M, t \in T and\ m[t\rangle m'\}$ is the set of labelled arcs. We denote with T_u the set of transitions controlled by the user, and with T_e the set of transitions uncontrollable for him. We assume that $T_e \cup T_u = T$, $T_e \cap T_u = \emptyset$.

We will also refer to the *unfolding* of the net (see [2] for the formal definitions). The unfolding is an acyclic, possibly infinite, net representing all the possible histories of the executions of a net system. In the unfolding of a net system, only forward conflicts are allowed: two events can share a precondition, but no postcondition. Since the unfolding is acyclic, the reflexive and transitive closure of the flow relation is a partial order. If two events e_1 and e_2 are in conflict, then we say that every descendant of e_1 is in conflict with every descendant of e_2. Two elements are *concurrent* if they are neither ordered nor in conflict. The events of the unfolding are partitioned into *controllable* and *uncontrollable* events depending on their correspondence to occurrences of controllable or uncontrollable transitions, respectively.

A *run* is a subnet of the unfolding representing an execution, i.e. it is a net without conflicts and close with respect to the past of its elements. A run is *maximal* if no event can be added without creating a conflict.

A *play* in the game is formally defined as a run in the unfolding of the net, maximal with respect to uncontrollable transitions. The winning condition for the user is a set of plays. The user wins a play if the play belongs to the winning condition. In the example discussed above, the winning condition is formed by all the maximal runs with an infinite number of occurrences of the place q. Figure 2 shows, on the left, a play ending in a deadlock (the environment wins) and, on the right, the initial segment of an infinite play (the user wins).

2. Strategies

In order to reach his goal, the user can apply a strategy. We assume that the user has no memory and only a partial knowledge of the current state of the net. This is formalized by assuming an equivalence relation, denoted by $\equiv$, on the set of reachable markings. The equivalence classes of this relation will be called *observations*, and a strategy is defined as a map from observations to sets of controllable transitions. For example, if we assume the user may observe only some places, two markings are considered equivalent if they share the same observable places. An example of equivalence relation between partially observable markings has been considered in the game presented in [3].

Let Obs be the set of observations. Then a strategy is a map $\alpha : \mathsf{Obs} \to 2^{T_u}$. The notion of observation can be extended to unfoldings: a maximal set of pairwise concurrent places in the unfolding is called a *B-cut*. Every B-cut corresponds to a reachable marking. We say that two B-cuts are equivalent if their corresponding markings are equivalent.

Two B-cuts of a run π, γ and γ', are ordered, denoted $\gamma < \gamma'$, iff $\gamma \neq \gamma'$ and $\forall b \in \gamma, \exists b' \in \gamma'$ such that: $b \leq b'$. A sequence $\gamma_1\gamma_2...\gamma_n...$ of B-cuts is *increasing* if $\gamma_i < \gamma_j$ for each $i, j \in \mathbf{N}$ with $i < j$.

A play π *complies* with a strategy α if (1) there is a sequence μ of observations which is the projection of an increasing sequence of B-cuts in π, and (2) every controllable event in π is chosen by the strategy in one of the observations in μ.

Given a strategy α, we can construct the reduced version of $\mathsf{MG}(\Sigma)$, by removing arcs corresponding to controllable transitions not chosen by the strategy, and then removing all states and transitions unreachable from the initial marking. The reduced version will be denoted by $\mathsf{MG}_\alpha(\Sigma)$, and the operation will be referred to as the α-reduction of $\mathsf{MG}(\Sigma)$. Fig. 3 shows the reduced version of the marking graph of the net in Fig. 1, when α is the strategy described above.

2.1. Implementable strategies

As defined above, the notion of strategy is not encoded in the net system itself. We will now investigate the possibility of defining a strategy within the net.

The idea we pursue is this: a strategy is encodable within a net if its decisions can be represented as extra places, corresponding to causal relations from uncontrollable transitions to controllable transitions. We will then say that a strategy is implementable if we can add new places to the given net, so that the runs of the augmented nets are exactly the plays complying with the strategy in the original net. Formally, we need a notion of "augmented" net system, in which we add places that restrict the possible behaviours.

For $\Sigma_i = (P_i, T_i, F_i, m_{0i})$, with $i = 1, 2$, we write $\Sigma_1 \sqsubseteq \Sigma_2$ if Σ_2 is obtained by adding new places to Σ_1, hence $P_1 \subseteq P_2$, $T_1 = T_2$, $F_1 \subseteq F_2$ and new arcs in F_2 have one end in $P_2 \setminus P_1$, $m_{01} \subseteq m_{02}$ and $m_{02} \setminus m_{01} \subseteq P_2 \setminus P_1$.

A given strategy α might prohibit any occurrence of a given controllable transition, and might make some uncontrollable transition unreachable in complying plays. Let T_α be the set of reachable transitions in complying plays, and let Σ_α be the net system whose underlying net is the subnet generated by T_α, i.e.: $\Sigma_\alpha = (P_\alpha, T_\alpha, F_\alpha, m_{0\alpha})$, where $P_\alpha = {}^\bullet T_\alpha \cup T_\alpha^\bullet$, F_α is F

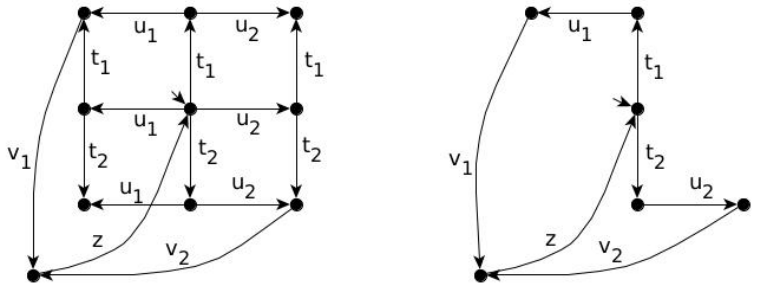

Figure 3: The marking graph of the net in Fig. 1 and its reduced version $\mathsf{MG}_\alpha(\Sigma)$

restricted to $(P_\alpha \times T_\alpha) \cup (T_\alpha \times P_\alpha)$ and $m_{0\alpha} = m_0 \cap P_\alpha$.

Definition 1. *A strategy α for a game on Σ is* implementable *if there exists a 1-safe net system $\Sigma' = (P', T_\alpha, F', m'_0)$ such that (1) $\Sigma_\alpha \sqsubseteq \Sigma'$; (2) the marking graph of Σ' is isomorphic to $\mathsf{MG}_\alpha(\Sigma)$.*

Consider again the net system in Fig. 1. Suppose that the user can observe all the places in the net, so that it has full knowledge of the current marking. If his goal is to reach infinitely often the place q, then a winning strategy is defined by the following clauses: (1) $\alpha(\{p_0, s_0\}) = \emptyset$; (2) $\alpha(\{p_0, s_1\}) = \{u_1\}$; (3) $\alpha(\{p_0, s_2\}) = \{u_2\}$. In this case, the strategy α does not prevent the occurrence of any controllable transition, it only selects which one between u_1 and u_2 to choose, depending on the observation of $\{p_0, s_1\}$ or $\{p_0, s_2\}$; therefore the set of reachable transitions in plays complying with α, T_α, is T itself, and then $\Sigma_\alpha = \Sigma$.

The marking graph of Σ, $\mathsf{MG}(\Sigma)$, is shown on the left side of Fig. 3, whereas the reduced version $\mathsf{MG}_\alpha(\Sigma)$ is on the right side of the same figure. The information needed to take the right decision in accordance with α, in this simple case, can be encoded by adding to Σ a couple of places, one going from t_1 to u_1, the other from t_2 to u_2, as shown in Fig. 4, on the right, where the new places are drawn in dark grey. It is easy to see that the obtained system has a marking graph isomorphic to $\mathsf{MG}_\alpha(\Sigma)$.

3. An algorithm to decide if a strategy is implementable

In this section, we propose a simple algorithm to decide whether a given strategy is implementable. The algorithm checks whether the reduced marking graph is isomorphic to the marking graph of a 1-safe Petri net, and relies on the theory of regions [4]. We first recall some basic notions related to regions of transition systems.

Region theory and separation problems As remarked above, each node of the marking graph of a 1-safe Petri net is a set of places. Hence, we can associate to each place its *extension*, namely the set of reachable markings to which it belongs: $\forall p \in P \quad r(p) = \{m \in M \mid p \in m\}$. The extension of a place satisfies the *uniform crossing* property: for any given transition t, if

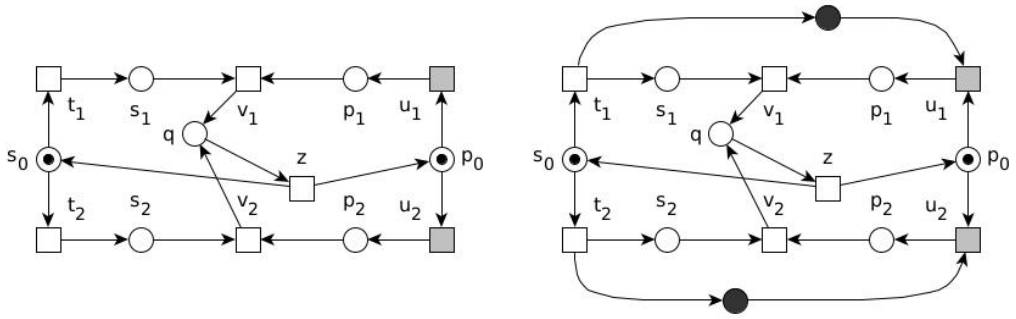

Figure 4: A game net with an implemented strategy

one occurrence of t in the marking graph enters $r(p)$, then all occurrences of t do the same, and analogously when an occurrence leaves $r(p)$. This property can be defined more abstractly for general labelled transition systems, giving the notion of a region of a transition system.

The synthesis problem consists in deciding if a given labelled transition system is isomorphic to the marking graph of a net system, and, if so, in constructing such a net. The problem admits a solution if the set of regions satisfies the so-called state separation problems and the event-state separation problems. In this case, we say that the transition system is separated.

Consider now a marking graph $\mathsf{MG}(\Sigma)$, a strategy α, and the reduced transition system $\mathsf{MG}_\alpha(\Sigma) = (Q, T_\alpha, A, q_0)$. Then, it is easy to prove that, for each region r of $\mathsf{MG}(\Sigma)$, the set $r \cap Q$ is a region of $\mathsf{MG}_\alpha(\Sigma)$. This implies that all state separation problems of $\mathsf{MG}_\alpha(\Sigma)$ are solved by those regions, since $\mathsf{MG}(\Sigma)$ is separated by definition. This is not the case for the event-state separation problems: for each edge labelled with t removed from $\mathsf{MG}(\Sigma)$ in the state s, we need to check if there is a region without state s from which t leaves.

The algorithm Given a strategy α for a game on Σ, we can now describe an algorithm to decide whether α is implementable. The algorithm first computes $\mathsf{MG}_\alpha(\Sigma)$, and then checks if it is separated by trying to solve each new event-state separation problem. The new regions solving those problems, if any, encode the flow of information from observations to the controllable part of the system.

4. Conclusion

In this work we presented a notion of implementable strategy on 1-safe nets, and we provided an algorithm to decide whether the strategy is implementable. In future works, we plan to broad the definition of implementable strategy by allowing for strategies modelled by adding general bounded places. As an example, consider the net on the left of Fig. 5 where the user can observe the places $\{3, 4, 5, 6, 10\}$, and his goal is to reach place 12. A winning strategy is: $\alpha(\{4, 6, 10\})$ =H, $\alpha(\{3, 5, 10\}) = \alpha(\{3, 6, 10\}) = \alpha(\{4, 5, 10\}) =$ I. The choice of H can be represented by adding a place from B to H, and a place from D to H. Instead, the choice of I

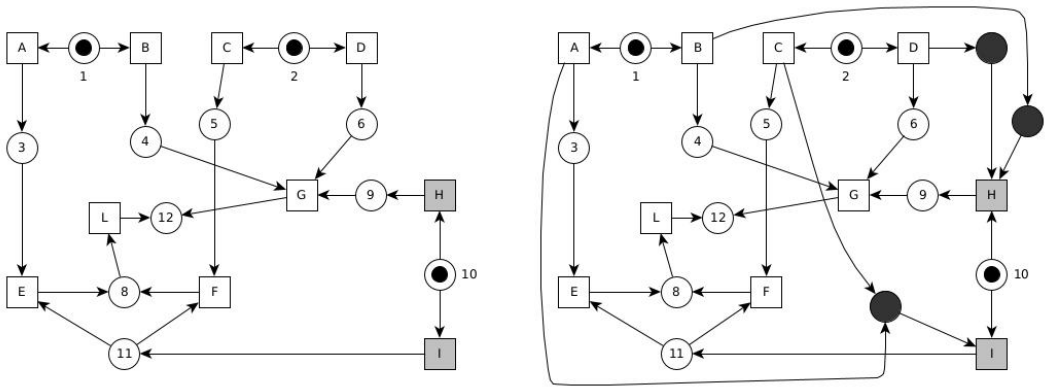

Figure 5: An example of non-implementable strategy

depends on the occurrence of either A or C, hence it may be represented with a place allowing for two tokens, making the net not 1-safe. Although the strategy is not implementable according to the given definition, the net on the right of Fig. 5 realizes the strategy.

Furthermore, we plan to relax the definition of implementable strategy, by allowing for implementations in which only parts of the behaviours in the reduced marking graph are reproducible on the net. Specifically, a strategy is implementable if we can add places to the initial net so that there is at least a maximal run in the unfolding of the augmented net, and all its maximal runs are associated to maximal paths of the reduced marking graph.

A similar use of regions can be found in applications to problems of control: in [5] and [6] regions are used to synthesize maximally permissive controllers avoiding unwanted states; [7] proposes an implementation based on region theory of a controller, applied to flexible manufacturing systems.

A general overview of works and approaches to automatic control using Petri nets can be found in [8].

Acknowledgments

This work is supported by the Italian MUR.

References

[1] F. Adobbati, L. Bernardinello, L. Pomello, A two-player asynchronous game on fully observable Petri nets., Trans. Petri Nets Other Model. Concurr. 15 (2021) 126–149.

[2] J. Engelfriet, Branching processes of Petri nets, Acta Inf. 28 (1991) 575–591. URL: https://doi.org/10.1007/BF01463946. doi:10.1007/BF01463946.

[3] F. Adobbati, L. Bernardinello, L. Pomello, Looking for winning strategies in two-player

games on Petri nets with partial observability, 2022. URL: https://arxiv.org/abs/2204.01603. doi:10.48550/ARXIV.2204.01603.

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

[5] A. Ghaffari, N. Rezg, X. Xie, Design of a live and maximally permissive Petri net controller using the theory of regions, IEEE Transactions on Robotics and Automation 19 (2003) 137–141. doi:10.1109/TRA.2002.807555.

[6] Z. Li, M. Zhou, M. Jeng, A maximally permissive deadlock prevention policy for fms based on petri net siphon control and the theory of regions, IEEE Transactions on Automation Science and Engineering 5 (2008) 182–188.

[7] S. Rezig, C. Ghorbel, Z. Achour, N. Rezg, PLC-based implementation of supervisory control for flexible manufacturing systems using theory of regions, International Journal of Automation and Control 13 (2019) 619–640.

[8] A. Giua, M. Silva, Petri nets and automatic control: A historical perspective, Annual Reviews in Control 45 (2018) 223–239.

Deadlocks and livelocks in resource constrained workflow nets

Gabriel Juhás[1], Ana Juhásová[2] and Tomáš Kováčik[1]

[1]*Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, Ilkovičova 3, 812 19 Bratislava, Slovakia*

[2]*BIREGAL s.r.o. Klincová 37/B, 821 08 Bratislava, Slovakia*

Abstract

The paper is presenting a method for detection of locks (both deadlocks and livelocks) in discrete event systems with shared resources and multiple instances modeled by resource constrained workflow nets. We consider that multiple instances with determined initial state and a correct final state are running in workflow nets. We consider workflow processes, in which instances can share several types of resources, with instances neither creating nor destroying shared resources. It means, that instances can use resources but the used resources are returned at the latest by the correct finish of the instance. Such resources are said to be durable. Examples of durable resources include resources of information systems, such as memory and processors or employees in roles in an organizational structure. We consider processes, which have enough resources to execute a single instance, such processes are said to be sound. A lock is a state of the process, where several instances are running but because of the lack of shared resources not all running instances can finish properly. The main result of the paper is the theorem stating that in sound workflow processes with several types of shared durable resources for given initial number of resources and an arbitrary unbounded number of running instances it is enough to test locks for a finite bounded number of instances, with the upper bound indicated.

Keywords

Petri net, Deadlock, Livelock, Resource constrained workflow net

1. Introduction

In [1] authors work with workflow processes with instances, in which instances share common resources. The processes are modeled by resource constrained workflow nets, where shared resources are modeled by so called static places. Except shared resources, instances are independent. Shared resources considered in [1] are durable, i.e. the resources are neither created nor destroyed by instances. Resources in information systems, such as memory, processors, i/o ports are typical examples of durable resources. Similarly, employees in particular roles considered by workflow processes can represent an example of durable resources.

The main problem solved in [1] is the problem of correct finish of all instances of a workflow process with shared durable resources. The problem is solved for processes with one type of durable resources. The method from [1] decides whether there is a number of shared resources

The Workshop Algorithms & Theories for the Analysis of Event Data (ATAED 2022): A Satellite Event of the 43rd International Conference on Application and Theory of Petri Nets and Concurrency, June 19 - 24, 2022, Bergen, Norway

gabriel.juhas@stuba.sk (G. Juhás); ana.juhasova@biregal.sk (A. Juhásová); tomas_kovacik@stuba.sk (T. Kováčik)

0000-0001-8302-5112 (G. Juhás); 0000-0003-3157-8609 (T. Kováčik)

such that for this number of resources and any greater number of resources any number of instances can be correctly finished. If the answer is true, then the workflow process is according to [1] called sound.

In [2] authors solve the soundness problem for several subclasses of resource constrained workflow nets. In [3] four necessary conditions of soundness of resource constrained workflow nets based on structural analysis are presented.

In [4, 5] we defined a technique based on a constructor and a runtime net to detect instance deadlocks of resource constrained workflow nets (called workflow nets with static places) for a fixed number of durable resources and an arbitrary number of instances. However, method from [4] did not work for livelocks.

In [6, 7] authors using a technique based on constructor and using a transformed net from [4] (called production net in [6, 7]) proved that the soundness problem for resource constrained workflow nets which are sound for one instance, is decidable even for the case where resources are not durable. The proof is based on the reduction to the home space problem of Petri nets, which was proved to be decidable in [8]. At the same time, the papers [6, 7] prove that in general, i.e. if the soundness for one instance is not required and the resources does not necessarily need to be durable, the problem of soundness in resource constrained workflow nets is undecidable. In [9] authors using the constructor from [4] proved, that if the nets are sound for one case, then the soundness problem is decidable even if the instances are not independent (the investigated net corresponding to our runtime net is not serializable). For more about serializability see e.g. [10, 11]. Paper [9] extends an unsuccessful attempt to prove decidability of soundness for resource constrained workflow nets presented in [12]. In the proof of results from [9] authors again used reduction to home space problem of Petri nets. As it is written at the end of the paper [9], soundness of resource constrained workflow nets is decidable, but up to now there is no effective algorithm, because the algorithm proposed in [9] requires the test of general reachability in possibly unbounded Petri nets [13, 14, 15, 16, 17]. The similar argument can be found at the end of original paper solving home space problem [8].

There are workflow processes, which for some fixed initial number of shared resources can correctly finish any number of instances, but for some greater number of shared resources the same process cannot finish for some number of instances running in parallel. Thus, such processes are not sound according to the definition of [1, 9]. An example of such a not sound process is on Figure 1, which can finish any number of instances for the given number of shared resources in static places *free key, free memory, free processor*. However, adding any number of resources to the static place *free key* will cause that the process may deadlock for the number of instances greater than 3. In general, once the number of resources in the place *free key* is smaller than the sum of free resources in places *free memory* and *free processor*, then the process will have no deadlock, otherwise it will contain a deadlock for any number of instances greater of equal the sum of shared resources in places *free memory* and *free processor*.

In this work we will prove, that in order to decide, whether the workflow process with arbitrary finite number of resource types and arbitrary unbounded number of instances with fixed initial number of durable shared resources will contain a lock (i.e. a deadlock or a liveclock), it suffices to check the existence of locks for bounded number of instances. We also will show how to compute such upper bounds.

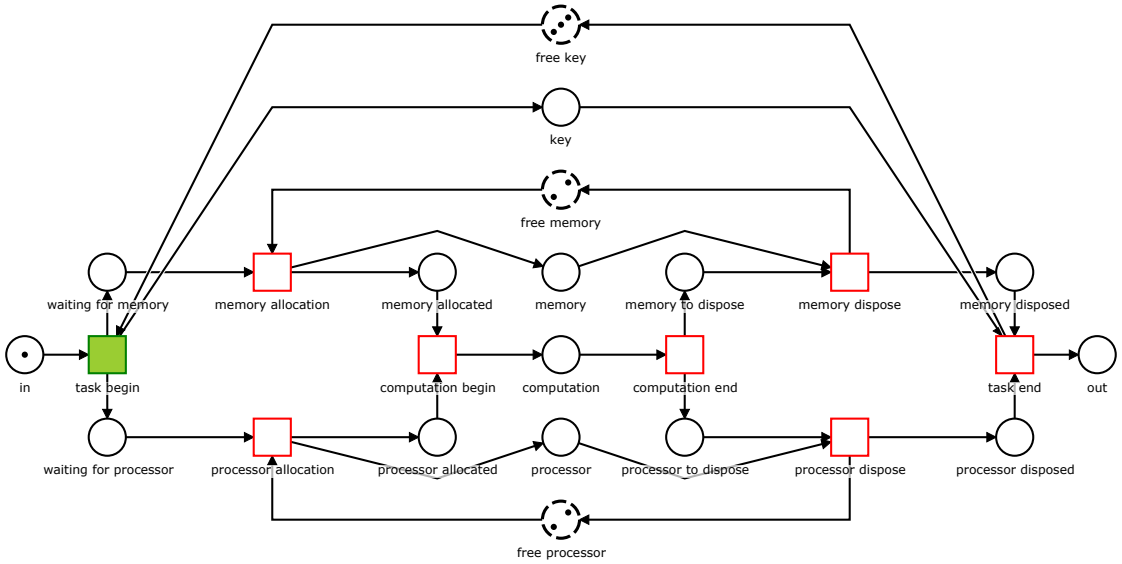

Figure 1: A marked remembering workflow net with static places, modeling allocation of memory units and processors to computing tasks with possibility to constrain the number of tasks, which can be executed in parallel.

2. Labelled transition systems

As a basic model of process behaviour we will use labelled transition systems [18, 19].

Definition 1 (Labelled transition system).
A labelled transition system is an ordered triple $(S, E, \longrightarrow)$*, where*

- S *is a set of states,*
- E *is a set of events,*
- $\longrightarrow \subseteq S \times E \times S$ *is a transition relation.*

The fact, that (s, e, s') *belongs to* $\longrightarrow$ *is referred as* $s \xrightarrow{e} s'$*.*

We will use labelled transition systems that have determined an initial state from which any other state is reachable.

To define reachability of states, we first define the notion of a sequence and the notion of an index set.

Definition 2 (Index set).
Index set $\mathbb{I}$ *is a subset of positive integers satisfying: if a positive integer* i *belongs to the index set* $\mathbb{I}$*, then any positive integer smaller than* i *belongs to the index set* $\mathbb{I}$*. If* $\mathbb{I}$ *is a finite nonempty set with the maximum* n*, then the number* n *is referred as* $max_{\mathbb{I}}$*. If* $\mathbb{I}$ *is the empty set, then* $max_{\mathbb{I}} = 0$*.*

Definition 3 (Sequence).
Let $\mathbb{I}$ be an index set and S be a set. Then a function $\alpha : \mathbb{I} \to S$ associating an element $\alpha(i)$ from the set S to each index i from the index set $\mathbb{I}$, is called a sequence of elements from the set S. If $\mathbb{I}$ is a finite set, then we say that α is finite sequence with length $max_{\mathbb{I}}$. Especially, if $\mathbb{I}$ is the empty set, then we say that α is the empty sequence. If $\mathbb{I}$ is equal to the set of all positive integers, then we say that α is an infinite sequence.

For illustration, consider a sequence of characters $\alpha = abba$ from the set of characters $S = \{a, b\}$. Intuitively, we understand that it is a finite sequence with length 4. In accordance with Definition 2 we use the index set $\mathbb{I} = \{1, 2, 3, 4\}$. The sequences α is formalized as follows: $\alpha(1) = a$, $\alpha(2) = b$, $\alpha(3) = b$ a $\alpha(4) = a$, i.e. the first element of the sequence α is character a, the second element is b, the third element is again b a the fourth element is character a.

Definition 4 (Occurrence sequence, reachability).
Let $(S, E, \longrightarrow)$ be a labelled transition systems. Let $s \in S$ be a state and let $\epsilon : \mathbb{I} \to E$ be a finite sequence of events. Sequence ϵ can occur in state s iff there exists a function $\sigma : \mathbb{I} \cup \{0\} \to S$ such that $\sigma(0) = s$ and for each positive integer $i \in \mathbb{I}$: $\sigma(i-1) \xrightarrow{\epsilon(i)} \sigma(i)$. The occurrence of the sequence ϵ in the state s leads to the state $\sigma(max_{\mathbb{I}})$.

A state $s' \in S$ is reachable from a state s iff there is an occurrence sequence ϵ, which leads from s to s'.

The fact that a finite sequence ϵ can occur in s and its occurrence leads to s' is referred as $s \xrightarrow{\epsilon} s'$.

Remember, that according to Definition 4 for any state s we have: the empty sequence can occur in s and its occurrence leads to s, i.e. s is self-reachable by occurrence of the empty sequence.

Definition 5 (Pointed labelled transition system).
A pointed labelled transitions system is an ordered quadruple $(S, E, \longrightarrow, q)$, where $(S, E, \longrightarrow)$ is a labelled transition system and $q \in S$ is its initial state, while for each state $s \in S$ there holds that s is reachable from q.

3. Petri nets

In this section we briefly introduce the basic definition of Petri nets [20, 21, 22, 23, 24, 25].

Definition 6 (Petri net).
A Petri net is an ordered quadruple (P, T, I, O), where:

- *P is a set of places*
- *T is a set of transitions*
- *$P \cap T = \emptyset$*
- *$I : P \times T \to \mathbb{N}$ is an input function, where $\mathbb{N}$ stands for the set of non-negative integers.*
- *$O : P \times T \to \mathbb{N}$ is an output function.*

A state of a Petri net is given by a marking.

Definition 7 (Marking).
Let $PN = (P, T, I, O)$ be a Petri net. A function $m : P \to \mathbb{N}$ attaching a non-negative integer to each place is called a marking of Petri net PN. The value $m(p)$ defines the number of tokens in a place $p \in P$. A marking will be written in form of a sum of marking of places, i.e. $\sum_{p \in P} m(p)p$. The set of places, for which $m(p)$ is greater than zero, is called the support of marking m and is denoted by $sup(m)$, formally for each $p \in P$ there holds that p belongs to $sup(m)$ iff $m(p) > 0$.

Places can be understood as types of tokens. As an example, if a set of places is given by $P = \{a, b, c\}$, we have three types of tokens, tokens of type a, tokens of type b and tokens of type c. Consider a marking $m : P \to \mathbb{N}$ such that $m(a) = 2$, $m(b) = 0$ and finally $m(c) = 1$. It means, that the marking expresses a state with two tokens of type a, no tokens of type b and one token of type c, which will be expressed by expression $2a + 0b + 1c$, or more simply by expression $2a + c$ (i.e. two tokens of type a and a token of type c), generally by expression $\sum_{p \in P} m(p)p$.

The fact, that for markings m and m' there holds $m(p) \leq m'(p)$ for each $p \in P$, is denoted by $m \leq m'$. Further, $m < m'$, respectively $m' > m$ denotes the fact that $m \leq m'$ and there exists $p \in P$ such that $m(p) < m'(p)$. If $m < m'$, we say that m is smaller than m', and m' is greater than m, respectively. The sum of two markings m a m' is marking $m + m'$ such that $(m + m')(p) = m(p) + m'(p)$ for each $p \in P$. If $m \leq m'$, then the difference of markings m' a m is given by marking $m' - m$ such that $(m' - m)(p) = m'(p) - m(p)$ for each $p \in P$.

Dynamics of a Petri net is given by firing of transitions.

Definition 8 (Transition firing).
Let $PN = (P, T, I, O)$ be a Petri net. Let $m : P \to \mathbb{N}$ be a marking and $t \in T$ be a transition of the net PN. Transition t is enabled to fire in marking m iff for each $p \in P$ there holds: $m(p) \geq I(p, t)$.

If transition t is enabled to fire in marking m, then its firing in m leads to marking m' such that for each $p \in P$ there holds: $m'(p) = m(p) - I(p, t) + O(p, t)$.

We will consider that a Petri net has initial marking. A Petri net together with an initial marking will be referred as marked Petri net.

Definition 9 (Marked Petri net).
A marked Petri net is an ordered quintuple $MPN = (P, T, I, O, m_0)$, where $PN = (P, T, I, O)$ is a Petri net and m_0 is a marking of PN called initial marking.

Graphically places are depicted as circles, markings of places by number of tokens inside of places, transitions are depicted as squares. Enabled transitions are filled (by green color). Non-zero value of input function $I(p, t)$ is expressed by an arrow from place p to transition t, while the value greater than one is written by the arc. Non-zero value of output function $O(p, t)$, is expressed by an arrow from transition t to place p, while the value greater than one is written by the arc. These non-zero values will be referred as weights of arcs. All Petri nets used in this work where modelled in an online Petri net editor, accessible at www.petriflow.com.

Petri nets defines a labelled transition system in a natural way by firing enabled transitions.

Definition 10 (Reachability graph of a Petri net).
Let $PN = (P, T, I, O)$ be a Petri net. A labelled transition system $(S, E, \longrightarrow)$, where

- *S is the set of all markings, i.e. the set of all functions from P to non-negative integers $\mathbb{N}$,*
- *$E = T$,*
- *$m \xrightarrow{t} m'$ iff transition t is enabled to fire in marking m and firing of t in m leads to marking m',*

is called reachability graph of Petri net PN.

We also define reachability graphs of marked Petri nets.

Definition 11. (Reachability graph of a marked Petri Net)
Let $MPN = (P, T, I, O, m_0)$ be a marked Petri net. Let $(S, E, \longrightarrow)$ be reachability graph of Petri net $PN = (P, T, I, O)$. Let $[m_0\rangle$ denote the set of all markings reachable from m_0 in reachability graph of Petri net PN. Then pointed labelled transition system $([m_0\rangle, E, \longrightarrow \cap([m_0\rangle \times E \times [m_0\rangle), m_0)$ is called reachability graph of marked Petri net MPN. If sequence ϵ can occur in m and its occurrence in m leads to m' in the reachability graph of MPN, then we say that sequence ϵ is enabled to fire in marking m in net MPN and its firing in m leads to the marking m' in net MPN. We also say that marking m' is reachable from marking m in the marked Petri net MPN.

Definition 12 (Boundedness).
A marked Petri net $MPN = (P, T, I, O, m_0)$ is bounded iff there exists a function $b : P \to \mathbb{N}$, such that for each marking m reachable from m_0 in MPN there holds: $m \leq b$. Function b is called the bound of net MPN.

In the case that a marked Petri net has finitely many places is the boundedness equivalent with the finiteness of the number of reachable markings. For more results on boundedness see e.g. [26, 27, 28].

Corollary 1. *A marked Petri net $MPN = (P, T, I, O, m_0)$ with finite set of places is bounded iff the number of markings reachable from m_0 in MPN is finite.*

4. Workflow nets

We focus on processes, where instances has a unique start and unique correct finish. In literature, such systems are modeled by workflow nets [29, 30, 31, 32].

Definition 13 (Workflow net).
A workflow net is a Petri net $PN = (P, T, I, O)$ with finite number of places and transitions in which there exists a unique place $in \in P$ and a unique place $out \in P$ such that

- *for each $t \in T$ there holds $O(in, t) = 0$ and there exists such $t \in T$ that $I(in, t) \neq 0$,*
- *for each $t \in T$ there holds $I(out, t) = 0$ and there exists such $t \in T$ that $O(out, t) \neq 0$.*

Place in is called input place of workflow net PN and place out is called output place of workflow net PN.

A marked workflow net is a workflow net with a special initial marking, in which only the input place is marked.

Definition 14 (Marked workflow net).
A marked workflow net is a marked Petri net $MPN = (P, T, I, O, m_0)$, where $PN = (P, T, I, O)$ is a workflow net and $m_0 = in$, i.e. $m_0(in) = 1$ and $m_0(p) = 0$ for each $p \in P$ different from in.

5. Petri nets and workflow nets with static places

Processes with instances and shared resources will be modeled by Petri nets with static places [4, 5]. Static places inspired by static variables in Java will model shared resources. Petri nets with static places were inspired by workflow nets with static places originally defined in papers [33, 1, 3] under the name resource constrained workflow nets.

Definition 15 (Petri net with static places).
Petri net with static places is a quintuple $PNS = (D, S, T, I, O)$, where

- *D is a set of dynamic places*
- *S is a set of static places*
- *$D \cap S = \emptyset$*
- *$PN = (P = D \cup S, T, I, O)$ is a Petri net.*

Static places will be depicted by dashed circles.

Definition 16. (Marked Petri net with static places)
Marked Petri net with static places is a sextuple $MPNS = (D, S, T, I, O, m_0)$, where

- *$PNS = (D, S, T, I, O)$ is a Petri net with static places*
- *$MPN = (P = D \cup S, T, I, O, m_0)$ is marked Petri net.*

We say that a transition is enabled to fire in a marked Petri net with static places $MPNS$ if it is enabled to fire in marked Petri net MPN. A reachability graph of a marked Petri net with static places $MPNS$ is the reachability graph of marked Petri net MPN. All notion defined for marked Petri net MPN will analogously used for marked Petri net with static places $MPNS$.

Definition 17. (Workflow net with static places/resource constrained workflow net) *A workflow net with static places, also called resource constrained workflow net, is a Petri net with static places $W = (D, S, T, I, O)$, where $PN = (P = D \cup S, T, I, O)$ is a workflow net such that $in \in D$ and $out \in D$.*

Initial marking of a marked workflow net with static places contains one token in the input place, no tokens in dynamic places different from the input place and any number of tokens representing shared resources in static places.

Definition 18. (Marked workflow net with static places)
A marked workflow net with static places is a sextuple $MW = (D, S, T, I, O, m_0)$, where

- *$W = (D, S, T, I, O)$ is a workflow net with static places*
- *m_0 is initial marking such that $m(in) = 1$ and $m(d) = 0$ for each dynamic place $d \in D$, which is different from in.*

In comparison with [33, 1, 3] we formalize durable resources via (weak) complementary places [34] of static places.

Definition 19. (Remembering workflow net with static places)
Let $W = (D, S, T, I, O)$ be such workflow net with static places, that for each places $s \in S$ there exists a unique (weak)complementary place $d_s \in D$ different from in and out, which for each $t \in T$ satisfies $I(d_s, t) - O(d_s, t) = O(s, t) - I(s, t)$. Workflow net W is called remembering workflow net with static places. The set of all complementary places of static places is denoted by D_S.

The equality $I(d_s, t) - O(d_s, t) = O(s, t) - I(s, t)$ in a marked remembering workflow net with static places implies that in case that $O(s, t) - I(s, t) = 0$, we also have $I(d_s, t) - O(d_s, t) = 0$. It means, that firing any transition may create a token in a complementary place d_s of a static place s iff it consumes a token from the static place s. Because in any initial marking the complementary place d_s is empty, we get that the sum of tokens in a static place s and its complementary place d_s equals $m_0(s)$.

Corollary 2. *Let $MW = (D, S, T, I, O, m_0)$ be a marked remembering workflow net with static places. Let $s \in S$ be a static place and let $d_s \in D_S$ be its complementary place. Then for each marking m reachable from m_0 there holds: $m(s) + m(d_s) = m_0(s)$ and therefore $m(s) \leq m_0(s)$.*

1-soundness of a marked workflow net with static places is defined analogously to soundness of workflow nets in [32] as the ability to finish correctly a single instance.

Definition 20. (Final marking of marked workflow net with static places)
Let $MW = (D, S, T, I, O, m_0)$ be a marked workflow net with static places. A marking m_f of net MW reachable from m_0 is called a final marking of MW iff there holds: $m_f(out) = 1$, $m_f(d) = 0$ for each $d \in D$ different from out.

Definition 21. (1-soundness of marked workflow net with static places)
Let $MW = (D, S, T, I, O, m_0)$ be a marked workflow net with static places and let $out \in P$ denote the output place. Marked workflow net with static places MW is 1-sound iff for each marking m reachable from m_0 there holds:

- *there exists a final marking m_f of net MW, which is reachable from marking m,*
- *if $m(out) \geq 1$ then m is a final marking of net MW.*

For marked remembering workflow nets with static places we have:

Corollary 3. *Let $MW = (D, S, T, I, O, m_0)$ be a marked remembering workflow net with static places and let m_f be a final marking of MW. Then $m_f(s) = m_0(s)$ for each $s \in S$ and therefore m_f is a unique final marking of MW.*

For 1-sound marked remembering workflow nets with static places we get:

Lemma 1. *If a marked remembering workflow net with static places is 1-sound then the number of its reachable markings is finite, i.e. the net is bounded.*

Proof. Let the number of markings reachable from m_0 is not finite. According to Dickson's lemma [35], let m and m' are markings reachable from m_0 such that $m < m'$. From definition of 1-soundness we get that $m'(out) = 1$ or $m'(out) = 0$ and $m(out) = 1$ or $m(out) = 0$. Because $m < m'$, we also get $m(s) \leq m'(s)$ for each $s \in S$.

- The combination $m'(out) = 1$ and $m(out) = 1$ means that $m' = m_f = m$, what contradicts that $m < m'$.
- The combination $m'(out) = 0$ and $m(out) = 1$ is in contradiction with $m < m'$.
- The combination $m'(out) = 1$ and $m(out) = 0$ means that $m' = m_f$. Assuming that $m_f > m$ we get $m(d) = 0$ for each $d \in D$). From first item of 1-soundness we further get that m_f is reachable from m by firing of a sequence of transitions. From definition of transition firing it is clear that whenever a sequence is enabled to fire from a marking, it is enabled to fire from any greater marking, and therefore also from m_f. Because firing of that sequence leads from m to m_f, its firing from m_f leads to the marking m'' such that $m''(out) = m_f(out) + (m_f(out) - m(out))$, i.e. $m''(out) = 1 + (1 - 0)) = 2$, what contradicts with the second item of 1-soundness implying that for each marking m'' reachable from m_0 there holds $m''(out) \leq 1$.
- The combination $m'(out) = 0$ a $m(out) = 0$ means by assumption $m < m'$, that there exists $p \in P$ different from out such that $m'(p) > m(p)$. At the same time from the first item of 1-soundness we get that m_f is reachable from m by firing a sequence of transitions. From definition of transition firing it is clear that whenever a sequence is enabled to fire from a marking, it is enabled to fire from any greater marking, and therefore also from m'. Because firing of that sequence leads from m to m_f, its firing from m' leads to the marking $m'' = m' + (m_f - m)$. Because $m'(out) = 0$ and $m(out) = 0$, we get $m''(out) = 1$. At the same time $m'(p) > m(p)$. If $p \in D$, we get $m''(p) > 0$, what contradicts the second item of 1-soundness. If $p \in S$, we get $m''(p) > m_f(p)$, what contradicts Corollaries 2 a 3.

It means that if a marked remembering workflow net with static places is 1-sound, then the number of its reachable markings is finite and therefore the net is bounded. □

6. Reachability nets

In order to investigate the ability to finish properly arbitrary number of instances running in parallel, we need to find a net, which will simulate the runtime environment with copies of a

dynamic part of a workflow net for each instance sharing just static places. Such a net should prohibit the mixing of tokens from different copies. In other words, the net simulating the runtime environment should separate reachable markings of dynamic places of instances. One possible way to reach this goal is to use the reachability graph of the original workflow net as an inspiration.

First we will define some basic notions which will be further used to define such so called reachability net.

Definition 22 (Notation).
Let P be a set, let A be a set and let D be a set such that $D \subseteq P$. Let $m : P \to A$ be a function. Let $m|D$ denote restriction of m to D, i.e. $m|D : D \to A$ such that $m|D(d) = m(d)$ for each $d \in D$. Let S be a set such that $S \subseteq P$, and let $D \cap S = \emptyset$ (D a S are disjoint), then we denote $m|D \cup m|S = m|(D \cup S)$. Let us denote by $P \setminus D$ the set difference P and D, as a set satisfying $D \cap (P \setminus D) = \emptyset$ and $(P \setminus D) \cup D = P$ (we define the set difference only for the case that D is a subset of P). Let $[P \to A]$ denote the set of all functions from P to A. For A being a finite set, let $|A|$ denote the number of elements of A.

Definition 23 (Set of reachable D-markings).
Let $MPN = (P, T, I, O, m_0)$ be a marked Petri net. Let D be a subset of places, i.e. $D \subseteq P$. Let m be a marking of MPN. Then function $m|D$ is called a D-marking. By symbol $[m_0\rangle|D$ we denote the set of all D-markings w satisfying: $w \in [m_0\rangle|D$ iff there exists such $m \in [m_0\rangle$ that $w = m|D$. It means that symbol $[m_0\rangle|D$ denotes the set of all D-markings $m|D$ such that m is reachable from m_0. We also say that $[m_0\rangle|D$ denote the set of all D-markings reachable from initial D-marking $m_0|D$ in MPN.

Each reachable D-marking $m|D$ from $[m_0\rangle|D$ of the original workflow net will become to be a place in reachability net. In addition, static places of the original workflow net will be static places of reachability net. Elements of transition relation (m, t, m') z $\longrightarrow$ of the reachability graph of original workflow net will be used to create transitions of the reachability net. A triple $(m|D, t, m'|D) \in ([m_0\rangle|D) \times T \times ([m_0\rangle|D)$ will be a transition of the reachability net iff $m \xrightarrow{t} m'$. A transition $(m|D, t, m'|D)$ of the reachability net will consume exactly one token from the place $m|D$ of the reachability net and it will produce exactly one token in place $m'|D$ of the reachability net. Arcs and their weights between static places and transition $(m|D, t, m'|D)$ of the reachability net will be identical with arcs between static places and transition t of the original net. Finally, we add a constructor consisting of two transitions $\{new, stop\}$ and one place {source}.

In graphical expression of reachability nets, we will label transition $(m|D, t, m'|D)$ of the reachability net only by the name of transition t of the original net.

Definition 24 (Reachability net).
Let $MW = (D, S, T, I, O, m_0)$ be a marked workflow net with static places and let new, $stop$ and $source$ denote elements satisfying $\{new, stop, source\} \cap (D \cup S \cup T) = \emptyset$. Reachability net of the net MW is the marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$, where

- $P^r = ([m_0\rangle|D) \cup S \cup \{source\}$.

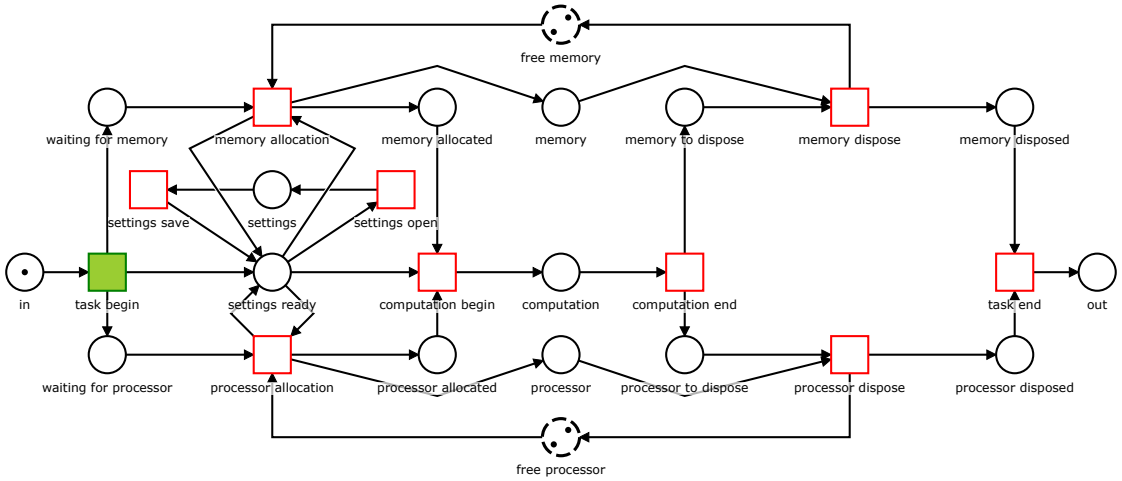

Figure 2: A 1-sound marked remembering workflow net with static places, modeling allocation of memory units and processors to computing tasks with possibility to change the setting for the computation.

- $T^r = T^a \cup \{new, stop\}$, *where* T^a *is the set of all triples* $(x,t,y) \in ([m_0\rangle|D) \times T \times ([m_0\rangle|D)$, *for which there exists a triple* $(m,t,m') \in [m_0\rangle \times T \times [m_0\rangle$, *such that* $x = m|D$, $y = m'|D$ *and* $m \xrightarrow{t} m'$, *i.e. the transitions of the reachability net are* new *a* $stop$ *and such triples* (x,t,y), *where* x *a* y *are D-markings and* t *is enabled to fire in marking* m *in* MW *and its firing leads to* m' *in* MW, *while* $x = m|D$ *and* $y = m'|D$.
- $I^r(x,(x,t,y)) = 1$ *for each* $(x,t,y) \in T^a$
- $I^r(source, new) = 1$ *and* $I^r(source, stop) = 1$, *i.e. constructor* new *is enabled to fire only in a marking with a token in place* $source$, *similarly* $stop$
- $I^r(s,(x,t,y)) = I(s,t)$ *for each* $s \in S$, $(x,t,y) \in T^a$, *i.e. arcs from static places to copies of transitions are copied*
- $O^r((x,t,y),y) = 1$ *for each* $(x,t,y) \in T^a$
- $O^r(in, new) = 1$ *and* $O^r(source, new) = 1$, *i.e. firing of constructor* new *creates a token in place* in *and returns a token to place* $source$ *(remember that from Definition 18 we get* $in = m_0|D$*)*
- $O^r(s,(x,t,y)) = O(s,t)$ *for each* $s \in S$, $(x,t,y) \in T^a$, *i.e. arcs from copies of transitions to static places are copied*
- $I^r(p,t) = 0$ *a* $O^r(p,t) = 0$ *for each other pairs* $(p,t) \in (P^r \times T^r)$
- $m_0^r(source) = 1$, $m_0^r|S = m_0|S$ *a* $m_0^r(x) = 0$ *for each* $x \in [m_0\rangle|D$, *i.e. in the initial marking is one token in place* $source$, *tokens in static places are copied and places of the reachability net equal to reachable D-markings of the net* MW *are empty.*

To illustrate a reachability net, we will consider the net in Figure 2

Existence of the complementary places of static places in remembering workflow net implies following result:

Corollary 4. *Let $MW = (D, S, T, I, O, m_0)$ be a marked remembering workflow net with static places. Then for each two markings m and m' from $[m_0\rangle$ there holds: if $m|D = m'|D$ then $m = m'$.*

Corollary 5. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places. Then its reachability net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ has a finite number of places P^r and a finite number of transitions T^r.*

Places in the reachability net of remembering net represent states of the instance, including information how many tokens from static places are used by the instance. Each token in a place of the reachability net represent an instance in the corresponding state. The number of tokens in a place of the reachability net determine how many instances are in that state. Thus, marking of the reachability net determines how many instances are in states represented by single places of the reachability net.

Definition 25 (Final marking of a reachability net).
Let $MW = (D, S, T, I, O, m_0)$ be a marked workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. A marking m_f^r of the reachability net MPN reachable from m_0^r, is called a final marking of MPN, if $m_f^r(x) = 0$ for each $x \in P^r \setminus (S \cup \{out\})$ (where out denotes in accordance with the introduced notation the reachable D-marking given by function $1 \cdot out$ from D to $\mathbb{N}$).

A lock of the reachability net is defined as a marking, from which no final marking is reachable.

Definition 26. (Lock, deadlock and livelock of a reachability net)
Let $MW = (D, S, T, I, O, m_0)$ be a marked workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. A marking m^r of the reachability net MPN reachable from the initial marking m_0^r, is called a lock, if from m^r no final marking of the reachability net MPN is reachable. If no transition of the reachability net MPN is enabled to fire in a lock m^r, then it is called a deadlock of MPN, otherwise it is called a livelock of MPN.

7. Basic locks of a reachability net

Following result, which is implied directly by the construction of the reachability net is important for detection of locks of reachability nets.

Lemma 2. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Let marking m_1^r be reachable from m_0^r in the reachability net MPN. Then for arbitrary marking $w : (P^r \setminus S) \to \mathbb{N}$ satisfying $m_1^r|(P^r \setminus S) > w$ there exists such marking m_2^r reachable from m_0^r in MPN, that $m_2^r|(P^r \setminus S) = w$.*

In the following definition we will divide the places of the reachability net according to the fact, whether they represent states, in which resources from static places are used.

Definition 27 (Places with resources).
Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW.

- *A place $x \in P^r \setminus S$ of the reachability net is called a place without resource s for $s \in S$ if either $x = source$ or for the complementary places $d_s \in D_S$ of place s there holds $x(d_s) = 0$.*
- *The set of all places without resource s is denoted by B_s^r.*
- *If place $x \in P^r \setminus S$ of the reachability net is a place without resource for each $s \in S$, we call it simply a place without resources.*
- *The set of all places without resources is denoted by B^r.*
- *A place $x \in P^r \setminus S$ of the reachability net is called place with resource s for $s \in S$ if for the complementary place $d_s \in D_S$ of s there holds $x(d_s) > 0$. The set of all places with resource s is denoted by Z_s^r.*
- *If for a place $x \in P^r \setminus S$ of the reachability net there is $s \in S$ such that x is a place with resource s, then we call it simply a place with resources.*
- *The set of all places with resources is denoted by Z^r.*
- *For a place $x \in Z^r$ of the reachability net we denote by symbol $Z(x)$ the set of such $s \in S$, for which $x \in Z_s^r$.*

Similarly as states, we divide the transitions of the reachability net for those, that need tokens from static places to be enabled to fire and those, that do not need tokens from static places. We will divide the places according to the fact, whether there is a transition, which moves a token from a place with resources and at the same time it requires resources.

Definition 28. (Transitions requiring resources, critical places)
Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW.

- *A transition $(x, t, y) \in T^a$ is called a transition requiring resources if $I(s, t) > 0$ for some $s \in S$.*
- *If for a place with resources $x \in Z^r$ there holds that there is a transition $(x, t, y) \in T^a$ requiring resources, then $x$is called a critical place of the reachability net MPN.*
- *The set of all critical places is denoted by K^r.*

On Figure 3 there is the reachability net of the 1-sound marked remembering workflow net with static places from Figure 2.

The set of places without resources B^r in the reachability net on Figure 3 is given by six places: by place *source*, by place *in*, by place *waiting for memory + waiting for processor + settings ready*, by place *waiting for memory + waiting for processor + settings*, by place *memory disposed + processor disposed* and by the place *out*.

The set of places with resources Z^r is given by ten places, two of which are critical.

First critical place of the reachability net is its place *waiting for memory + processor allocated + settings ready + processor*. Second critical place of the reachability net is its place *memory allocated + waiting for processor + settings ready + memory*.

On Figure 4 there is a livelock of the process for ten instances.

Corollary 6. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Let $s \in S$ be a static place and let $d_s \in D_S$ be its complementary place. Then for each marking m^r reachable from m_0^r there holds $m^r(s) + \sum_{x \in Z_s^r} m^r(x) \cdot x(d_s) = m_0(s)$, and therefore $m^r(s) \leq m_0^r(s) = m_0(s)$.*

Another important result is given as follows:

Lemma 3. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Then for arbitrary marking m_1^r reachable from m_0^r in the reachability net there exists a marking m_2^r reachable from m_1^r such that $m_2^r(x) = 0$ for each $x \in Z^r \setminus K^r$.*

Proof. We show, that if a token is in a place $x \in Z^r \setminus K^r$, we can move that token to a places without resources from B^r, or to a critical place from K^r. Repeating the procedure we can set to zero all places $x \in Z^r \setminus K^r$. Because the original remembering workflow net is 1-sound, from each $x \in Z^r \subseteq [m_0\rangle|D$ is in original workflow net reachable a final marking $out \cup m_0|S$. It means, that for each place $x \in Z^r \setminus K^r$ there exists a finite sequence $\epsilon : \mathbb{I} \to T$ a $\sigma : \mathbb{I} \cup \{0\} \to [m_0\rangle|D$ such that $\sigma(0) = x$, $(\sigma(i-1), \epsilon(i), \sigma(i)) \in T^a$ for each positive integer $i \in \mathbb{I}$ and $\sigma(max_{\mathbb{I}}) = out$. Let $\alpha : \mathbb{I} \to T^a$ be a sequence of transitions from T^a such that $\alpha(i) = (\sigma(i-1), \epsilon(i), \sigma(i))$ for $i \in \mathbb{I}$.

- Let no transition $\alpha(i)$, where $i \in \mathbb{I}$, is a transition requiring resources. Then sequence α is enabled to fire in any marking of the reachability net $m_1^r \in [m_0^r\rangle$ satisfying $m_1(x) > 0$ and its firing leads to marking m_2^r, such that $m_2^r(x) = m_1^r(x) - 1$ and $m_2^r(out) = m_1^r(out) + 1$, i.e. firing of sequence α moves a token from a place with resources x of the reachability net to the place without resources out of the rechability net.
- Let there is an $i \in \mathbb{I}$ such that transition $\alpha(i)$ is a transition requiring resources. Let i be the smallest index, for which $\alpha(i)$ is a transitions requiring resources. Because x is not a critical place, $i \geq 2$.
  - Let $\sigma(i-1) \in B^r$, i.e. $\sigma(i-1)$ is a place without resources. Then sequence $\alpha|\{1, \ldots, i-1\}$ is enabled to fire in each marking of the reachability net $m_1 \in [m_0^r\rangle$ satisfying $m_1^r(x) > 0$ and its firing leads to marking m_2^r, such that $m_2^r(x) = m_1^r(x) - 1$ and $m_2^r(\sigma(i-1)) = m_1^r(\sigma(i-1)) + 1$, i.e. firing of sequence α moves a token from place x of the reachability net to a place without resources $\sigma(i-1)$ of the reachability net.
  - Let $\sigma(i-1) \in Z^r$, i.e. $\sigma(i-1)$ is a place with resources. Because $\alpha(i) = (\sigma(i-1), \epsilon(i), \sigma(i))$ is a transition requiring resources, $\sigma(i-1) \in K^r$, i.e. $\sigma(i-1)$ is a critical place. Then sequence $\alpha|\{1, \ldots, i-1\}$ is enabled to fire in each marking of the rechability net $m_1^r \in [m_0^r\rangle$ satisfying $m_1(x) > 0$ and its firing leads to marking m_2^r, such that $m_2^r(x) = m_1^r(x) - 1$ and $m_2^r(\sigma(i-1)) = m_1^r(\sigma(i-1)) + 1$, i.e. firing of sequence α moves a token from place x of the reachability net to a critical place $\sigma(i-1)$ of the reachability net.

□

If we apply the result of Lemma 3 to locks, we get that from each lock of the reachability net we can reach a lock, in which from all places with resources only critical places are marked. These locks are called critical locks.

Definition 29 (Critical locks of a reachability net).
Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Then a lock m^r, such that $m^r(x) = 0$ for each $x \in Z^r \setminus K^r$, is called a critical lock of the reachability net.

Corollary 7. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Then there holds: if an arbitrary marking m_1^r reachable from m_0^r in the reachability net is a lock, then there exists a critical lock m_2^r reachable from m_1^r.*

The lock in Figure 4 is not a critical lock, because the place with resources *waiting for memory + processor allocated + settings + processor* and the place with resources *memory allocated + waiting for processor + settings + memory* are not empty. These places with resources are not critical places of the reachability net.

From marking in Figure 4 the marking in Figure 5 is reachable. The marking in Figure 5 is a critical lock.

A special set of critical lock are those locks, for which no other places except critical places and static places are marked. Such lock are called basic locks.

Definition 30 (Basic locks of a reachability net).
Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Then a critical lock m^r, such that $m^r(x) = 0$ for each place without resources $x \in B^r$, is called a basic lock of the reachability net.

The following results states, that if m_1^r is a critical lock, then marking m_2^r created from m_1^r by removing all tokens from all places without resources, is reachable from z m_0^r and therefore it is a basic lock of the reachability net.

Lemma 4. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Then there holds: if an arbitrary marking m_1^r reachable from m_0^r in the reachability net is a critical lock but not a basic lock, then m_2^r, such that $m_2^r(x) = m_1^r(x)$ for each place with resources $x \in Z^r$ and $m_2^r(x) = 0$ for each place without resources $x \in B^r$, is a basic lock of the reachability net.*

Proof. According to Lemma 2, m_2^r is reachable from m_0^r. Assume, that m_2^r is not a basic lock. Because m_1^r is a critical lock, $m_2^r(x) = 0$ for each $x \in Z^r \setminus K^r$, i.e. m_2^r satisfies the condition of critical locks. Because $m_2^r(x) = 0$ for each place without resources $x \in B^r$, m_2^r satisfies also the condition of basic locks. It means, that m_2^r is not a lock. Then there is a sequence of transitions

β enabled to fire in m_2^r, such that its firing leads to a final marking m_f^r, where $m_f^r|S = m_0^r|S$. Because $m_2^r(x) = m_1^r(x)$ for each place with resources $x \in Z^r$, i.e. in non-static places of the reachability net differ m_2^r and m_1^r only in marking of places without resources, there also holds that $m_2^r|S = m_1^r|S$, i.e. the markings of static places of m_2^r and m_1^r are equal. Then the sequence β is enabled to fire also in m_1^r and its firing leads to m_3^r such that $m_3^r|Z^r = 0$ and $m_3^r|S = m_0^r|S$.

Because original net MW is 1-sound, (similarly as in the proof of Lemma 3) from each $x \in B^r \subseteq [m_0\rangle|D$ is in MW reachable the final marking $out \cup m_0|S$ of MW. It means, that for each place $x \in B^r$ there is a finite sequence $\epsilon : \mathbb{I} \to T$ a $\sigma : \mathbb{I} \cup \{0\} \to [m_0\rangle|D$ such that $\sigma(0) = x$, $(\sigma(i-1), \epsilon(i), \sigma(i)) \in T^a$ for each finite positive integer $i \in \mathbb{I}$ and $\sigma(max_{\mathbb{I}}) = out$. Let $\alpha : \mathbb{I} \to T^a$ be the sequence of transitions from T^a such that $\alpha(i) = (\sigma(i-1), \epsilon(i), \sigma(i))$ for $i \in \mathbb{I}$. Sequence α is enabled to fire in each marking of the reachability net $m_1^r \in [m_0^r\rangle$ satisfying $m_1(x) > 0$ and $m_1^r|S = m_0^r|S$ and its firing leads to marking m_4^r, where $m_4^r(x) = m_1^r(x) - 1$ and $m_4^r(out) = m_1^r(out) + 1$, i.e. firing of sequence α moves a token from a place without resources x of the reachability net to a place without resources out of the reachability net, and also $m_4^r|Z^r = 0$, $m_4^r|B^r \setminus \{x, out\} = m_1^r|B^r \setminus \{x, out\}$ a $m_4^r|S = m_1^r|S = m_0^r|S$. By repeating the firing of sequence α we get marking m_5^r, where $m_5^r(x) = 0$ and $m_5^r(out) = m_1^r(out) + m_1^r(x)$, i.e. $m_1^r(x)$-times repeated firing of sequence α moves all $m_1^r(x)$ tokens from place without resources x of the reachability net to the place without resources out of the reachability net, and also $m_5^r|Z^r = 0$, $m_5^r|B^r \setminus \{x, out\} = m_1^r|B^r \setminus \{x, out\}$ and $m_5^r|S = m_1^r|S = m_0^r|S$. By repeating the procedure for such $x \in B^r$ that marking of x is not zero, we get a marking m_f^r such that $m_f^r|((Br \setminus \{out\}) \cup Z^r) = 0$ and $m_f^r|S = m_0^r|S$, i.e. we get a final marking of the reachability net. This contradicts with the assumption that m_1 is a critical lock. □

By application of Lemma 4 on critical lock from Figure 5 we get the reachable basic lock in Figure 6.

Altogether, we get the following main result.

Theorem 1. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Then there holds: if there exists a lock m_1^r, then there exists a basic lock m_2^r.*

We also get:

Corollary 8. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. If the reachability net has no critical places, then it has no locks.*

Critical places are bounded in the reachability net, and therefore basic locks are bounded, i.e. there are finitely many basic locks.

Definition 31 (Simple bound of critical places).
Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Let $k \in K^r$ be a critical place. Let $s \in Z(k)$ and let $d_s \in D_S$ be the complementary

place of s. Denote $pb(k,s) = m_0(s) \div k(d_s)$, where symbol $\div$ denotes integer division. Denote $sbound(k) = \min_{s \in Z(k)} pb(k,s)$ the minimum of values $pb(k,s)$ for $s \in Z(k)$. The values $sbound(k)$ is called the simple bound of a critical place k. The sum of values

$$sbound(K^r) = \sum_{k \in K^r} sbound(k)$$

is called the simple bound of critical places K^r.

Corollary 9. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Let m^r be an arbitrary marking of the reachability net reachable from m_0^r. For each critical place $k \in K^r$ there holds $m^r(k) \leq sbound(k)$, i.e. the number of tokens in any reachable marking m^r does not exceed the simple bound of a critical place. There holds that $\sum_{k \in K^r} m^r(k) \leq sbound(K^r)$. In addition, $sbound(K^r) \geq 1$.*

More exact upper bound of the number of tokens in critical places can be computed as a solution of the following integer linear programming problem.

Definition 32 (Bound of critical places).
Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Let $y : K^r \to \mathbb{N}$ be an integer solution of the linear inequality system

$$\sum_{k \in K^r} k(d_s) \cdot y(k) \leq m_0(s) \; for \; s \in S$$

$$y(k) \geq 0 \; for \; k \in K^r$$

which maximizes the objective function

$$\sum_{k \in K^r} k(d_s) \cdot y(k)$$

This maximal value

$$bound(K^r) = \sum_{k \in K^r} k(d_s) \cdot y(k)$$

is called the bound of critical places K^r.

From the formulation of the integer linear programming problem we get the following result.

Corollary 10. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Let m^r be an arbitrary marking of the reachability net reachable from m_0^r. Then there holds $\sum_{k \in K^r} m^r(k) \leq bound(K^r)$. Moreover, there holds that $sbound(K^r) \geq bound(K^r)$.*

Methods for solving the integer linear programming problems can be found e.g. in [36].

For the reachability net in Figure 4 of the 1-sound marked remembering workflow net with static places from Figure 2 we get $sbound(K^r) = bound(K^r) = 4$.

From definition of the reachability net we get the following result.

Lemma 5. *Let $MW = (D, S, T, I, O, m_0)$ be a 1-sound marked remembering workflow net with static places and let a marked Petri net $MPN = (P^r, T^r, I^r, O^r, m_0^r)$ be the reachability net of the net MW. Let m_1^r be an arbitrary marking of the reachability net reachable from m_0^r. Then for each marking m_2^r reachable from m_1^r there holds:*

$$\sum_{x \in [m_0\rangle|D} m_1^r(x) \leq \sum_{x \in [m_0\rangle|D} m_2^r(x)$$

If $m_1^r(source) = 0$ then there holds:

$$\sum_{x \in [m_0\rangle|D} m_1^r(x) = \sum_{x \in [m_0\rangle|D} m_2^r(x)$$

Consider a constrained reachability net for a given positive integer $n \in \mathbb{Z}$ by adding a place, let us call it a *buffer*, from which firing of transition new consumes just one token, while in the initial marking this place get n number of tokens, i.e. $m_0^r(\textit{buffer}) = n$.

Such constrained reachability net preserves all the reachable markings for which the sum of tokens in places from $[m_0\rangle|D$ does not exceed n. It also preserves the firing of all transitions from T^a in these markings. The transition new is enabled to fire in such a net exactly n-times. Such net is bounded and represent the running of at most n instances in parallel. Basic locks in such a constrained net can be defined analogously as for the reachability net without constraints, just allowing non zero marking of $buffer$.

If one set the initial value $m_0^r(\textit{buffer}) = sbound(K^r)$ or $m_0^r(\textit{buffer}) = bound(K^r)$ one can guarantee that whenever the reachability net has a basic lock then the constrained net has the basic lock differing at most in the marking of the added place *buffer*. The constrained reachability net with $m_0^r(\textit{buffer}) = sbound(K^r) = bound(K^r) = 4$ for the reachability net from Figure 3 is in Figure 7.

8. Conclusion

We have investigated workflow processes with independent instances, which share durable resources. We have considered the fixed number of resources and arbitrary number of instances. We modeled such nets by so called 1-sound marked remembering workflow nets with static places, which are in fact resource constrained workflow nets sound for one instance. We have shown that detection of deadlocks and livelocks of such processes can be reduced to detection of basic locks of the constrained bounded reachability nets of 1-sound marked remembering workflow nets with static places.

One of the main aims of the further research is to find out how to synthesize a minimally restrictive completion of a net which has deadlocks or livelock in order to avoid them. Another aim of the further research is to determine the set of initial markings of static places, i.e. to determine the number of resources, needed to avoid the deadlock and livelocks.

References

[1] K. Hee, A. Serebrenik, N. Sidorova, M. Voorhoeve, Soundness of resource-constrained workflow nets, in: Applications and Theory of Petri nets, LNCS, volume 3536, Springer, Berlin, Heidelberg, 2005, pp. 250–267. doi:10.1007/11494744_15.

[2] K. Barkaoui, R. Ben Ayed, Z. Sbai, Workflow soundness verification based on structure theory of Petri nets, International Journal of Computing and Information Sciences 5 (2007) 51–61.

[3] K. Hee, N. Sidorova, M. Voorhoeve, Resource-constrained workflow nets., Fundamenta Informaticae 71 (2006) 243–257.

[4] G. Juhás, I. Kazlov, A. Juhásová, Instance deadlock: A mystery behind frozen programs, in: Applications and Theory of Petri Nets, LNCS, volume 6128, Springer, Berlin, Heidelberg, 2010, pp. 1–17. doi:10.1007/978-3-642-13675-7_1.

[5] A. Juhásová, G. Juhás, Soundess of resource constrained workflow nets is decidable., Petri Net Newsletter. (2009) 3–6.

[6] M. Martos-Salgado, F. Rosa-Velardo, Dynamic soundness in resource-constrained workflow nets, in: Formal Techniques for Distributed Systems, LNCS, volume 6722, Springer, Berlin, Heidelberg, 2011, pp. 259–273. doi:10.1007/978-3-642-21461-5_17.

[7] M. R. Martos Salgado, Verification of priced and timed extensions of Petri Nets with multiple instances., Ph.D. thesis, 2016.

[8] D. F. Escrig, C. Johnen, Decidability of home space property, in: LRI Report, volume 503, Université Paris-Sud, 1989, pp. 1–25.

[9] N. Sidorova, C. Stahl, Soundness for resource-constrained workflow nets is decidable, IEEE Transactions on Systems, Man, and Cybernetics: Systems 43 (2013) 724–729. doi:10.1109/TSMCA.2012.2210415.

[10] K. Hee, N. Sidorova, M. Voorhoeve, Soundness and separability of workflow nets in the stepwise refinement approach, in: Applications and Theory of Petri nets, LNCS, volume 2679, Springer, Berlin, Heidelberg, 2003, pp. 337–356. doi:10.1007/3-540-44919-1_22.

[11] E. Best, P. Darondeau, Separability in persistent petri nets, Fundamenta Informaticae 113 (2011) 179–203.

[12] F. L. Tiplea, C. Bocaneala, Decidability results for soundness criteria of resource-constrained workflow nets, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 42 (2012) 238–249. doi:10.1109/TSMCA.2011.2147310.

[13] E. Mayr, Persistence of vector replacement systems is decidable, Acta Informatica 15 (1981) 309–318. URL: https://doi.org/10.1007/BF00289268. doi:10.1007/BF00289268.

[14] S. R. Kosaraju, Decidability of reachability in vector addition systems (preliminary version), in: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC '82, Association for Computing Machinery, New York, NY, USA, 1982, p. 267–281. doi:10.1145/800070.802201.

[15] E. W. Mayr, An algorithm for the general Petri net reachability problem, SIAM Journal on computing 13 (1984) 441–460.

[16] J. Lambert, A structure to decide reachability in Petri nets, Theoretical Computer Science 99 (1992) 79–104. doi:https://doi.org/10.1016/0304-3975(92)90173-D.

[17] J. Leroux, Vector addition system reachability problem: A short self-contained proof, ACM

SIGPLAN Notices 46 (2011) 307–316. doi:10.1145/1925844.1926421.
[18] R. Keller, Formal verification of parallel programs, Communications of the ACM 19 (1976) 371–384. doi:10.1145/360248.360251.
[19] G. Winskel, M. Nielsen, Models for Concurrency, Oxford University Press, Inc., USA, 1995, p. 1–148.
[20] R. M. Karp, R. E. Miller, Parallel program schemata, Journal of Computer and System Sciences 3 (1969).
[21] T. Murata, Petri nets: Properties, analysis and applications, Proceedings of the IEEE 77 (1989) 541–580. doi:10.1109/5.24143.
[22] J. L. Peterson, Petri nets, ACM Computing Surveys 9 (1977) 223–252. doi:10.1145/356698.356702.
[23] W. Reisig, A Primer in Petri Net Design, Springer, Berlin, Heidelberg, 1992.
[24] J. Desel, W. Reisig, Place/transition Petri nets, in: Lectures on Petri Nets I: Basic Models: Advances in Petri Nets, LNCS, volume 1491, Springer, Berlin, Heidelberg, 1998, pp. 122–173. doi:10.1007/3-540-65306-6_15.
[25] J. Desel, G. Juhás, What is a Petri net? An informal answer for an informed reader, in: Unifying Petri Nets, Advances in Petri Nets, LNCS, volume 2128, Springer, Berlin, Heidelberg, 2001, pp. 1–25. doi:10.1007/3-540-45541-8_1.
[26] R. Lipton, The reachability problem is exponential-space hard, in: Technical Report, volume 62, Department of Computer Science, Yale University, 1976.
[27] C. Rackoff, The covering and boundedness problems for vector addition systems, Theoretical Compututer Science 6 (1978) 223–231. doi:10.1016/0304-3975(78)90036-1.
[28] L. E. Rosier, H.-C. Yen, A multiparameter analysis of the boundedness problem for vector addition systems, Journal of Computer and System Sciences 32 (1986) 105–135. doi:https://doi.org/10.1016/0022-0000(86)90006-1.
[29] W. M. P. van der Aalst, Three good reasons for using a Petri-net-based workflow management system, in: Information and Process Integration in Enterprises, Springer US, Boston, MA, 1998, pp. 161–182. doi:10.1007/978-1-4615-5499-8_10.
[30] W. M. P. van der Aalst, Verification of workflow nets, in: Application and Theory of Petri Nets 1997, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 407–426. doi:10.1007/3-540-63139-9_48.
[31] W. M. P. van der Aalst, The application of Petri nets to workflow management, Journal of Circuits, Systems, and Computers 8 (1998) 21–66. doi:10.1142/S0218126698000043.
[32] W. Van Der Aalst, K. M. Van Hee, K. van Hee, Workflow management: models, methods, and systems, The MIT Press, Cambridge, Massachusetts, 2002.
[33] K. Barkaoui, L. Petrucci, Structural analysis of workflow nets with shared ressources, in: Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98). Computing science reports, volume 9807, TU Eindhoven, 1998, pp. 82–95.
[34] R. Devillers, The semantics of capacities in P/T nets, in: Advances in Petri Nets 1989, Springer, Berlin, Heidelberg, 1990, pp. 128–150. doi:10.1007/3-540-52494-0_28.
[35] D. Figueira, S. Figueira, S. Schmitz, P. Schnoebelen, Ackermannian and primitive-recursive bounds with Dickson's lemma, in: 2011 IEEE 26th Annual Symposium on Logic in Computer Science, 2011, pp. 269–278. doi:10.1109/LICS.2011.39.
[36] A. Schrijver, Theory of Linear and Integer Programming, John Wiley, 1986.

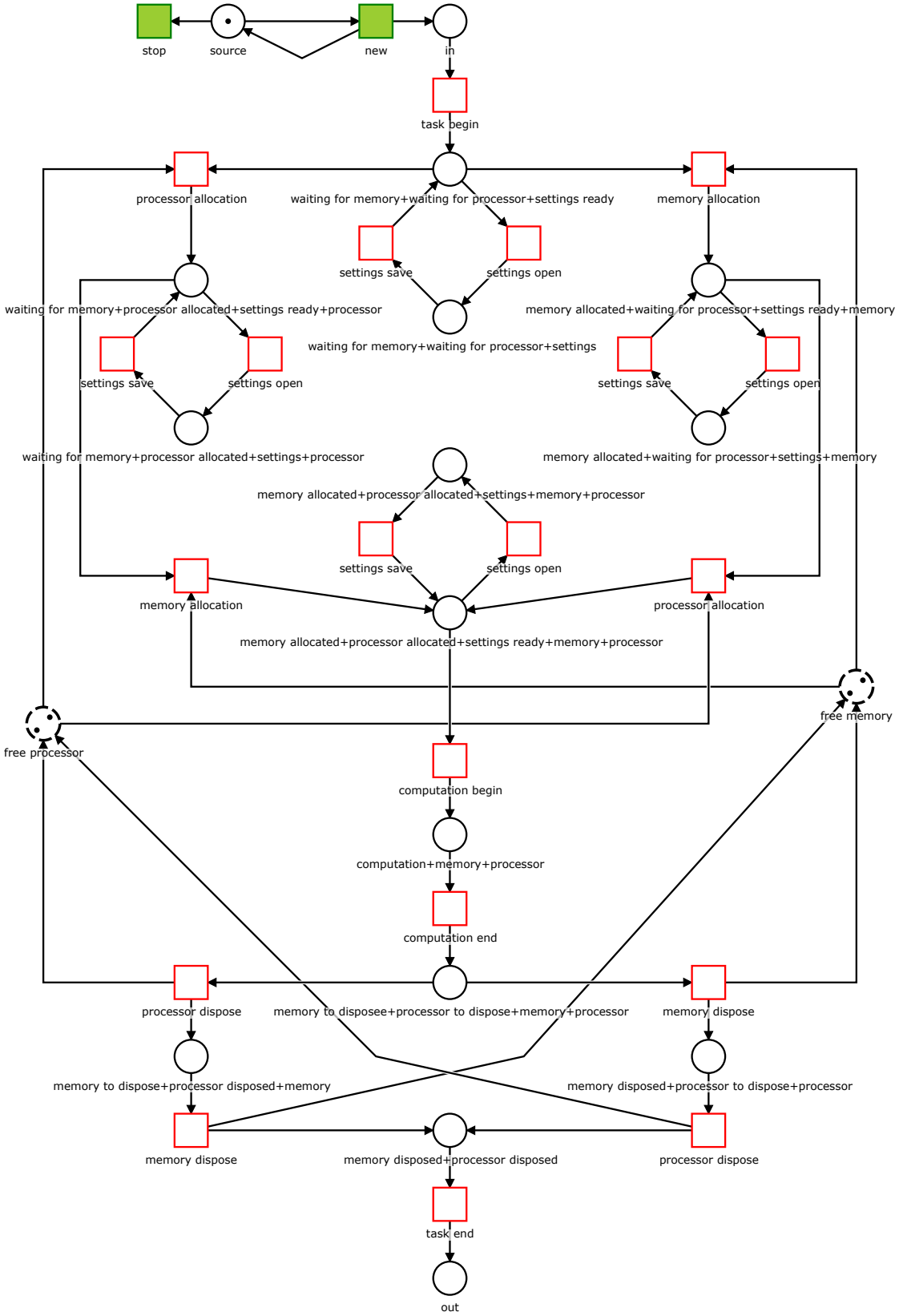

Figure 3: The reachability net of the 1-sound marked remembering workflow net with static places from Figure 2

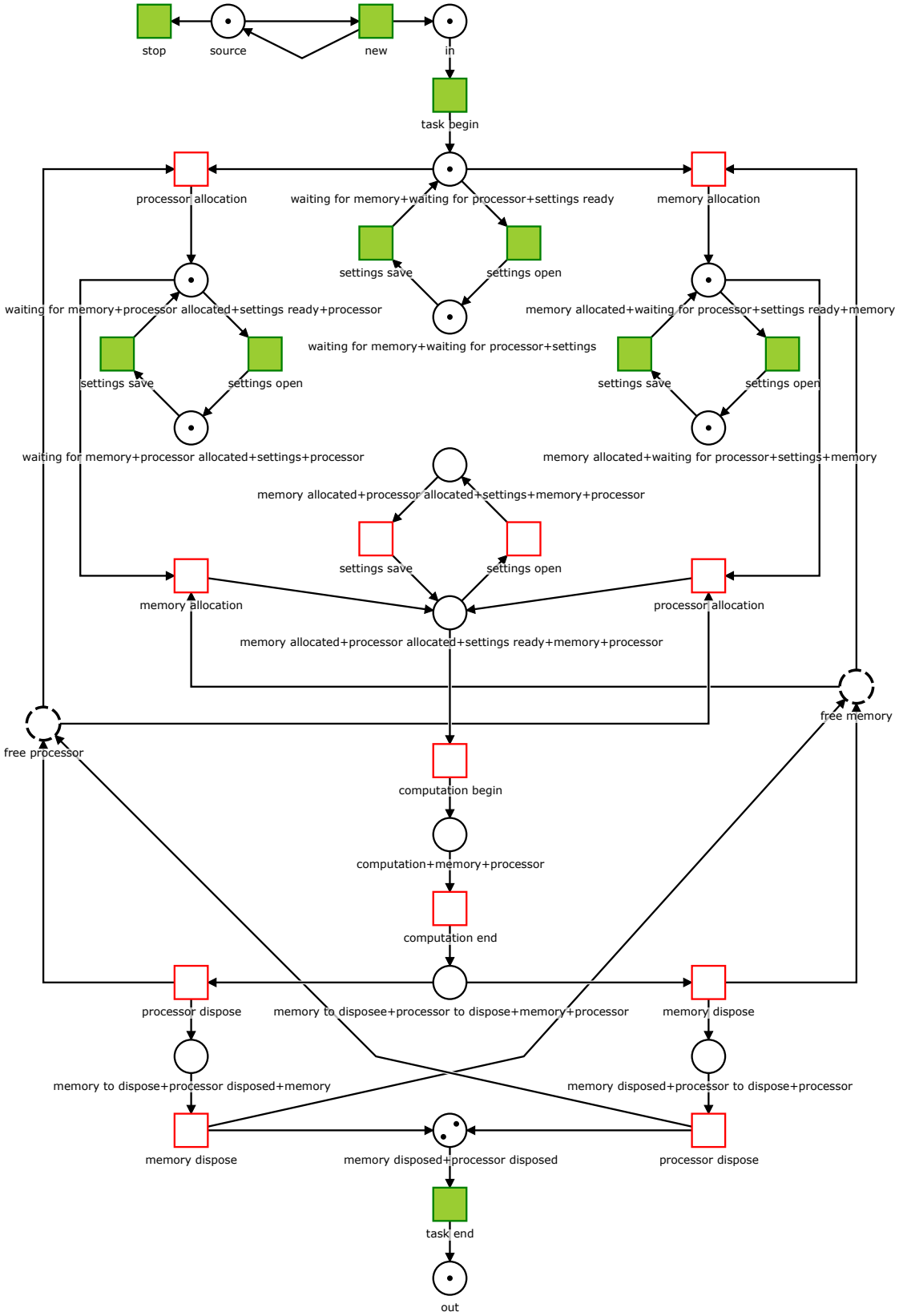

Figure 4: The reachability net of the 1-sound marked remembering workflow net with static places from Figure 2 in a livelock for ten instances

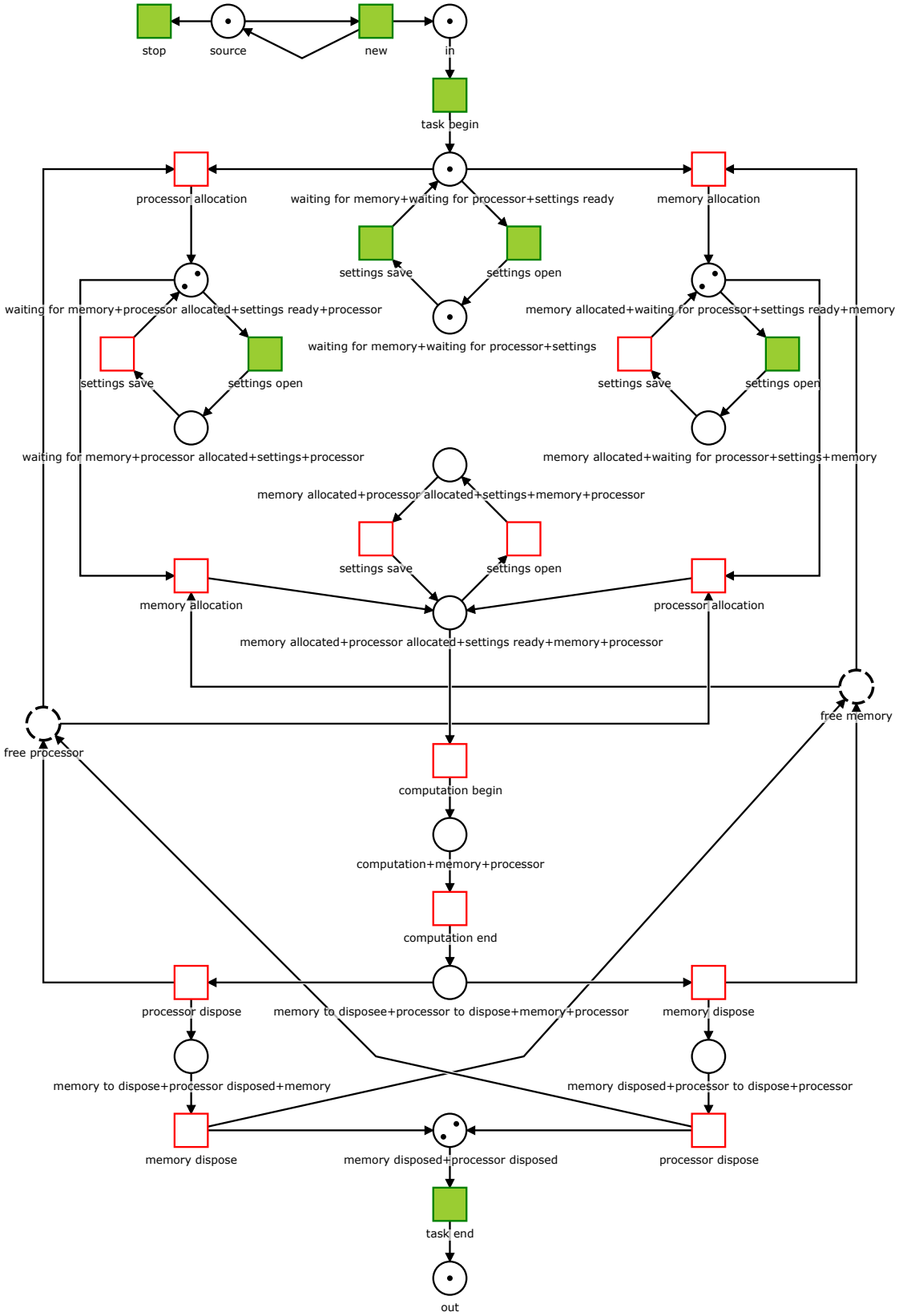

Figure 5: The reachability net of the 1-sound marked remembering workflow net with static places from Figure 2 in a critical livelock for ten instances

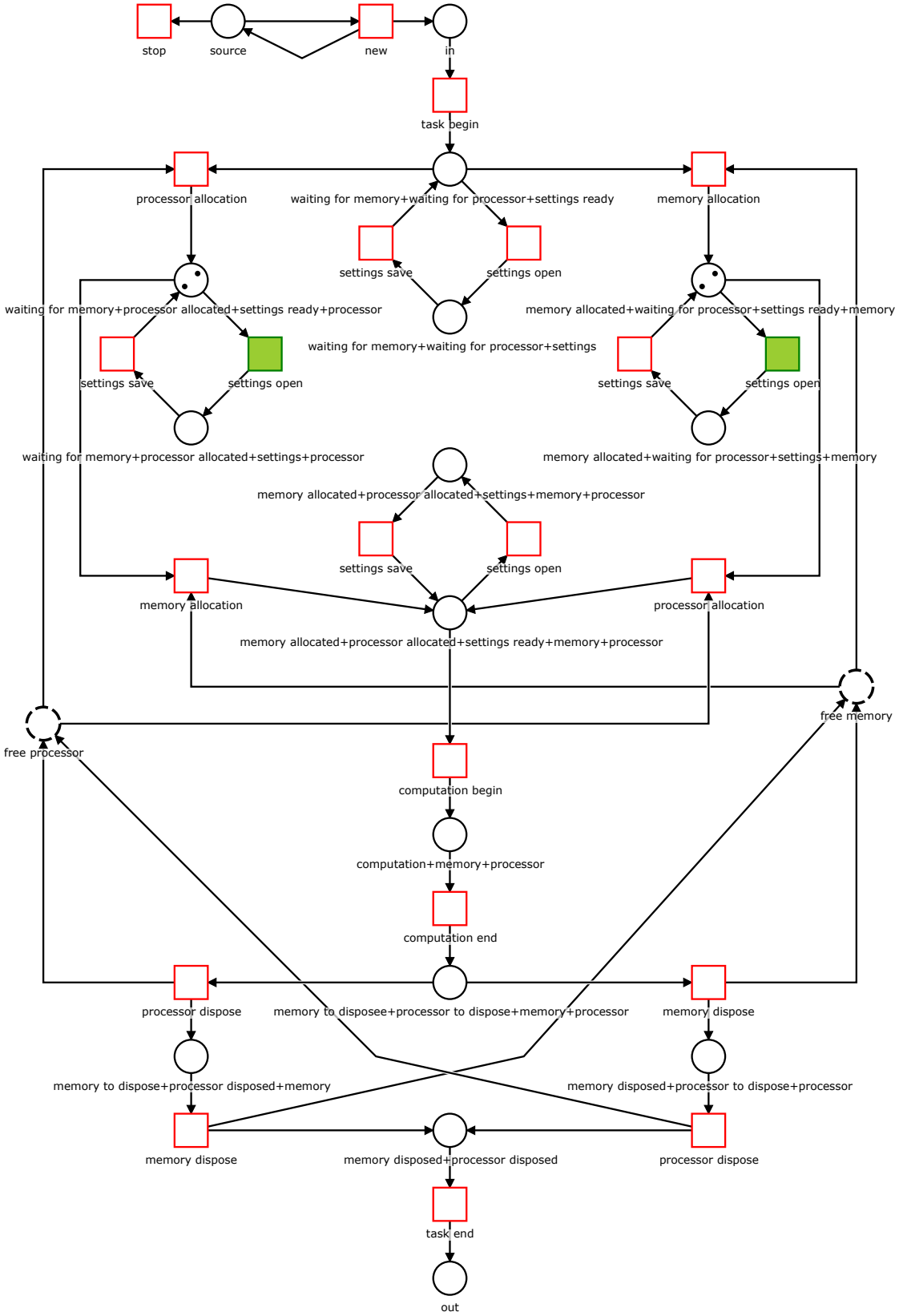

Figure 6: The reachability net of the 1-sound marked remembering workflow net with static places from Figure 2 in a basic livelock for four instances

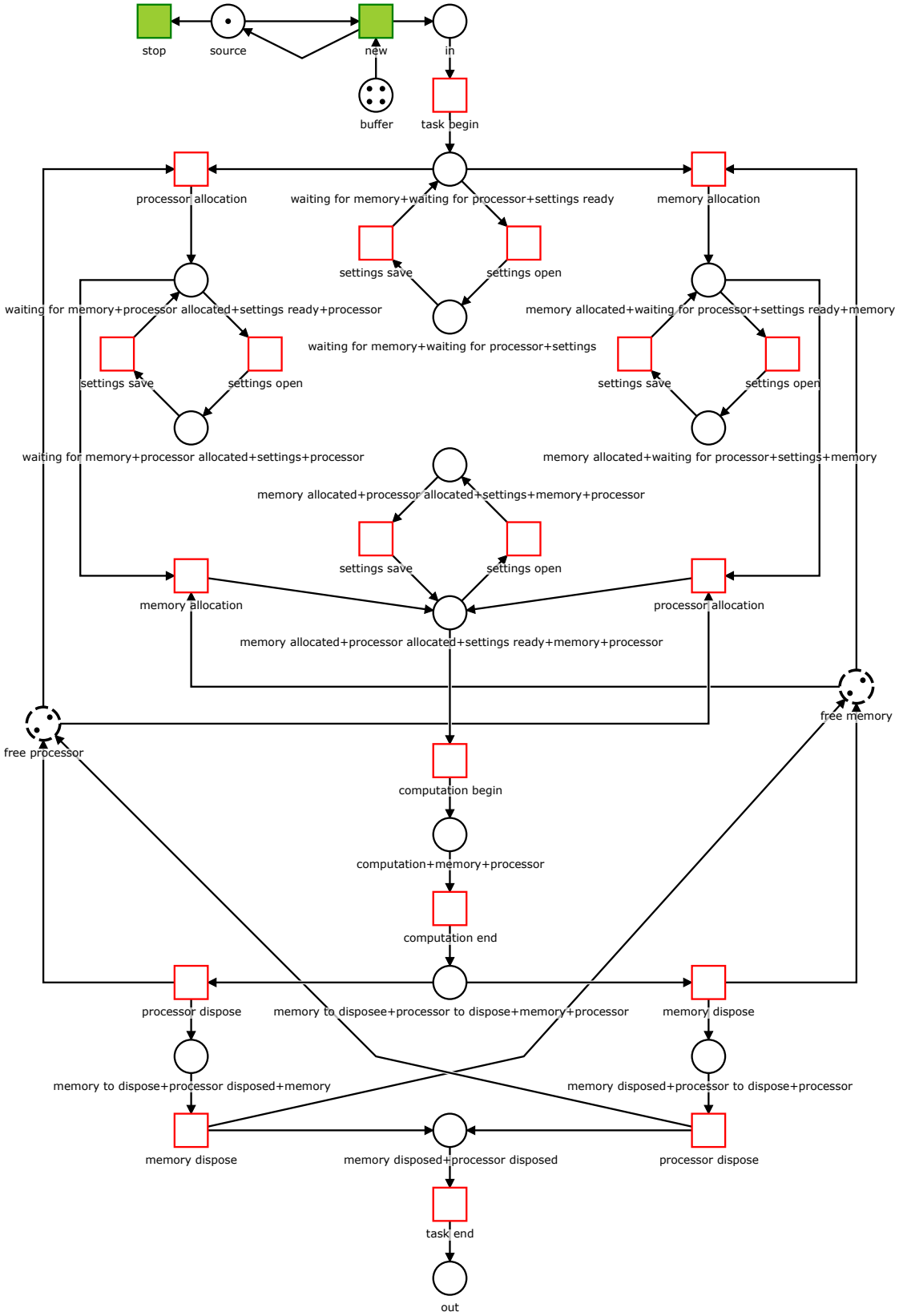

Figure 7: n-constrained bounded reachability net of the reachability net from Figure 3 for $n = sbound(K^r) = bound(K^r) = 4$