Constraint-based Analysis of Distributed Systems

Philipp Klara Johanna Meyer

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Theoretische Informatik

Technische Universität München
Fakultät für Informatik

Constraint-based Analysis of Distributed Systems

Philipp Klara Johanna Meyer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. Matthias Althoff

Prüfende der Dissertation: 1. Prof. Dr. Francisco Javier Esparza Estaun

2. Prof. Antonín Kučera, Ph.D.

Die Dissertation wurde am 01.03.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 28.06.2021 angenommen.

Philipp Klara Johanna Meyer
Constraint-based Analysis of Distributed Systems
Dissertation

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Theoretische Informatik

Abstract

Petri nets are a common model to analyze properties of distributed systems. In this thesis, we analyze the following properties of Petri nets and related models:

- safety and liveness properties of Petri nets, which can be used to analyze concurrent programs;
- well-specification and correctness of population protocols, a model of distributed computing based on interactions of simple mobile agents;
- quantitative properties concerning required resources and execution time of workflow nets, a model for analysis of business processes.

For Petri nets and population protocols, verification of these properties has a very high complexity, where in the worst case any complete procedure needs a non-elementary amount of memory. To address this problem, we develop incomplete procedures, i.e. procedures that may not always yield a result, but which have a much lower complexity and succeed on many examples from the literature. These procedures are based on constraints that allow implementation on top of efficient solvers and which can further be understood as certificates for the given property.

Our procedures for Petri nets are in many cases orders of magnitude faster than other tools and can verify safety and liveness properties for a larger number of instances. Our procedures for population protocols are the first to fully and automatically verify correctness of many protocols from the literature.

For workflow nets, which correspond to a simple subclass of Petri nets, polynomial-time algorithms exist for many problems, e.g. for computation of the expected cost. We analyze the problem of computing the minimal amount of resources needed to execute a workflow net as fast as possible. Further, we consider the problem of computing the expected execution time of a workflow net. For both problems, we show that no polynomial-time algorithm computing the respective quantity exists unless $\mathsf{P} = \mathsf{NP}$, even on restrictive subclasses of workflow nets.

We also develop constraint-based algorithms that compute these resource measures exactly for a large subclass of workflow nets and approximate them for general nets, and the first algorithm to compute the expected execution time. We evaluate our algorithms on a large set of workflow nets and show that they can compute or suitably approximate these quantities within milliseconds on these nets.

Zusammenfassung

Petrinetze sind ein weitverbreitetes Modell, um Eigenschaften von verteilten System zu analysieren. In dieser Dissertation untersuchen wir folgende Eigenschaften von Petrinetzen und verwandten Modellen:

- Sicherheits- und Lebendigkeitseigenschaften von Petrinetzen, was zur Analyse von nebenläufigen Programmen verwendet werden kann;
- Wohlspezifiziertheit und Korrektheit von Populationsprotokollen, ein Modell des verteilten Rechnens basierend auf Interaktionen von einfachen mobilen Agenten;
- quantitative Eigenschaften bezüglich benötigten Ressourcen und Ausführungszeit von Workflownetzen, einem Modell zur Analyse von Geschäftsprozessen.

Für Petrinetze und Populationsprotokolle hat Verifikation von diesen Eigenschaften eine sehr hohe Komplexität, wobei im schlimmsten Fall jede vollständige Prozedur eine nicht-elementare Menge an Speicher benötigt. Um dieses Problem anzugehen entwickeln wir unvollständige Prozeduren, welche nicht immer ein Ergebnis liefern, aber eine deutlich niedrigere Komplexität haben und auf vielen Beispielen aus der Literatur erfolgreich sind. Diese Prozeduren basieren auf Constraints, was eine Implementierung auf existierenden effizienten Lösern ermöglicht und welche als ein Zertifikat für die gegebene Eigenschaft aufgefasst werden können.

Unsere Prozeduren für Petrinetze sind in vielen Fällen um Größenordnungen schneller als andere Tools und können Sicherheits- und Lebendigkeitseigenschaften für mehr Instanzen verifizieren. Unsere Prozeduren für Petrinetze sind die ersten, die vollautomatisch Korrektheit von vielen Protokollen aus der Literatur verifizieren können.

Für Workflownetze, welche einer einfachen Teilklasse von Petrinetzen entsprechen, existieren Polynomialzeitalgorithmen für viele Probleme, z.B. für die Berechnung der erwarteten Kosten. Wir untersuchen das Problem der Berechnung der minimalen Anzahl an Ressourcen, um ein Workflownetz so schnell wie möglich auszuführen. Weiterhin betrachten wir das Problem der Berechnung der erwarteten Ausführungszeit eines Workflownetzes. Für beide Probleme zeigen wir, dass kein Polynomialzeitalgorithmus existiert außer $\mathsf{P} = \mathsf{NP}$, selbst auf eingeschränkten Teilklassen von Workflownetzen.

Wir entwickeln weiterhin Constraint-basierte Algorithmen, die diese Ressourcenwerte für eine große Teilklasse von Workflownetzen exakt berechnen und für allgemeine Netze approximieren, und den ersten Algorithmus zur Berechnung der erwarteten Ausführungszeit. Wir evaluieren unsere Algorithmen auf einer großen Menge von Workflownetzen und zeigen, dass sie diese Werte auf den Netzen innerhalb von Millisekunden berechnen oder geeignet approximieren.

Acknowledgments

First of all, I would like to thank my advisor Javier Esparza for his support and guidance throughout the years. He always had time for discussions on my research, and offered invaluable help in presenting my ideas in a clear and effective way. Without him, this thesis would not have been possible.

Big thanks go to my mentor Michael Luttenberger and to my long-time office mate Salomon Sickert, both for taking their time to proofread and comment on this thesis, and also for occasionally providing welcome distractions at work, which then often to lead to new ideas and also a very interesting and successful side project.

Further, I would like to express my thanks to my co-authors at our chair for the many fruitful discussions and collaborations: Stefan Jaax, Michael Blondin, Martin Helfrich, Antonín Kučera, Philip Offtermatt and Ruslán Ledesma-Garza. I learned a lot from the work we did together. My thanks also go to all my other past and present colleagues at I7 for the numerous pleasant conversations and joint activities. A special shout-out goes to Stefan Kugele for his cheerful welcomes.

Last, but not least, I thank my family for helping me follow my academic path, and especially my partner Antonia for her unconditional support and encouragement at all times.

Contents

Abstract iii

Acknowledgments vii

Contents ix

List of Figures and Tables xi

1 Introduction 1
- 1.1 Related Work . . . 3
- 1.2 Main Contributions . . . 4
- 1.3 Publication Summary . . . 6
- 1.4 Outline of the Thesis . . . 7

2 Preliminaries 9
- 2.1 Basic Notation . . . 9
- 2.2 Constraints and Complexity . . . 10
- 2.3 Petri Nets . . . 12

3 Constraint-based Petri Net Analysis 15
- 3.1 Problem Statement . . . 16
- 3.2 New Contributions for Safety Properties . . . 16
  - 3.2.1 Marking Equation . . . 17
  - 3.2.2 Refinement with Traps and Siphons . . . 18
  - 3.2.3 Iterative Refinement for Safety . . . 19
  - 3.2.4 Inductive Invariants for Safety . . . 21
  - 3.2.5 Experimental Evaluation . . . 22
- 3.3 New Contributions for Liveness Properties . . . 26
  - 3.3.1 T-surinvariants . . . 26
  - 3.3.2 Refinement with P-components . . . 27
  - 3.3.3 Refinement with Traps . . . 30
  - 3.3.4 Certificates for Termination . . . 32
  - 3.3.5 Experimental Evaluation . . . 32
- 3.4 Related Work . . . 36
- 3.5 Open Problems . . . 38

4 Verification of Population Protocols 39
4.1 Problem Statement . . . 42
4.2 New Contributions . . . 42
4.2.1 Layered Termination . . . 43
4.2.2 Strong Consensus . . . 45
4.2.3 The Class of WS^3-protocols . . . 47
4.2.4 Experimental Evaluation . . . 48
4.3 Related Work . . . 48
4.4 Open Problems . . . 51

5 Quantitative Analysis of Workflow Nets 53
5.1 Problem Statement . . . 57
5.2 New Contributions . . . 57
5.2.1 Timed Sequences . . . 57
5.2.2 Resource Threshold . . . 59
5.2.3 Concurrency Threshold . . . 62
5.2.4 Probabilistic Sequences . . . 66
5.2.5 Expected Execution Time . . . 67
5.3 Related Work . . . 75
5.4 Open Problems . . . 76

Bibliography 79

I First Author Publications 91
A An SMT-based Approach to Fair Termination Analysis. FMCAD, 2015. . 93
B Computing the Concurrency Threshold of Sound Free-Choice Workflow Nets. TACAS, 2018. . . . 103
C Computing the Expected Execution Time of Probabilistic Workflow Nets. TACAS, 2019. . . . 121

II Non-first Author Publications 141
D An SMT-based Approach to Coverability Analysis. CAV, 2014. . . . 143
E Towards Efficient Verification of Population Protocols. PODC, 2017. . . 161

III Note on Copyrights 171

List of Figures and Tables

2.1 Lamport's 1-bit algorithm for mutual exclusion [Lam86]. 13

3.1 Safe instances that were successfully proved safe by PETRINIZER and other tools. 23
3.2 Mean and median execution times in seconds for all configurations of PETRINIZER, IIC, BFC and MIST 24
3.3 Graphs comparing size of invariants and execution times in seconds for IIC and different configurations of PETRINIZER. 25
3.4 Graphs comparing execution times in seconds for IIC, BFC, MIST and two configurations of PETRINIZER. 25
3.5 P-component and subnet of the Petri net for Lamport's algorithm. 29
3.6 Terminating systems for which refinement is insufficient. 30
3.7 Execution time of PETRINIZER in dependence on the number of place for the benchmark suites IBM, SAP, Erlang and Literature. 34
3.8 Fairly terminating instances with rate of success by PETRINIZER. 34
3.9 Comparison of refinement for liveness with and without minimization and comparison of execution time with SPIN on the asynchronous suite. 35

4.1 Majority protocol shown in form of a Petri net. 39
4.2 Results of the experimental evaluation for PEREGRINE 49

5.1 A workflow net for processing an order with quantitative information. . 53
5.2 A timed probabilistic workflow net (TPWN) 56
5.3 Process and schedule for a run of the timed workflow net in Figure 5.2. . 58
5.4 A TWN without an online $(3,5)$-schedule and $(3,5)$-schedules for two of its runs. 61
5.5 Sound and free-choice TWNs where the constraints for the concurrency threshold are inexact. 64
5.6 Statistics on the size of nets and analysis time of the concurrency threshold. 66
5.7 $MDP_{\mathcal{W}}$ for the TPWN $\mathcal{W}$ in Figure 5.2. 68
5.8 Two Markov chains for the "earliest-first" scheduler. 70
5.9 A PERT network and its corresponding timed probabilistic workflow net. 72
5.10 Results for computing the expected execution time of the IBM suite. . . . 73
5.11 Results for computing the expected execution time of the BPI net. 74

1 Introduction

> *The need for correctness proofs is especially great with multiprocess programs. The asynchronous execution of several processes leads to an enormous number of possible execution sequences, and makes exhaustive testing impossible. A multiprocess program which has not been proved to be correct will probably have subtle errors, resulting in occasional mysterious program failures.*
>
> **— Leslie Lamport**
> Proving the Correctness of Multiprocess Programs [Lam77]
> © 1977 IEEE

At least since the rise of distributed and parallel computing, the theoretical study of distributed systems, which are systems with multiple components distributed over different locations, has become highly relevant. The components in distributed systems may be agents, machines, processes, threads or other entities that have a local state and follow some procedure or protocol. All components may communicate with each other synchronously or asynchronously, e.g. through pairwise interactions, message passing or shared variables, and are executed concurrently, which defines the joint distributed executions. Examples of distributed systems are mutual exclusion algorithms controlling access to shared resources, computer networks for distributed computing, or multi-agent environments working towards a common goal.

Distributed systems are becoming increasingly larger and more complex due to higher demand, more sophisticated algorithms and cheaper access to computational resources. Because of this, ensuring their correctness is also becoming increasingly important at the same time. For example, a fault in a safety-critical wireless sensor network for monitoring industrial plants could endanger lives, or unforeseen delays in a distributed business process of a bank could incur high costs. While single components of a distributed system are usually easy to understand, the behavior of the whole system can become very hard to grasp due to the large number of possible different interleavings of executions, also known as the state explosion problem [Val96]. This makes distributed systems hard to design and reason about. Therefore, to ensure correctness of such systems and verify its desired properties, automatic analysis is a vital tool. With appropriate formalisms for these systems, techniques from computer-aided verification such as model-checkers can be used.

A well-established and popular model for the analysis of distributed systems are Petri nets. They are used for verification of concurrent programs [ABQ11; DKO13; DRB02; GM12; KKW10; KKW14], analysis of biological or chemical processes as well as sensor networks in the form of *population protocols* [Ang+06; Che+17; Esp+17; NB15], or analysis of business processes in the form of *workflow nets* [Aal03; Aal97; Aal98; EHS17]. Petri nets can express systems with an infinite state-space and unbounded counters — under certain restrictions on the distinguishability of agents or processes and operations allowed on unbounded counters. This lends them considerable expressive power, but not too much. Many decision problems for Petri nets are still decidable, unlike for other Turing-complete formalisms of distributed systems. For certain extensions of Petri nets some problems also remain decidable, but others then become undecidable or still have an unknown decidability status [Aks+17; LT17].

However, decidability comes at an astronomically high cost in terms of complexity: for a Petri net model of a concurrent program, deciding thread-state reachability, known as coverability, is EXPSPACE-complete [Lip76; Rac78], and deciding reachability of a global configuration is TOWER-hard [Cze+19]. This means that any complete algorithm for these problems needs an exponential or even non-elementary amount of memory in the worst case. Similar lower bounds hold for other problems, e.g. verification of liveness properties.

To overcome these crippling complexities, one can usually choose two approaches. The first approach is using *incomplete* decision procedures. Those correctly decide whether a given property holds when returning a positive or negative result, but may also return an inconclusive result, in which case one does not know whether the property holds or not. Such procedures may have a much lower worst-case complexity, but often still work on a large number of systems encountered in practice. The second approach is restricting the analysis to certain *subclasses* of Petri nets. Those subclasses should be easy to recognize, e.g. a syntactic restriction, and have a decision problem with a lower complexity than in the general case, without sacrificing too much expressive power to allow modeling of many systems from the application domain. For example, the reachability problem, while being TOWER-hard for general Petri nets, is PSPACE-complete for 1-safe nets [CEP95], NP-complete for live 1-safe free-choice nets [Esp98], and in P for cyclic live 1-safe free-choice-nets [DE95]. The last subclass corresponds to a very common class of workflow nets, called *sound free-choice* workflow nets [Aal97].

In this thesis, we follow both approaches. In Chapter 3, we develop incomplete decision procedures for the verification of *safety* and *liveness* properties of general Petri nets, two essential classes of properties for program verification [Lam77] where the decision problems are EXPSPACE-hard. In Chapter 4, we adapt our methods for verification of population protocols, a model closely related to Petri nets which has gained a lot interest from the research community since their introduction in 2004 [Ang+04].

Here, we focus on verifying *correctness*, which is TOWER-hard [Cze+19; Esp+17], and also develop an incomplete decision procedure for this problem.

All our procedures are based on constructing a set of *constraints* for a Petri net, which are formulas in a logic with a simple satisfiability check in P or NP. From one of the results for satisfiability, we can then infer whether the property holds. One advantage of the constraint-based approach is that one can employ an independent and highly optimized solver for these constraints, e.g. an SMT solver, which directly leads to a practical implementation. Another advantage is that the constraints can be used as a *certificate* to show that the property holds, and the certificate can often even be transformed into an intuitive form explaining why the property holds. Such constraint-based procedures for safety and liveness properties have been employed successfully in the past [EM00; EM97; WW12]. We extend these procedures with novel refinement methods that allow them to verify more systems and give an extensive experimental evaluation. We also give the first procedure to practically verify correctness of population protocols, a problem which has only recently been shown to be decidable [Esp+17].

Further, we analyze workflow nets, a form of Petri nets commonly used to analyze business processes, and specifically consider the subclass of sound free-choice workflow nets. Reachability is polynomial for this class, and recent results give polynomial algorithms for some *quantitative* properties [EHS17]. We analyze quantitative properties concerning the necessary amount of resources and execution time in workflow nets, for which complexity or decidability questions were still open [BVT16]. Verification of quantitative properties is often harder than the corresponding qualitative decision problems. For example, in a probabilistic system, one could ask if the system will never reach the final state (safety), if it will always reach the final state (liveness), or what the expected number of steps is until reaching the final state (quantitative). If the quantitative problem has a finite value, then the answer to the liveness problem is yes, but the answer does not give any information about the value itself.

In Chapter 5, we show that for workflow nets, several quantitative properties cannot be computed in polynomial time unless $\mathsf{P} = \mathsf{NP}$, in contrast to related decision problems or similar quantitative properties with polynomial-time algorithms. We also give efficient constraint-based procedures for these quantitative properties, which can either compute the exact value or, with a lower complexity, an approximation that we show to be good enough in practice.

1.1 Related Work

We give an overview on the related work that introduces the models we consider in this thesis. A discussion of work related to our methods and contributions is given

in Section 3.4 of Chapter 3 for Petri nets, Section 4.3 of Chapter 4 for population protocols, and Section 5.3 of Chapter 5 for workflow nets.

Petri nets are a well-established model for distributed systems, originally introduced by Carl Adam Petri [Pet62]. Since then, the concepts and theory of Petri nets have developed considerably. We refer to [Rei13] for a modern introduction. A related model often used instead of Petri nets are Vector Addition Systems with States (VASS) [HP79], which is equivalent to Petri nets [Reu90] for all purposes considered in this thesis.

Population protocols were introduced by Angluin et al. [Ang+04; Ang+06] as a model of computations in networks of passively mobile finite-state sensors. An analysis of their computational power followed in [AAE06; Ang+07]. We refer to [AR09] for an introduction to the basic model of population protocols as well as information on the computational power of related models. In [Esp+15; Esp+17] population protocols are associated with Petri nets by giving reductions between both models for the central problems of reachability and well-specification, yielding new decidability results.

Workflow nets were introduced by Wil van der Aalst [Aal97; Aal98] as a subclass of Petri nets to model and analyze business processes. We refer to [AH02] for a comprehensive overview. There has been a large amount of work on application of techniques from Petri nets to workflow nets, e.g. to analyze the central correctness notion of soundness, although the specific class of workflow nets as well the notion of soundness considered often differs. We refer to [Aal+11] for a large list of existing work and a discussion on different models and soundness properties. More recent work [DE16; EHS17] associates a specific class of workflow nets to Petri nets through the model of negotiations, which produces techniques for the analysis of some quantitative properties.

1.2 Main Contributions

We briefly summarize the main contributions of this thesis. A more detailed and technical presentation is given in Chapters 3 to 5.

Constraint-based analysis methods for Petri nets

In [EM15; Esp+14], we consider safety and liveness properties of distributed systems that reduce to the reachability and the fair termination problem for Petri nets. We develop incomplete decision procedures for these problems, based on constraints derived from structural invariants of the net. These procedures have a low complexity, being either polynomial, in NP or co-NP, in contrast to the EXPSPACE-hardness of the general problems, and can be implemented efficiently by using an SMT solver. We also show how to construct certificates from the constraints that can be used to easily verify the given property indepently.

In an extensive experimental evaluation we show that our procedures can be used to verify safety and liveness properties for a large set of distributed systems from practical applications. Further, our procedures are often several orders of magnitude faster than other existing methods, and are able to produce smaller invariants in many cases.

Verification of population protocols

In [Blo+17], we consider the well-specification and correctness problems for population protocols. Both problems are as hard as the reachability problem for Petri nets. We develop incomplete decision procedures for these problems with a lower complexity, based on constraints solvable by an SMT solver. We also provide the first implementation for fully verifying population protocols using these procedures.

We show that our procedures are complete for an expressive subclass of population protocols, and show in an experimental evaluation that they can be used to verify many protocols from the literature.

Quantitative analysis of workflow nets

In [MEO19a; MEV18a], we analyze workflow nets extended with quantitative information about time and probabilities. We consider the problems of computing the resource or concurrency threshold, two measures that quantify the number of resources needed to execute the net as fast as possible, and the problem of computing the expected execution time of a workflow net.

We show that the problems of deciding if the resource or concurrency threshold exceeds a given bound are both NP-hard, and give results on which amount of resources allows online schedules. We give an NP algorithm based on linear constraints to approximate the concurrency threshold that is exact for a large subclass of workflow nets. Further, we show that computing the expected execution time is #P-hard and give the first decidability result by an exponential-time algorithm. All hardness results hold for rather restrictive classes of workflow nets and imply that no polynomial-time algorithm exists unless $\mathsf{P} = \mathsf{NP}$. In contrast, polynomial-time algorithms exist for many other common problems on these classes, e.g. reachability or computing the expected cost.

In an experimental evaluation on a large standard suite of workflow nets, we show that our algorithm to approximate the concurrency threshold always gives the exact result within milliseconds and is faster than using state-space exploration. We further show that we can compute the expected execution time within milliseconds for this suite of nets and suitably approximate it within minutes for a more complex net.

1.3 Publication Summary

We list the publications by the thesis author which we discuss in this publication-based thesis. All papers are included included in the Appendix. In Part I of the Appendix, we present the following papers where the thesis author is first author of the paper:

A Javier Esparza and Philipp J. Meyer. An SMT-based Approach to Fair Termination Analysis. FMCAD, 2015. [EM15]

B Philipp J. Meyer, Javier Esparza and Hagen Völzer. Computing the Concurrency Threshold of Sound Free-Choice Workflow Nets. TACAS, 2018. [MEV18a]

C Philipp J. Meyer, Javier Esparza and Philip Offtermatt. Computing the Expected Execution Time of Probabilistic Workflow Nets. TACAS, 2019. [MEO19a]

In Part II of the Appendix, we present the following papers where the thesis author is not first author of the papers:

D Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp Meyer and Filip Niksic. An SMT-based Approach to Coverability Analysis. CAV, 2014. [Esp+14]

E Michael Blondin, Javier Esparza, Stefan Jaax and Philipp J. Meyer. Towards Efficient Verification of Population Protocols. PODC, 2017. [Blo+17]

All papers are preceded by a page listing the full citation, a short summary and the contributions of the thesis author. In Part III of the Appendix, we give a note on the copyrights for the included papers. Content-wise, the papers relate to the chapters of this thesis as follows: Papers D and A are discussed in Chapter 3, Paper E in Chapter 4, and Papers B and C in Chapter 5.

Besides the included papers, while working on this thesis the author has also co-authored the following papers that appeared in peer-reviewed conference proceedings or journals. These are not part of this thesis, but given for completeness.

Papers on population protocols

- Michael Blondin, Javier Esparza, Martin Helfrich, Antonín Kučera and Philipp J. Meyer. Checking Qualitative Liveness Properties of Replicated Systems with Stochastic Scheduling. CAV, 2020. [Blo+20b]
- Javier Esparza, Martin Helfrich, Stefan Jaax and Philipp J. Meyer. Peregrine 2.0: Explaining Correctness of Population Protocols Through Stage Graphs. ATVA, 2020. [Esp+20]

Papers on games and synthesis

- Philipp J. Meyer and Michael Luttenberger. Solving Mean-Payoff Games on the GPU. ATVA, 2016. [ML16]
- Philipp J. Meyer, Salomon Sickert and Michael Luttenberger. Strix: Explicit Reactive Synthesis Strikes Back! CAV, 2018. [MSL18]
- Michael Luttenberger, Philipp J. Meyer and Salomon Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. Acta Informatica, 2020. [LMS20]

1.4 Outline of the Thesis

In Chapter 2, we introduce mathematical notation, the notion of constraints and the fundamental model of Petri nets, while giving existing complexity results for classical problems.

In Chapter 3, we present the results of [EM15; Esp+14] for constraint-based analysis of safety and liveness properties of Petri nets. In Chapter 4, we present the results of [Blo+17] for verification of the correctness and well-specification problems for population protocols. In Chapter 5, we present the results of [MEO19a; MEV18a] for the quantitative analysis of the resource threshold, concurrency threshold and expected execution time of workflow nets.

Besides presenting the main results of the included papers, Chapters 3 to 5 each contain a short introduction for the respective formal model, a section on related work and a section on open problems.

2 Preliminaries

2.1 Basic Notation

Numbers, sets and functions

We denote by $\mathbb{B}$ the set $\{0,1\}$ of Boolean values, by $\mathbb{N}$ the set $\{0,1,2,\ldots\}$ of non-negative integers, by $\mathbb{Z}$ the set $\{\ldots,-2,-1,0,1,2,\ldots\}$ of integers and by $\mathbb{Q}$ the set of rational numbers. As usual, $\cdot, +, -$ denote the canonical operations and $\leq, \geq$ the linear order on $\mathbb{Q}$. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1,2,\ldots,n\}$, and by $[\omega]$ the set $\{1,2,\ldots\}$ of positive integers.

For a set A, we denote by 2^A the set of all subsets of A. For two sets A, B, we denote by B^A the set of all (total) functions $f : A \to B$. We often implicitly change the codomain of functions, e.g. consider a function $f : A \to \mathbb{N}$ also as a function $f : A \to \mathbb{Q}$. For two functions $f : A \to B$ and $g : B \to C$, we denote by $g \circ f$ the function $g \circ f : A \to C$ with $(g \circ f)(a) \stackrel{\text{def}}{=} g(f(a))$ for all $a \in A$. For a function $f : A \to B$, we denote by f^{-1} the function $f^{-1} : B \to 2^A$ with $f^{-1}(b) \stackrel{\text{def}}{=} \{a \in A \mid f(a) = b\}$.

Vectors and matrices

Let $K \in \{\mathbb{B}, \mathbb{N}, \mathbb{Z}, \mathbb{Q}\}$. A *vector* is a function $\mathbf{x} : A \to K$ for a finite set A. If K is $\mathbb{B}$, $\mathbb{Z}$ or $\mathbb{Q}$, we also talk about Boolean, integer or rational vectors, respectively. Let $\mathbf{x}, \mathbf{y} \in K^A$ be vectors. We use the vector space $\mathbb{Q}^A$ to define scalar multiples, addition and inverses of vectors, i.e. have $(c \cdot \mathbf{x})(a) \stackrel{\text{def}}{=} c \cdot \mathbf{x}(a)$, $(\mathbf{x} + \mathbf{y})(a) \stackrel{\text{def}}{=} \mathbf{x}(a) + \mathbf{y}(a)$ and $(-\mathbf{x})(a) \stackrel{\text{def}}{=} -(\mathbf{x}(a))$ for $c \in \mathbb{Q}$ and $a \in A$. The partial orders $\leq, \geq$ on vectors are defined by component-wise lifting of the respective linear orders, i.e. $\mathbf{x} \leq \mathbf{y}$ if $\mathbf{x}(a) \leq \mathbf{y}(a)$ for all $a \in A$ and $\mathbf{x} \geq \mathbf{y}$ if $\mathbf{x}(a) \geq \mathbf{y}(a)$ for all $a \in A$. The (scalar) product $\mathbf{x} \cdot \mathbf{y}$ is defined as $\sum_{a \in A} \mathbf{x}(a) \cdot \mathbf{y}(a)$. The *size* of $\mathbf{x}$ is the number $|\mathbf{x}| \stackrel{\text{def}}{=} \sum_{a \in A} |\mathbf{x}(a)|$ and the *support* of $\mathbf{x}$ is the set $[\![\mathbf{x}]\!] \stackrel{\text{def}}{=} \{a \in A \mid \mathbf{x}(a) \neq 0\}$. For a subset $A' \subseteq A$, we lift the application of $\mathbf{x}$ to A' by $\mathbf{x}(A') \stackrel{\text{def}}{=} \sum_{a \in A'} \mathbf{x}(a)$. For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{N}^A$, we also define the vector $\mathbf{x} \ominus \mathbf{y} \in \mathbb{N}^A$ by $(\mathbf{x} \ominus \mathbf{y})(a) \stackrel{\text{def}}{=} \max(\mathbf{x}(a) - \mathbf{y}(a), 0)$ for $a \in A$.

A *matrix* is a function $\mathbf{Z} : A \times B \to K$ for finite sets A, B. Let $\mathbf{Z} \in K^{A \times B}$ be a matrix and $\mathbf{x} \in K^A, \mathbf{y} \in K^B$ be vectors. The matrix-vector products $\mathbf{Z} \cdot \mathbf{y} \in \mathbb{Q}^A$ and $\mathbf{x} \cdot \mathbf{Z} \in \mathbb{Q}^B$ are defined as the vectors with $(\mathbf{Z} \cdot \mathbf{y})(a) \stackrel{\text{def}}{=} \sum_{b \in B} \mathbf{Z}(a,b) \cdot \mathbf{y}(b)$ for $a \in A$ and $(\mathbf{x} \cdot \mathbf{Z})(b) \stackrel{\text{def}}{=} \sum_{a \in A} \mathbf{x}(a) \cdot \mathbf{Z}(a,b)$ for $b \in B$. We generally use boldface lower-case roman letters for vectors and boldface upper-case roman letters for matrices to distinguish them.

If we have a fixed enumeration $a_1, a_2, \ldots, a_n$ of a set A, we use the tuple notation $(k_1, k_2, \ldots, k_n)$ to denote the vector $\mathbf{x} \in K^A$ with $\mathbf{x}(a_i) = k_i$ for $i \in [n]$. We also use the multiset notation $\wr a_1, a_2, \ldots, a_k \wr$ with $a_i \in A$ for $i \in [k]$ to denote the vector $\mathbf{x} \in \mathbb{N}^A$ with $\mathbf{x}(a) = |\{i \in [k] \mid a_i = a\}|$ for $a \in A$. For example, with the enumeration a, b, c of A, both $(1, 2, 0)$ and $\wr a, b, b \wr$ denote the vector $\mathbf{x}$ with $\mathbf{x}(a) = 1$, $\mathbf{x}(b) = 2$ and $\mathbf{x}(c) = 0$. We denote by $\mathbf{0}$ and $\mathbf{1}$ vectors with $\mathbf{0}(a) = 0$ and $\mathbf{1}(a) = 1$ for every element a in their domain.

Sequences

A *sequence* of a set A is a function $\sigma : [n] \to A$ with $n \in \mathbb{N} \cup \{\omega\}$, where the *length* of σ is $|\sigma| \stackrel{\text{def}}{=} n$. A sequence σ is *empty* if $|\sigma| = 0$, *finite* if $|\sigma| \in \mathbb{N}$ and *infinite* if $|\sigma| = \omega$. We denote by ϵ the empty sequence of length 0, and by $a_1 \cdot a_2 \cdot \ldots \cdot a_n$ the finite sequence σ of length n with $\sigma(i) = a_i$ for $i \in [n]$. For a set A, we denote by A^* the set of finite sequences of A and by A^ω the set of infinite sequences of A. For a non-empty finite sequence σ, we denote by σ^ω the infinite sequence with $\sigma^\omega(i) \stackrel{\text{def}}{=} \sigma(((i-1) \bmod |\sigma|)+1)$ for $i \in [\omega]$. Concatenation of finite sequences or a finite and an infinite sequence by the operator $\cdot$ is defined as usual.

The *support* of a sequence σ of A is the set $[\![\sigma]\!] \stackrel{\text{def}}{=} \{a \in A \mid \exists i \in [|\sigma|] : \sigma(i) = a\}$. The *Parikh vector* of a finite sequence σ of a finite set A is the vector $\Psi(\sigma) \in \mathbb{N}^A$ with $\Psi(\sigma)(a) \stackrel{\text{def}}{=} |\{i \in [|\sigma|] \mid \sigma(i) = a\}|$. For an infinite sequence σ, the set of elements occurring infinitely often is given by $\mathrm{Inf}(\sigma) \stackrel{\text{def}}{=} \{a \in A \mid \forall i \in [\omega] : \exists j \geq i : \sigma(j) = a\}$.

2.2 Constraints and Complexity

Predicates and constraints

A *predicate* over a set X is a function $\varphi : X \to \mathbb{B}$. For $x \in X$, $\varphi(x)$ *holds* if $\varphi(x) = 1$. A *constraint* over a set X is a predicate over X given by a concrete formula over X. Below we define the specific constraints used in this thesis. Let $K \in \{\mathbb{B}, \mathbb{N}, \mathbb{Z}, \mathbb{Q}\}$ and $A_1, \ldots, A_k$ be finite sets. Then a k-ary constraint $\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ given k vectors $\mathbf{x}_1 \in K^{A_1}, \ldots, \mathbf{x}_k \in K^{A_k}$ is

- a *linear constraint* if $\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_k) = \left(\sum_{i=1}^k \mathbf{A}_i \cdot \mathbf{x}_i \geq \mathbf{b}\right)$ for some matrices $\mathbf{A}_1 \in \mathbb{Z}^{B \times A_1}, \ldots, \mathbf{A}_k \in \mathbb{Z}^{B \times A_k}$ and a vector $\mathbf{b} \in \mathbb{Z}^B$ for some finite set B; and
- an (existential) *Presburger constraint* if it is of the form

$$\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_k) = \left(\exists \mathbf{y}_1, \ldots, \mathbf{y}_l : \psi(\mathbf{x}_1, \ldots, \mathbf{x}_k, \mathbf{y}_1, \ldots, \mathbf{y}_l)\right)$$

  where ψ is a Boolean formula with linear constraints as atoms, evaluated on the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_k$ and existentially quantified vectors $\mathbf{y}_1 \in K^{C_1}, \ldots, \mathbf{y}_l \in K^{C_l}$ for some finite sets $C_1, \ldots, C_l$.

Further we consider *Boolean constraints* over 2^A for some set A, which are simply given by a Boolean formula with elements of A as atoms, evaluated by set membership.

A predicate is a *Presburger predicate* if it can be given by a Presburger constraint. In the literature, Presburger predicates are often also specified using formulas in the first-order theory of the natural numbers with addition. As this theory admits quantifier elimination [Coo72; Pre29], every such formula is equivalent to an existential Presburger constraint in our definition. However note that the complexity for quantifier elimination is between double and triple exponential time [FR98; Opp78].

A set $S \subseteq \mathbb{N}^A$ is *semi-linear* if it is definable by a Presburger predicate, i.e. there is a Presburger predicate $\varphi : \mathbb{N}^A \to \mathbb{B}$ such that $S = \{\mathbf{x} \in \mathbb{N}^A \mid \varphi(\mathbf{x}) = 1\}$. We omit the usual definition of semi-linear sets in the literature by basis and period vectors and use this equivalent characterization [GS66], which may however change the size of a minimal representation.

Complexity of problems for constraints

For the complexity results below, we assume that constraints are given by an appropriate encoding of the structure of the Boolean formula, matrices and vectors, and all numbers are encoded in binary.

The *satisfiability problem* for a class of constraints is, given a constraint φ of that class over a set X, to decide whether there exists some $x \in X$ such that $\varphi(x)$ holds. For linear constraints where $K = \mathbb{Q}$, the satisfiability problem can be solved in polynomial time using linear programming [Kar84]. For linear constraints with $K \in \{\mathbb{B}, \mathbb{N}, \mathbb{Z}\}$, Boolean constraints and Presburger constraints, the satisfiability problem is NP-complete, which follows from NP-hardness for Boolean constraints [Coo71] and membership in NP for existential Presburger constraints [Pap81].

The *linear optimization problem* is, given a linear constraint φ over K^A and an objective vector $\mathbf{c} \in \mathbb{Q}^A$, either (i) decide that there is no $\mathbf{x}$ such that $\varphi(\mathbf{x})$ holds, (ii) compute a vector $\mathbf{x}$ such that $\varphi(\mathbf{x})$ holds and the value $\mathbf{c} \cdot \mathbf{x}$ is maximized among all $\mathbf{x}$ where $\varphi(\mathbf{x})$ holds, or (iii) decide that this value is unbounded. If $K = \mathbb{Q}$, the linear optimization problem is solvable in polynomial time [Kar84]. Otherwise for $K \in \{\mathbb{B}, \mathbb{N}, \mathbb{Z}\}$, NP-hardness already follows for the decision part from the satisfiability problem, and a solution can be computed in exponential time [Len83].

2.3 Petri Nets

A *Petri net* is a tuple $N = (P, T, F)$, where P is a finite set of *places*, T is a finite set of *transitions*, and $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the *flow function* assigning a weight to every pair of a place and a transition. We generally assume $P \cap T = \emptyset$. The elements $a \in (P \times T) \cup (T \times P)$ where $F(a) \neq 0$ are the *arcs* of N. For $x \in P \cup T$, we define the *preset* ${}^\bullet x$ and *postset* $x^\bullet$ as the vectors given by ${}^\bullet x(y) \stackrel{\text{def}}{=} F(y, x)$ and $x^\bullet(y) \stackrel{\text{def}}{=} F(x, y)$ for $y \in T$ if $x \in P$ or $y \in P$ if $x \in T$. We extend the notion of pre- and postset to sets $X \subseteq P$ or $X \subseteq T$ by ${}^\bullet X \stackrel{\text{def}}{=} \sum_{x \in X} {}^\bullet x$ and $X^\bullet \stackrel{\text{def}}{=} \sum_{x \in X} x^\bullet$. Further, for a transition t, we denote the *effect* of t by the vector $\Delta(t) \stackrel{\text{def}}{=} t^\bullet - {}^\bullet t$.

Let $N = (P, T, F)$ be a Petri net. A *marking* of N is a vector $\mathbf{m} \in \mathbb{N}^P$, and *marks* the places $[\![\mathbf{m}]\!]$. A transition $t \in T$ is *enabled* at a marking $\mathbf{m}$ if $\mathbf{m} \geq {}^\bullet t$, and t *leads* to a marking $\mathbf{m}'$ from $\mathbf{m}$ if t is enabled at $\mathbf{m}$ and $\mathbf{m}' = \mathbf{m} + \Delta(t)$. A (finite or infinite) *transition sequence* of N is a sequence $\sigma \in T^* \cup T^\omega$. A transition sequence σ is enabled at a marking $\mathbf{m}_0$ if there are markings $\mathbf{m}_i$ for each $i \in [|\sigma|]$ such that $\sigma(i)$ leads to $\mathbf{m}_i$ from $\mathbf{m}_{i-1}$ for all $i \in [|\sigma|]$, in which case σ leads to $\mathbf{m}_{|\sigma|}$ from $\mathbf{m}_0$ if σ is finite. We use the following notations for markings $\mathbf{m}$ and $\mathbf{m}'$:

- $\mathbf{m} \xrightarrow{t} \mathbf{m}'$ with $t \in T$ if t leads to $\mathbf{m}'$ from $\mathbf{m}$.
- $\mathbf{m} \to \mathbf{m}'$ if $\mathbf{m} \xrightarrow{t} \mathbf{m}'$ for some $t \in T$.
- $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$ with $\sigma \in T^*$ if σ leads to $\mathbf{m}'$ from $\mathbf{m}$.
- $\mathbf{m} \xrightarrow{*}_U \mathbf{m}'$ with $U \subseteq T$ if $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$ for some $\sigma \in U^*$.
- $\mathbf{m} \xrightarrow{*} \mathbf{m}'$ if $\mathbf{m} \xrightarrow{*}_T \mathbf{m}'$.

If $\mathbf{m} \xrightarrow{*} \mathbf{m}'$, then $\mathbf{m}'$ is *reachable* from $\mathbf{m}$. For a Petri net N and a marking $\mathbf{m}$, we denote by $\mathcal{R}_N(\mathbf{m}) \stackrel{\text{def}}{=} \{\mathbf{m}' \mid \mathbf{m} \xrightarrow{*} \mathbf{m}'\}$ the set of markings reachable from $\mathbf{m}$ in N. A *system* is a tuple $\mathcal{S} = (N, \mathbf{m_0})$, where N is a Petri net and $\mathbf{m_0}$ is a marking of N. Let $\mathcal{S} = (N, \mathbf{m_0})$ be a system. The set of reachable markings of $\mathcal{S}$ is the set $\mathcal{R}_\mathcal{S} \stackrel{\text{def}}{=} \mathcal{R}_N(\mathbf{m_0})$. A transition sequence of $\mathcal{S}$ is a transition sequence of N enabled at $\mathbf{m_0}$.

We depict Petri nets using circles for places, rectangles for transitions, and show arcs using directed arrows between places and transitions, where only weights greater than 1 are shown. Arcs in both directions are also shown as double-ended arrows. Markings are shown using tokens inscribed into places. See Figure 2.1b for an example.

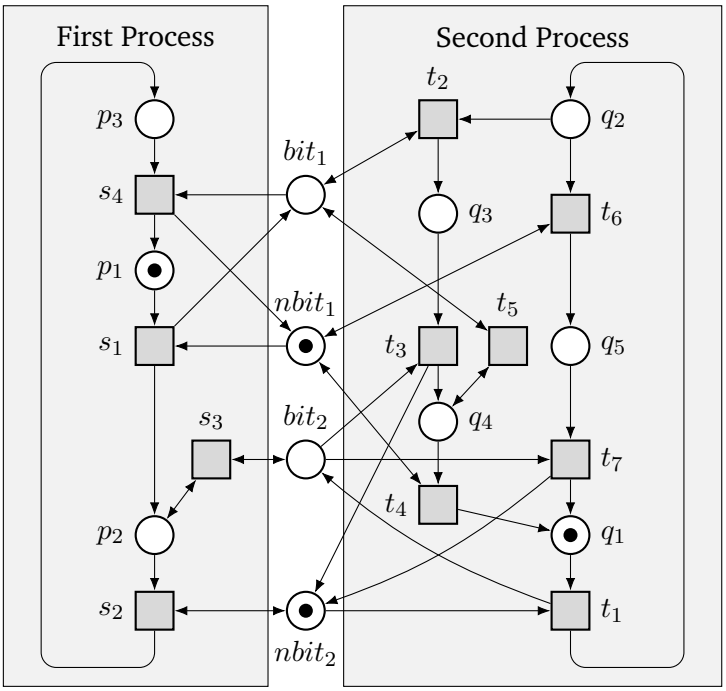
Subnets

A *subnet* of a Petri net (P, T, F) is a tuple (P', T', F') such that $P' \subseteq P$, $T' \subseteq T$ and $F' : (P' \times T') \cup (T' \times P') \to \mathbb{N}$ is a function with $F'(a) = F(a)$ for all $a \in (P' \times T') \cup (T' \times P')$. Since F' is completely defined by P' and T', we specify subnets by simply giving the tuple (P', T').

```
                    procedure PROCESS 1
                        bit1 ← false
                        while true do
p1 :                        bit1 ← true
p2 :                        while bit2 do
                                skip
                            end while
p3 :                        (∗ critical section ∗)
                            bit1 ← false
                        end while
                    end procedure

                    procedure PROCESS 2
                        bit2 ← false
                        while true do
q1 :                        bit2 ← true
q2 :                        if bit1 then
q3 :                            bit2 ← false
q4 :                            while bit1 do
                                    skip
                                end while
                            end if
q5 :                        (∗ critical section ∗)
                            bit2 ← false
                        end while
                    end procedure
```

(a) Pseudo-code with partially annotated process locations.



(b) Petri net model with initial marking.

Fig. 2.1 Lamport's 1-bit algorithm for mutual exclusion [Lam86].

Example

Figure 2.1b shows a Petri net model of Lamport's 1-bit mutual exclusion algorithm in Figure 2.1a, together with an initial marking for the initial state of the algorithm to obtain a system. The places p_1, p_2, p_3 represent the location of the first process, the places q_1, q_2, q_3, q_4, q_5 the location of the second process, and the remaining places the value of the shared variables bit_1 and bit_2. The algorithm should ensure that the two processes are never in their respective critical sections at p_3 and q_5 simultaneously.

Decision Problems

We consider the following decision problems for Petri nets, where we always assume that we are also given a system $(N, \mathbf{m_0})$ as part of the input:

- *Reachability*: given a marking $\mathbf{m}$, is $\mathbf{m}$ reachable from $\mathbf{m_0}$?
- *Coverability*: given a marking $\mathbf{m}$, is there a marking $\mathbf{m}'$ reachable from $\mathbf{m_0}$ such that $\mathbf{m}' \geq \mathbf{m}$?
- *General reachability*: given a Presburger constraint φ over the set of markings of N, is there a marking $\mathbf{m}$ reachable from $\mathbf{m_0}$ such that $\varphi(\mathbf{m})$ holds?
- *Termination*: is there no infinite transition sequence enabled at $\mathbf{m_0}$?
- *Fair termination*: given a Boolean constraint φ over sets of transitions, does $\varphi(\mathrm{Inf}(\sigma))$ hold for every infinite transition sequence σ enabled at $\mathbf{m_0}$?

Note that coverability and reachability are specific instances of general reachability, and termination is a specific instance of fair termination. Moreover, termination implies fair termination and reachability of a marking $\mathbf{m}$ implies coverability of $\mathbf{m}$.

We restrict ourselves here to fairness constraints that only constrain the set of transitions that occur infinitely often. Often also other fairness constraints are considered, which are usually specified in the problem but not part of the problem input. Examples are transition-based *strong fairness* or *weak fairness* [AFK88], which not only consider if transitions occur infinitely often in a sequence, but also if they are enabled infinitely often or eventually always enabled along the corresponding sequence of markings. In Chapter 4, we will consider yet another type of fairness constraint.

Complexity of the decision problems

The complexity of the previously introduced classical decision problems for Petri nets has been studied extensively, and we give the best known results below. Note that the exact complexity for reachability is still open.

- Reachability: TOWER-hard [Cze+19] and in ACKERMANN [LS19].
- Coverability: EXPSPACE-complete [Lip76; Rac78].
- General reachability: TOWER-hard and in ACKERMANN, as it can be reduced to the reachability problem [Hac76] using elementary space by constructing a semi-linear set representation of the solutions of φ [CH16; CH17].
- Termination: EXPSPACE-complete [AH11; Lip76; Yen92].
- Fair termination: EXPSPACE-complete [AH11; Yen92].

We refer to [Sch16] for a description of the complexity classes TOWER and ACKERMANN. The TOWER-hardness result for reachability shows that it is a non-elementary problem, and thus harder than the EXPSPACE-complete problems.

3 Constraint-based Petri Net Analysis

Safety and liveness properties of distributed systems can often be modeled as corresponding reachability and termination problems for Petri nets. This makes Petri nets a suitable choice for the analysis of distributed systems and to verify their correct behavior. However, due to the high complexity of the reachability and termination problems (TOWER- and EXPSPACE-hard, respectively), complete decision procedures are often impractical. Therefore one would like to derive procedures that have a lower complexity and an efficient implementation, but which may only be a decision procedure for a subset of the problem. This is also known as an incomplete (but sound) procedure: A positive answer to an instance of the problem then means that the property holds, while a negative answer yields no information if the property holds or not. However, such a procedure may give a positive result for a sufficiently large set of systems and properties encountered in practice and thus make the procedure feasible for practical problems. Furthermore, an incomplete procedure can be used as a preprocessing step of another complete method to make it more efficient on average.

Safety properties

A *safety property* of a system $(N, \mathbf{m_0})$ is a predicate over the set of markings of N. A system satisfies a safety property φ if φ holds for all markings reachable from $\mathbf{m_0}$. Dually, the system does not satisfy φ if there is a reachable marking where $\neg\varphi$ holds. Therefore verification of safety properties is reducible to the general reachability problem. For the common class of downward-closed safety properties it also relates to the coverability problem, as in the following example.

Consider the mutual exclusion algorithm in Figure 2.1a and the corresponding system with a Petri net model in Figure 2.1b. The safety property here is the the mutual exclusion property, i.e. that the two processes are never in the critical section simultaneously. We can specify this using $\varphi(\mathbf{m}) = (\mathbf{m}(p_3) = 0 \vee \mathbf{m}(q_5) = 0)$. For the dual general reachability problem we have $\neg\varphi(\mathbf{m}) = (\mathbf{m}(p_3) \geq 1 \wedge \mathbf{m}(q_5) \geq 1)$, which also translates to a coverability problem with the marking $\mathbf{m} = \{\!|p_3, q_5|\!\}$.

Liveness properties

We restrict us to liveness properties that can be given as a fair termination problem. Then, a *liveness property* of a system $(N, \mathbf{m_0})$ is a Boolean constraint over sets of transitions of N. A system satisfies a liveness property φ if $\varphi(\mathrm{Inf}(\sigma))$ holds for every infinite transition sequence σ enabled at $\mathbf{m_0}$. By taking the product of the Petri net

with a suitable Büchi automaton, this approach can be extended to verify transition-based LTL properties of systems [EM97]. Note that the simple termination problem can also be reduced to the liveness property $\varphi = \mathit{false}$.

As an example, for Lamport's mutual exclusion algorithm, we consider the liveness property that the first process enters its respective critical section infinitely often under a fair scheduler where both processes are executed infinitely often. On the Petri net model in Figure 2.1b, this can be expressed by the fair termination property φ with the following Boolean constraint:

$$\varphi = \left(\left(\bigvee_{i=1}^{4} s_i \right) \wedge \left(\bigvee_{i=1}^{7} t_i \right) \right) \rightarrow s_2 \tag{3.1}$$

The left side of the implication is the fairness assumption that states that both processes are executed infinitely often, and the right side is the liveness property that the first process enters its critical section infinitely often. Note that the Petri net model is designed so that in every reachable marking, a transition of each process is enabled, so every finite execution can be extended to an infinite execution satisfying the fairness assumption.

3.1 Problem Statement

We want to verify safety and liveness properties of distributed system which translate to the coverability, general reachability or fair termination problems of Petri nets. For this, we first investigate how structural invariants known from Petri net theory can be used to derive incomplete decision procedures for these properties with a lower complexity.

Second, we investigate how a constraint-based representation can be used to implement the procedure by utilizing existing solvers for these constraints. We also ask if and how such a representation can be used to give certificates that show that the given property holds.

Third, we want to evaluate the performance and the degree of completeness of our methods, i.e. the number of instances on which they succeed, on a large set of Petri nets stemming from the literature and distributed systems from practical applications.

3.2 New Contributions for Safety Properties

In [Esp+14] we present an incomplete decision procedure for verifying safety properties, based on the construction of a set of constraints solvable by an SMT solver. Here, we present a slightly generalized version of this procedure which lays the foundation for the methods used in [Blo+17], presented in Chapter 4. We note how the original

procedure in [Esp+14] can be recovered. Further, if we consider the coverability problem, we show in [Esp+14] how inductive invariants can be derived as a certificate to show that a given safety property is satisfied.

3.2.1 Marking Equation

We start by defining a simple overapproximation of the reachability relation in form of a linear equation. This equation counts the total effect of a transition sequence, but does not consider order or enabledness of transitions.

Definition 3.2 (Incidence matrix)**.** Let $N = (P, T, F)$ be a Petri net. The *incidence matrix* of N is the matrix $\mathbf{C} \in \mathbb{Z}^{P \times T}$ with $\mathbf{C}(p,t) \stackrel{\text{def}}{=} F(t,p) - F(p,t)$.

Definition 3.3 (Marking equation)**.** Let $N = (P, T, F)$ be a Petri net and $\mathbf{m_0}, \mathbf{m}$ be markings of N. The *marking equation* for $N, \mathbf{m_0}, \mathbf{m}$ and a vector $\mathbf{x} \in \mathbb{N}^T$ (or $\mathbf{x} \in \mathbb{Q}^T$) is the equation $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \mathbf{x}$, where $\mathbf{C}$ is the incidence matrix of N.

The Parikh vector $\Psi(\sigma)$ counts how often each transition occurs in a transition sequence σ. The marking equation is an overapproximation of the reachability relation, as shown by the following lemma.

Lemma 3.4. *Let N be a Petri net, $\mathbf{m_0}, \mathbf{m}$ two markings of N and σ a transition sequence with $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$. Then $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \Psi(\sigma)$.*

Proof. As $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$, we have $\mathbf{m} = \mathbf{m_0} + \sum_{i=1}^{|\sigma|} \Delta(\sigma(i))$. By the definition of the incidence matrix, we have $\mathbf{C} \cdot \vec{t} = t^\bullet - {}^\bullet t = \Delta(t)$ for any transition t and therefore $\mathbf{C} \cdot \Psi(\sigma) = \sum_{i=1}^{|\sigma|} \mathbf{C} \cdot \vec{\sigma(i)} = \sum_{i=1}^{|\sigma|} \Delta(\sigma(i))$. □

It follows from Lemma 3.4 that if a marking $\mathbf{m}$ is reachable from a marking $\mathbf{m_0}$, then the marking equation of Definition 3.3 interpreted as a constraint is satisfiable. We use this to give a first incomplete decision procedure for verifying safety properties. Given a system $(N, \mathbf{m_0})$ and a safety property φ, we construct the following constraint over $\mathbf{x} \in K^T$ ($K \in \{\mathbb{Q}, \mathbb{Z}\}$) and a marking $\mathbf{m}$, where we can fix $\mathbf{m_0}$ as given by the system:

$$\text{MarkingEquation}_\varphi(\mathbf{m_0}, \mathbf{m}, \mathbf{x}) \stackrel{\text{def}}{=} \mathbf{m} \geq \mathbf{0} \wedge \mathbf{x} \geq \mathbf{0} \wedge \neg\varphi(\mathbf{m}) \wedge \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \mathbf{x} \quad (3.5)$$

If the marking constraint is not satisfiable, then no marking $\mathbf{m}$ satisfying $\neg\varphi$ is reachable from $\mathbf{m_0}$, and therefore the system satisfies φ.

If $K = \mathbb{Z}$, the marking constraint is a Presburger constraint for which the satisfiability problem is in NP and where we employ an SMT solver to solve the constraint. This method is called SAFETY/$\mathbb{Z}$. If $K = \mathbb{Q}$ and $\neg\varphi$ is a linear constraint, e.g. from a coverability problem, then we can express the whole constraint as a linear constraint and decide satisfiability over the rationals in polynomial time. This method is called

SAFETY/$\mathbb{Q}$. In general, SAFETY/$\mathbb{Q}$ is weaker than SAFETY/$\mathbb{Z}$, as the marking equation may be satisfiable over the rationals but not over the integers.

Both procedures so far are incomplete. Consider again the Petri net for Lamport's algorithm in Figure 2.1b with $\mathbf{m_0} = \wr p_1, q_1, nbit_1, nbit_2 \wr$ and the safety property $\varphi(\mathbf{m}) = (\mathbf{m}(p_3) = 0 \vee \mathbf{m}(q_5) = 0)$. The marking constraint (3.5) is satisfiable over the integers with $\mathbf{x} = \wr s_1, s_2, t_1, t_6 \wr$ and $\mathbf{m} = \wr p_3, q_5, bit_1, bit_2 \wr$. However, this $\mathbf{m}$ is actually not reachable, as we will show in the following sections.

3.2.2 Refinement with Traps and Siphons

To strengthen the marking equation, we introduce traps and siphons, two other structural invariants well known from Petri net theory, which characterize sets of places that "trap" tokens or which can be drained of tokens.

Definition 3.6 (Traps and Siphons)**.** Let $N = (P, T, F)$ be a Petri net and $U \subseteq T$. A subset of places $Q \subseteq P$ is a *U-trap* of N if $[\![Q^\bullet]\!] \cap U \subseteq [\![{}^\bullet Q]\!]$ and a *U-siphon* of N if $[\![{}^\bullet Q]\!] \cap U \subseteq [\![Q^\bullet]\!]$. If $U = T$ then Q is a *trap* or *siphon* of N, respectively. A U-trap or U-siphon Q is *proper* if $Q \neq \emptyset$.

Note that a U-trap (resp. U-siphon) of N is also a U'-trap (resp. U'-siphon) of N for any $U' \subseteq U$, therefore traps and siphons are the most general kind. U-traps have the property that they cannot be become unmarked by transitions in U, since every transition of U that removes a token of a U-trap also puts one token back into it. Dually, unmarked U-siphons cannot become marked by transitions in U. This is captured by the following lemma.

Lemma 3.7. *Let $N = (P, T, F)$ be a Petri net, $U \subseteq T$ and $\mathbf{m_0}, \mathbf{m}$ two markings of N with $\mathbf{m_0} \xrightarrow{*}_U \mathbf{m}$. Then for every U-trap Q of N, if $\mathbf{m}(Q) = 0$, then $[\![{}^\bullet Q]\!] \cap U = \emptyset$, and for every U-siphon Q of N, if $\mathbf{m_0}(Q) = 0$, then $[\![Q^\bullet]\!] \cap U = \emptyset$.*

Using Lemma 3.7, we can refine the marking constraint (3.5). First, for a given U-trap Q or U-siphon R, we define the following constraints over two markings $\mathbf{m_0}, \mathbf{m}$ and a vector $\mathbf{x}$ as in the marking equation:

$$\text{UTrap}_Q(\mathbf{m_0}, \mathbf{m}, \mathbf{x}) \stackrel{\text{def}}{=} (\mathbf{x}([\![Q^\bullet]\!] \setminus [\![{}^\bullet Q]\!]) = 0 \wedge \mathbf{m}(Q) = 0) \rightarrow \mathbf{x}([\![{}^\bullet Q]\!]) = 0 \tag{3.8}$$

$$\text{USiphon}_R(\mathbf{m_0}, \mathbf{m}, \mathbf{x}) \stackrel{\text{def}}{=} (\mathbf{x}([\![{}^\bullet R]\!] \setminus [\![R^\bullet]\!]) = 0 \wedge \mathbf{m_0}(R) = 0) \rightarrow \mathbf{x}([\![R^\bullet]\!]) = 0 \tag{3.9}$$

The first equation of each constraint essentially checks Definition 3.6 for $U = [\![\mathbf{x}]\!]$, and the second and third equation are the properties of Lemma 3.7. This full set of constraints is only employed in [Blo+17] and not yet used in [Esp+14]. There, only T-traps Q with $\mathbf{m_0}(Q) \geq 1$ are used, i.e. initially marked traps. In this case, the constraint (3.8) simplifies to $\mathbf{m}(Q) \geq 1$, since $[\![Q^\bullet]\!] \setminus [\![{}^\bullet Q]\!] = \emptyset$ and $\mathbf{m}(Q) = 0$ for some $\mathbf{m}, \mathbf{x}$ with $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \mathbf{x}$ would imply removal of a token in Q and thus $\mathbf{x}([\![{}^\bullet Q]\!]) \geq 1$.

Similarly, for a T-siphon R with $\mathbf{m_0}(R) = 0$, i.e. an initially unmarked siphon, the constraint (3.9) simplifies to $\mathbf{m}(R) = 0$.

Now assume we are given a system $(N, \mathbf{m_0})$, a safety property φ, and further a set $\mathcal{Q}$ of U-traps and a set $\mathcal{R}$ of U-siphons. We then construct the following constraint:

$$\text{MarkingEquation}_{\varphi}(\mathbf{m_0}, \mathbf{m}, \mathbf{x}) \wedge \bigwedge_{Q \in \mathcal{Q}} \text{UTrap}_{Q}(\mathbf{m_0}, \mathbf{m}, \mathbf{x}) \wedge \bigwedge_{R \in \mathcal{R}} \text{USiphon}_{R}(\mathbf{m_0}, \mathbf{m}, \mathbf{x}) \tag{3.10}$$

If the constraint (3.10) is unsatisfiable for some set of U-traps $\mathcal{Q}$ and U-siphons $\mathcal{R}$, then no marking $\mathbf{m}$ where $\neg\varphi$ holds is reachable, and therefore the safety property holds. Again, we can check satisfiability of the constraint over the integers as a Presburger constraint in NP using an SMT solver. Assuming that $\mathcal{Q}$ is a set of initially marked traps and $\mathcal{R}$ a set of initially unmarked siphons, we can use the simplified constraints for UTrap_Q and USiphon_Q and rewrite the constraint (3.10) as a linear constraint, and check satisfiability over the rationals in polynomial time.

3.2.3 Iterative Refinement for Safety

Constructing the constraint (3.10) with the sets of all of traps and siphons is generally infeasible: a Petri net may have an exponential number of both, even if only minimal ones (by set inclusion) are considered. However often only a small number of traps or siphons are necessary to prove that a given safety property holds. We therefore give an iterative approach, inspired by *counter-example guided abstraction refinement (CEGAR)* methods.

We initially set $\mathcal{Q} = \emptyset$ and $\mathcal{R} = \emptyset$. The procedure then iteratively checks satisfiability of the constraint (3.10) with the current sets $\mathcal{Q}$ and $\mathcal{R}$. If it is unsatisfiable, then the safety property holds, and we are done. Otherwise, we extract a solution $\mathbf{m_0}, \mathbf{m}, \mathbf{x}$ that satisfies the marking equation and all current trap and siphon constraints. We then look for a U-trap Q or U-siphon R such that the conditions of Lemma 3.7 are violated, i.e. one that shows that there is no sequence σ with $\Psi(\sigma) = \mathbf{x}$ such that $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$. If we can find such a Q or R, we add it to the respective set $\mathcal{Q}$ or $\mathcal{R}$, and repeat the process. Otherwise, the procedure fails and reports that the checks have been inconclusive.

It remains to give procedures to find appropriate U-traps and U-siphons for the given $\mathbf{m_0}, \mathbf{m}, \mathbf{x}$. We can find a $[\![\mathbf{x}]\!]$-trap Q unmarked at $\mathbf{m}$ with $\mathbf{x}([\![{}^\bullet Q]\!]) \geq 1$ or a $[\![\mathbf{x}]\!]$-siphon R unmarked at $\mathbf{m_0}$ with $\mathbf{x}([\![R^\bullet]\!]) \geq 1$ by computing the largest proper trap or siphon in the subnet $(S, [\![\mathbf{x}]\!])$ in polynomial time [DE95], where $S = [\![[\![\mathbf{x}]\!]^\bullet]\!] \setminus [\![\mathbf{m}]\!]$ for a trap and $S = [\![{}^\bullet[\![\mathbf{x}]\!]]\!] \setminus [\![\mathbf{m_0}]\!]$ for a siphon. However, larger traps and siphons are in general of poor quality for refinement, since they exclude fewer markings than minimal ones.

We therefore also use constraints to find traps and siphons. We construct constraints over the Boolean vectors $\mathbf{q}, \mathbf{r} \in \mathbb{B}^P$, encoding the trap $Q = [\![\mathbf{q}]\!]$ or siphon $R = [\![\mathbf{r}]\!]$. We first define the following auxiliary constraints for a U-trap or U-siphon marked at $\mathbf{m}$ and unmarked at $\mathbf{m}'$.

$$\mathrm{Trap}_{\mathbf{m},\mathbf{m}',U}(\mathbf{q}) \stackrel{\text{def}}{=} \bigwedge_{t \in U} ({}^\bullet t \cdot \mathbf{q} \geq 1 \to t^\bullet \cdot \mathbf{q} \geq 1) \wedge \mathbf{m} \cdot \mathbf{q} \geq 1 \wedge \mathbf{m}' \cdot \mathbf{q} = 0 \tag{3.11}$$

$$\mathrm{Siphon}_{\mathbf{m},\mathbf{m}',U}(\mathbf{r}) \stackrel{\text{def}}{=} \bigwedge_{t \in U} ({}^\bullet t \cdot \mathbf{r} \geq 1 \to t^\bullet \cdot \mathbf{r} \geq 1) \wedge \mathbf{m} \cdot \mathbf{q} \geq 1 \wedge \mathbf{m}' \cdot \mathbf{q} = 0 \tag{3.12}$$

Now for a given $\mathbf{m_0}, \mathbf{m}, \mathbf{x}$, we can check satisfiability of one of the following constraints to find an initially marked trap (3.13), an initially unmarked siphon (3.14), a U-trap (3.15) or a U-siphon (3.16) violating the properties of Lemma 3.7.

$$\mathrm{FindTrap}_{\mathbf{m_0},\mathbf{m}}(\mathbf{q}) \stackrel{\text{def}}{=} \mathrm{Trap}_{\mathbf{m_0},\mathbf{m},T}(\mathbf{q}) \tag{3.13}$$

$$\mathrm{FindSiphon}_{\mathbf{m_0},\mathbf{m}}(\mathbf{r}) \stackrel{\text{def}}{=} \mathrm{Siphon}_{\mathbf{m},\mathbf{m_0},T}(\mathbf{r}) \tag{3.14}$$

$$\mathrm{FindUTrap}_{\mathbf{m_0},\mathbf{m},\mathbf{x}}(\mathbf{q}) \stackrel{\text{def}}{=} \mathrm{Trap}_{[\![\mathbf{x}]\!]^\bullet,\mathbf{m},[\![\mathbf{x}]\!]}(\mathbf{q}) \tag{3.15}$$

$$\mathrm{FindUSiphon}_{\mathbf{m_0},\mathbf{m},\mathbf{x}}(\mathbf{r}) \stackrel{\text{def}}{=} \mathrm{Siphon}_{{}^\bullet[\![\mathbf{x}]\!],\mathbf{m_0},[\![\mathbf{x}]\!]}(\mathbf{r}) \tag{3.16}$$

As noted before, T-traps and T-siphons are of higher quality, since they exclude a set of markings independent of the vector $\mathbf{x}$ used to reach them. We therefore look for traps and siphons in the following order, stopping as soon as the first constraint is satisfiable, in which case we add the found element to $\mathcal{Q}$ or $\mathcal{R}$.

(1) Look for a trap Q with $\mathbf{m}(Q) = 0$ and $\mathbf{m_0}(Q) \geq 1$ using constraint (3.13).

(2) Look for a siphon R with $\mathbf{m_0}(R) = 0$ and $\mathbf{m}(R) \geq 1$ using constraint (3.14).

(3) Look for a $[\![\mathbf{x}]\!]$-trap Q with $\mathbf{m}(Q) = 0$ and $\mathbf{x}([\![{}^\bullet Q]\!]) \geq 1$ using constraint (3.15).

(4) Look for a $[\![\mathbf{x}]\!]$-siphon R with $\mathbf{m_0}(R) = 0$ and $\mathbf{x}([\![R^\bullet]\!]) \geq 1$ using constraint (3.16).

The constraints (3.13)–(3.16) are all Presburger constraints, and thus we can always use an SMT solver. The advantage over using the polynomial algorithm from [DE95] for the maximal trap or siphon is that we can instruct the SMT solver to find a minimal trap or siphon by iterative solving of the constraint. In practice, we experienced that the solver already tends to directly compute minimal or close-to-minimal solutions.

Overall, the main constraint (3.10) can be solved over the integers or the rationals. Depending on this, we obtain two versions of our iterative refinement procedure, called SafetyByRefinement/$\mathbb{Z}$ and SafetyByRefinement/$\mathbb{Q}$, respectively. In [Esp+14], these procedures only use the first refinement step (1) for initially marked traps with the constraint (3.13).

We return to our running example of Lamport's algorithm in Figure 2.1b for an illustration of the refinement method. As we saw before, for the safety property

$\varphi(\mathbf{m}) = (\mathbf{m}(p_3) = 0 \vee \mathbf{m}(q_5) = 0)$ the marking constraint is satisfiable with the markings $\mathbf{m_0} = \wr p_1, q_1, nbit_1, nbit_2 \wr$ and $\mathbf{m} = \wr p_3, q_5, bit_1, bit_2 \wr$ (we will not use $\mathbf{x}$). Constructing the constraint (3.13) for this $\mathbf{m_0}$ and $\mathbf{m}$, one obtains that it is satisfiable with the trap $Q = \{p_2, q_2, q_3, nbit_1, nbit_2\}$. We have $\mathbf{m_0}(Q) \geq 1$ and $\mathbf{m}(Q) = 0$, so the marking $\mathbf{m}$ is not reachable. After adding the corresponding refinement constraint $\mathbf{m}(Q) \geq 1$ to the marking constraint, it becomes unsatisfiable, so the mutual exclusion property holds for Lamport's algorithm.

Overall our methods are still incomplete, as there are systems with an unreachable marking where the marking equation is satisfiable and all four refinement methods may fail. We want to note that whenever the procedure fails, a marking $\mathbf{m}$ satisfying the constraint system is obtained. This marking is a potentially spurious counterexample, which can be checked using e.g. a model checker. If the marking is then actually reachable, it is a counterexample showing that the safety property does not hold.

3.2.4 Inductive Invariants for Safety

In [Esp+14], we present a method to derive inductive invariants after the procedure SafetyByRefinement succeeds to prove safety. This method is applicable if the following assumptions are met:

- The constraint (3.10) is unsatisfiable over the rationals for our sets $\mathcal{Q}$ and $\mathcal{R}$.
- We have only used the first refinement step using constraint (3.13), i.e. $\mathcal{Q}$ is a set of initially marked traps and $\mathcal{R} = \emptyset$.
- φ is a co-linear constraint, i.e. $\neg\varphi$ is a linear constraint with $\neg\varphi(\mathbf{m}) = \mathbf{A} \cdot \mathbf{m} \geq \mathbf{b}$ for a matrix $\mathbf{A} \in \mathbb{Z}^{B \times P}$, a vector $\mathbf{b} \in \mathbb{Z}^B$ and some set B.

Now assume that we have computed a set $\mathcal{Q}$ such that the above properties hold. Define the matrix $\mathbf{D} \in \mathbb{B}^{\mathcal{Q} \times P}$ with $\mathbf{D}(Q, p) = 1$ if $p \in Q$ and $\mathbf{D}(Q, p) = 0$ otherwise. Recall that for a trap Q with $\mathbf{m_0}(Q) \geq 1$, $\mathrm{UTrap}_Q(\mathbf{m_0}, \mathbf{m}, \mathbf{x})$ is equivalent to $\mathbf{m}(Q) \geq 1$. This then holds for all traps in $\mathcal{Q}$ if $\mathbf{D} \cdot \mathbf{m} \geq \mathbf{1}$. Then the constraints (3.10) can be rewritten as follows:

$$\mathbf{m} \geq \mathbf{0} \wedge \mathbf{x} \geq \mathbf{0} \wedge \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \mathbf{x} \quad \wedge \quad \mathbf{A} \cdot \mathbf{m} \geq \mathbf{b} \quad \wedge \quad \mathbf{D} \cdot \mathbf{m} \geq \mathbf{1} \tag{3.17}$$

Using Farkas' lemma [Sch86], we prove the following in [Esp+14]:

Theorem 3.18 ([Esp+14]). *The constraint (3.17) is unsatisfiable over the rational numbers if and only if the following constraint is satisfiable over the rational numbers.*

$$\lambda \cdot \mathbf{m_0} \leq \mathbf{y_1} \cdot \mathbf{b} + \mathbf{y_2} \cdot \mathbf{1} \quad \wedge \quad \lambda \geq \mathbf{y_1} \cdot \mathbf{A} + \mathbf{y_2} \cdot \mathbf{D} \quad \wedge \quad \mathbf{y_1} \geq \mathbf{0} \wedge \mathbf{y_2} \geq \mathbf{0} \tag{3.19}$$

This is a constraint over the vectors $\lambda \in \mathbb{Q}^P$, $\mathbf{y_1} \in \mathbb{Q}^B$ and $\mathbf{y_2} \in \mathbb{Q}^{\mathcal{Q}}$.

If the constraint (3.17) is unsatisfiable, then we can construct the following invariant $I : \mathbb{N}^P \to \mathbb{B}$ by using the λ from a solution to the constraint (3.19):

$$I(\mathbf{m}) \stackrel{\text{def}}{=} \mathbf{D} \cdot \mathbf{m} \geq \mathbf{1} \wedge \lambda \cdot \mathbf{m} \leq \lambda \cdot \mathbf{m_0} \tag{3.20}$$

We show in [Esp+14] that the invariant I has the following properties:

- *I holds initially*: $I(\mathbf{m_0})$ holds.
- *I is inductive*: for any markings $\mathbf{m}, \mathbf{m}'$ of N (reachable from $\mathbf{m_0}$ or not), if $I(\mathbf{m})$ holds and $\mathbf{m} \to \mathbf{m}'$, then $I(\mathbf{m}')$ holds.
- *I proves safety*: for any marking $\mathbf{m}$ of N, if $I(\mathbf{m})$ holds, then $\varphi(\mathbf{m})$ holds.

Therefore I is an inductive invariant that proves safety of the system $(N, \mathbf{m_0})$ with the property φ. As I is a linear constraint, and the properties above also hold for rational-valued $\mathbf{m} \in \mathbb{Q}^P$, they can be verified in polynomial time given I, $(N, \mathbf{m_0})$ and φ. While this also holds for unsatisfiability of the original constraint (3.10) from which we derived the primal system, the invariant I may be much smaller than that constraint and also gives a more intuitive explanation of safety by its inductive nature.

This method to derive invariants is called InvariantBySafety when we do not use refinement and InvariantByRefinement when we use refinement to prove safety before. Both necessarily always use constraints over the rationals. In [Esp+14] we also present a method to compute a minimal invariant. For this, we add a constraint $|[\![\lambda]\!]| \leq k$ as an appropriate Presburger constraint to (3.19), and solve it with an SMT solver for decreasing k until the system becomes unsatisfiable. By this, we obtain a solution where $|[\![\lambda]\!]|$ is minimal and therefore an invariant I where the part for $\lambda \cdot \mathbf{m} \leq \lambda \cdot \mathbf{m_0}$ has a minimal number of atoms when written as a linear expression.

For our example with Lamport's algorithm in Figure 2.1b, we found that after adding the initially marked trap $Q = \{p_2, q_2, q_3, nbit_1, nbit_2\}$ the constraint (3.10) becomes unsatisfiable. This also holds over the rationals. After solving the constraint (3.19) and computing a minimal invariant, we obtain:

$$\begin{aligned} I(\mathbf{m}) = (\ &\mathbf{m}(p_2) &&+ \mathbf{m}(q_2) + \mathbf{m}(q_3) &&+ \mathbf{m}(nbit_1) + \mathbf{m}(nbit_2) \geq 1\ ) \\ \wedge (\ &\mathbf{m}(p_2) + \mathbf{m}(p_3) + \mathbf{m}(q_2) + \mathbf{m}(q_3) + \mathbf{m}(q_5) + \mathbf{m}(nbit_1) + \mathbf{m}(nbit_2) \leq 2\ ) \end{aligned}$$

Clearly $I(\mathbf{m_0})$ holds, and inductivity can be verified by inspecting every transition of the net. Subtracting the first inequation from the second then shows $\mathbf{m}(p_3)+\mathbf{m}(q_5) \leq 1$ for all reachable markings $\mathbf{m}$ and therefore proves safety of the system.

3.2.5 Experimental Evaluation

We implemented the methods presented in this section in a tool called Petrinizer, built on top of the Z3 SMT solver [MB08]. The tool takes a system $(N, \mathbf{m_0})$ and a safety prop-

	PETRINIZER							
Suite	SAF/$\mathbb{Q}$	SAF/$\mathbb{Z}$	REF/$\mathbb{Q}$	REF/$\mathbb{Z}$	IIC	BFC	MIST	Total
MIST	14	14	20	20	**23**	21	19	23
BFC	**2**	**2**	**2**	**2**	**2**	**2**	**2**	2
Medical	4	4	4	4	9	**12**	10	12
Bug-tracking	**32**	**32**	**32**	**32**	0	0	0	40
Erlang	32	32	36	**38**	17	26	2	38
Total	84	84	94	**96**	51	61	33	115

Tab. 3.1 Safe instances that were successfully proved safe by PETRINIZER and other tools.

erty φ as input and invokes one of the methods SAFETY/K, SAFETYBYREFINEMENT/K, INVARIANTBYSAFETY or INVARIANTBYREFINEMENT, the first two with $K \in \{\mathbb{Z}, \mathbb{Q}\}$ and the last two with optional minimization. For refinement, only the first refinement step with initially marked traps is enabled.

For the experimental evaluation, we have the following goals:

(1) Evaluate the degree of completeness of the methods.

(2) Evaluate the usefulness and necessity of traps.

(3) Evaluate the benefit of using integer arithmetic over rational numbers.

(4) Evaluate the quality of the produced invariants and the benefit of minimization.

(5) Compare the performance of PETRINIZER with other state-of-the-art tools.

For the last goal, we included the tools MIST[1], BFC[2] [KKW12] in version 1.0 and IIC [Klo+13] in the evaluation. All three tools implement complete methods for deciding coverability problems, and IIC can also construct inductive invariants.

We evaluate our tool on benchmarks from five different sources. Each benchmark instance is a combination of a system and a safety property, where the negation of the safety property is always a coverability problem. We have 29 instances from Petri net examples for the MIST toolkit, 46 instances from the analysis of concurrent C programs used in the evaluation of BFC [KKW12], 12 instances from the analysis of a medical system, 41 instances from the analysis of a bug-tracking system, and finally 50 instances from the analysis of Erlang programs. In total, there are 178 instances, of which 115 are safe.

All experiments were performed on a cluster of identical machines equipped with Intel Xeon 2.66 GHz CPUs and 48 GB of memory. Execution time was limited to 100,000

[1] https://github.com/pierreganty/mist
[2] https://www.cprover.org/bfc/

Method/tool	SAF/$\mathbb{Q}$	SAF/$\mathbb{Z}$	REF/$\mathbb{Q}$	REF/$\mathbb{Z}$	INVSAF	INVSAFMIN
Mean (safe)	69.26	70.20	69.36	72.20	168.46	203.05
Median (safe)	2.45	2.23	2.35	3.81	3.70	4.03
Mean (all)	45.17	46.04	45.52	47.70	109.23	131.58
Median (all)	0.44	0.43	0.90	0.93	0.66	1.00

Method/tool	INVREF	INVREFMIN	IIC	BFC	MIST
Mean (safe)	228.88	275.12	56954.09	47126.12	69196.77
Median (safe)	5.96	6.30	100000.00	1642.43	100000.00
Mean (all)	148.57	178.45	44089.93	31017.80	61586.56
Median (all)	1.37	1.94	138.00	0.77	100000.00

Tab. 3.2 Mean and medium execution times in seconds for all configurations of PETRINIZER, IIC, BFC and MIST. Memory-out cases were set to the timeout value of 100,000 s.

seconds (approx. 28 hours) and memory to 2 GB. Sequentially, each configuration of PETRINIZER and each other tool was run once on each instance of the benchmarks. The results of the experimental evaluation, originally conducted in [Esp+14], are given in Tables 3.1 and 3.2 and Figures 3.3 and 3.4. We abbreviate the different configurations of PETRINIZER, using SAF for SAFETY, REF for SAFETYBYREFINEMENT, INVSAF for INVARIANTBYSAFETY and INVREF for INVARIANTBYREFINEMENT, appending MIN for the configuration with invariant minimization. For the invariant size, we count the number of atoms in the linear expressions of the formula for the invariant.

We succinctly give the results of the evaluation, a more detailed presentation is given in [Esp+14]. For goal (1) and (5), we see in Table 3.1 that PETRINIZER can prove safety for 96 of the 115 safe instances in its best configuration, a rate of 83%. This is higher than any other tool which, even though they implement complete methods, are limited by time and memory. Across the different suites, IIC performs better on the MIST suite and BFC performs better on the medical suite. However, PETRINIZER can prove safety for all safe Erlang instances and for 80% of the bug-tracking suite, where it clearly beats the other tools. From Table 3.2 and Figure 3.4, we see that other tools are often faster for instances they solve quickly, which happens mainly on small instances due to the time overhead of the not fully optimized implementation of PETRINIZER. Nevertheless, PETRINIZER terminates within 100 seconds on all but two instances, and beats the other tools often by orders on magnitude on larger instances.

For goal (2) and (3), we see in Table 3.1 that using integer constraints instead of rational constraints does help in two cases when we also use refinement. In Table 3.2 and Figure 3.3 we also see that integer constraints do not take significantly longer

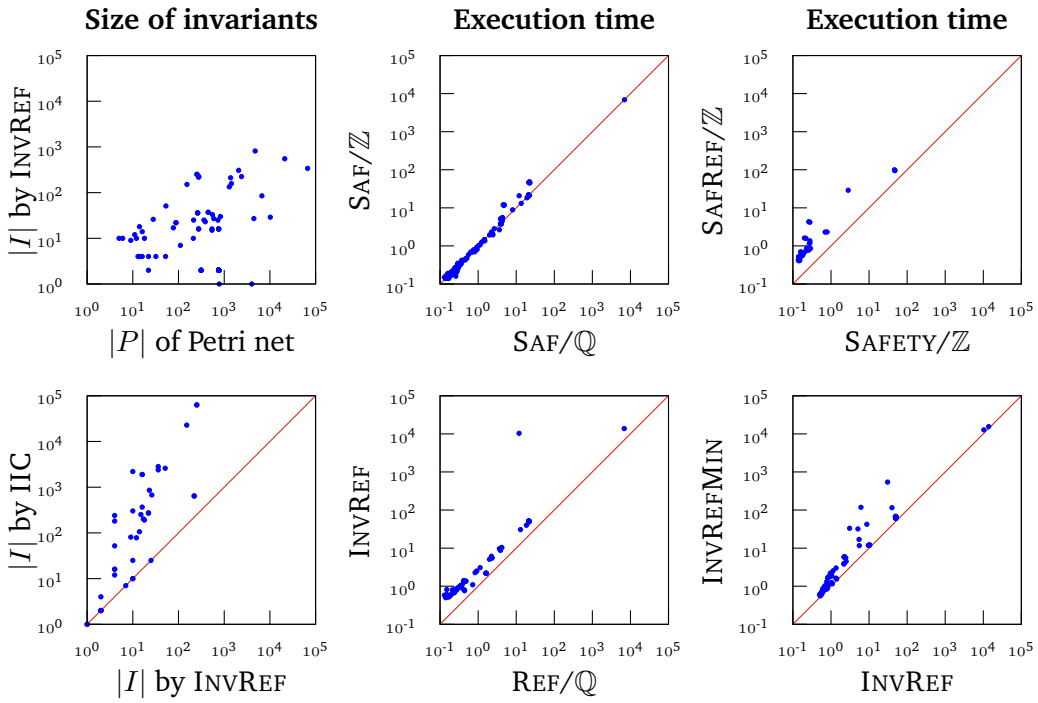


Fig. 3.3 Graphs comparing size of invariants and execution times in seconds for IIC and different configurations of Petrinizer. Each dot represents one instance. For the graphs including InvRef, only instances where this method successfully produced an invariant are shown. For the top right graph, only instances where at least one trap is needed to show safety are shown.

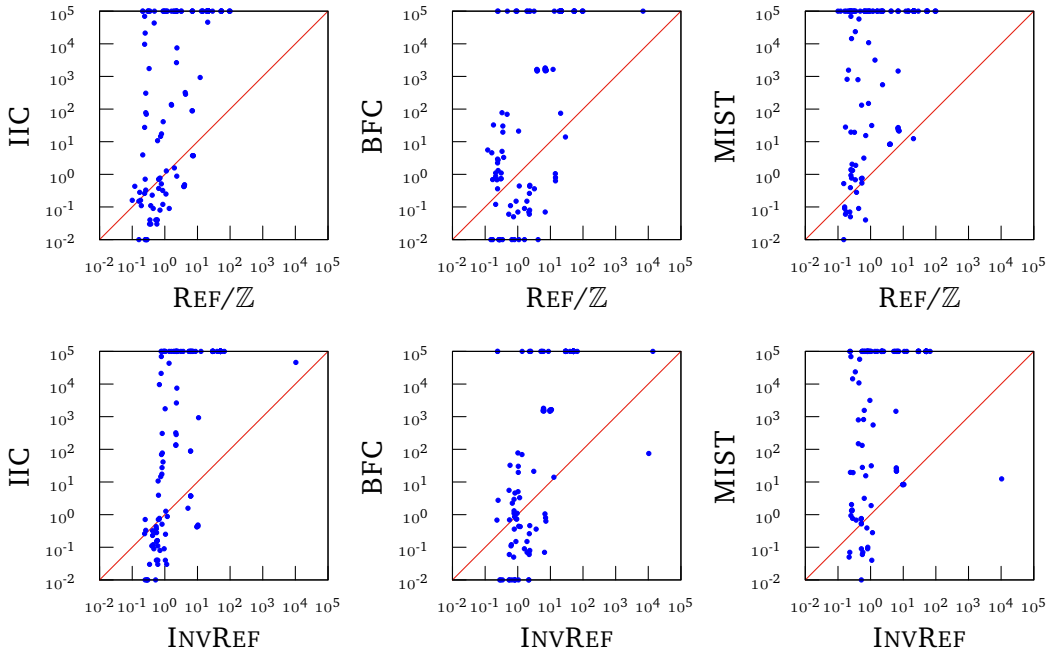


Fig. 3.4 Graphs comparing execution times in seconds for IIC, BFC, MIST and two configurations of Petrinizer. Each dot represents one instance.

to solve. The refinement with traps is even more helpful: we can solve 96 instances when using refinement and only 84 without. Using refinement does take more time, which is mainly due to repeatedly needing to solve the main constraint again for each newly found trap. In our experiments, PETRINIZER added at most 9 traps until it could prove the property or gave up.

For goal (4), we see in Figure 3.3 that the invariants produced by PETRINIZER are usually very succinct in comparison to the number of places of the Petri net. They are often several orders of magnitude smaller than those produced by IIC and never larger. Minimization of the invariants is only useful on four instances, and only resulted in a reduction in size of 2-3%. Meanwhile minimization can have a large impact on performance, leading to a slowdown of $30\times$ on one instance. An explanation for this behavior could be the fact that instead of only needing to solve only a satisfiable linear constraint over the rationals to derive the invariant, we have to also solve an unsatisfiable Presburger constraint to show minimality.

3.3 New Contributions for Liveness Properties

In [EM15], we present an incomplete decision procedure for verifying liveness properties or equivalently for determining fair termination. Similarly to the procedure for safety in the preceding section, this procedure is based on the construction of linear and Presburger constraints.

3.3.1 T-surinvariants

The first component of our procedure again employs the incidence matrix (Definition 3.2). Instead of the marking equation, which only characterizes finite transition sequences, we now introduce *T-surinvariants*, which characterize certain infinite sequences of the Petri net.

Definition 3.21 (T-surinvariant)**.** Let $N = (P, T, F)$ be a Petri net. A *T-surinvariant* of N is a vector $\mathbf{x} \in \mathbb{Q}^T$ such that $\mathbf{C} \cdot \mathbf{x} \geq \mathbf{0}$, where $\mathbf{C}$ is the incidence matrix of N. A T-surinvariant $\mathbf{x}$ is *semi-positive* if $\mathbf{x} \geq \mathbf{0}$ and $\mathbf{x} \neq \mathbf{0}$.

We have the following property of T-surinvariants, a slightly generalized version of the theorem given in [EM15].

Theorem 3.22 ([DE95; EM15; Rei13])**.** *Let N be a Petri net and $U \subseteq T$ be a set of transitions. There is an infinite transition sequence σ enabled at some marking $\mathbf{m_0}$ of N with $\mathrm{Inf}(\sigma) = U$ if and only if there is a semi-positive T-surinvariant $\mathbf{x}$ with $[\![\mathbf{x}]\!] = U$.*

Proof. ($\Rightarrow$) Let $\mathbf{m_0}$ be a marking of N such that σ is enabled at $\mathbf{m_0}$. By Dickson's lemma [Dic13] there are markings $\mathbf{m_1}, \mathbf{m_2}$ and sequences $\sigma_1, \sigma_2 \in T^*$ such that

$[\![\sigma_2]\!] = \mathrm{Inf}(\sigma)$, $\mathbf{m_0} \xrightarrow{\sigma_1} \mathbf{m_1} \xrightarrow{\sigma_2} \mathbf{m_2}$ and $\mathbf{m_2} \geq \mathbf{m_1}$. The vector $\mathbf{x} = \Psi(\sigma_2)$ is then a semi-positive T-surinvariant with $[\![\mathbf{x}]\!] = [\![\sigma_2]\!] = \mathrm{Inf}(\sigma) = U$.

($\Leftarrow$) Let σ be any finite sequence with $\Psi(\sigma) = \mathbf{x}$ and take $\mathbf{m_0} = \sum_{t \in T} \mathbf{x}(t) \cdot {}^\bullet t$. Then σ is non-empty, enabled at $\mathbf{m_0}$ and leads to a marking $\mathbf{m_1}$ with $\mathbf{m_1} \geq \mathbf{m_0}$, as $\mathbf{x}$ is a semi-positive T-surinvariant. Thus σ^ω is an infinite transition sequence enabled at $\mathbf{m_0}$ with $\mathrm{Inf}(\sigma^\omega) = [\![\sigma]\!] = [\![\mathbf{x}]\!] = U$. □

As the proof suggests, T-surinvariants characterize subsequences that lead to the same or a larger marking, and can thus be repeated infinitely often. Such a subsequence might however never be enabled at any reachable marking. We say that a T-surinvariant $\mathbf{x}$ is *realizable* for a system $(N, \mathbf{m_0})$ if there is an infinite transition sequence σ of N enabled at $\mathbf{m_0}$ such that $[\![\mathbf{x}]\!] = \mathrm{Inf}(\sigma)$.

Given a system $(N, \mathbf{m_0})$ and a liveness property φ, we define the following constraint over a vector $\mathbf{x} \in \mathbb{Q}^T$ for finding certain semi-positive T-surinvariants.

$$\mathrm{TInvariant}_\varphi(\mathbf{x}) \stackrel{\text{def}}{=} \mathbf{C} \cdot \mathbf{x} \geq \mathbf{0} \wedge \mathbf{x} \geq \mathbf{0} \wedge \mathbf{x} \neq \mathbf{0} \wedge \neg\varphi([\![\mathbf{x}]\!]) \tag{3.23}$$

We can express $\varphi([\![\mathbf{x}]\!])$ as a Presburger constraint by substituting each atom t in φ with $\mathbf{x}(t) \geq 1$. The whole constraint (3.23) is then a Presburger constraint. If it is unsatisfiable, then no semi-positive T-surinvariant satisfies $\neg\varphi([\![\mathbf{x}]\!])$, so by Theorem 3.22 we have that $\varphi(\mathrm{Inf}(\sigma))$ holds for every infinite sequence σ, showing that the system satisfies the liveness property φ. This gives us our first incomplete decision procedure for fair termination problems and related liveness problems. Given a system $(N, \mathbf{m_0})$ and a liveness property φ, we check satisfiability of (3.23) using an SMT solver. We denote this procedure by LIVENESS.

The procedure so far is efficient, but fails to prove liveness properties of more complex systems. This is because it is completely independent of the initial marking $\mathbf{m_0}$ of the system, and only checks if there is an infinite sequence enabled at *some* marking, reachable or not. Consider for example Lamport's algorithm in Figure 2.1b with the previous liveness property φ in constraint (3.1). We have that $\varphi([\![\mathbf{x}]\!])$ does not hold for the semi-positive T-surinvariant $\mathbf{x} = \wr s_3, t_5 \int$, so the constraint (3.23) is satisfiable. While $\mathbf{x}$ would be realizable from the marking $\mathbf{m} = \wr p_2, q_4, bit_1, bit_2 \int$, this $\mathbf{m}$ is not reachable from $\mathbf{m_0}$, as we will see later.

3.3.2 Refinement with P-components

We give two give refinement structures to show that certain T-surinvariants are not realizable, based on P-components and traps.

Definition 3.24 (P-component)**.** Let $N = (P, T, F)$ be a Petri net. A *P-component* of N is a subnet $N' = (P', T')$ of N such that $P' \neq \emptyset$, $T' = [\![{}^\bullet P']\!] \cup [\![P'^\bullet]\!]$ and for every $t \in T'$, we have ${}^\bullet t(P') = t^\bullet(P') = 1$.

An essentially property of P-components is that the number of tokens within never changes: for a P-component (P', T') of a Petri net N and two markings $\mathbf{m}, \mathbf{m}'$ of N with $\mathbf{m} \xrightarrow{*} \mathbf{m}'$, we have $\mathbf{m}(P') = \mathbf{m}'(P')$.

If a P-component only contains one token initially, then in any reachable marking only one place of the P-component will be marked, so transitions requiring tokens in other places of the component are not enabled at that point. If we know that the token will not be transported to another place, then those transitions will never be enabled.

To illustrate our procedure, let us for a moment return to our example for Lamport's algorithm in Figure 2.1b. We saw that the net has the semi-positive T-surinvariant $\mathbf{x} = \{\!|s_3, t_5|\!\}$. A P-component $(P', T') = (\{q_1, q_4, bit_2\}, \{s_3, t_1, t_3, t_4, t_5, t_7\})$ of the net is shown in Figure 3.5a, from which we can infer that bit_2 is set to $false$ if the second process is in state q_1 or q_4. Figure 3.5b shows the subnet $(P', T' \cap [\![\mathbf{x}]\!])$. This subnet is not strongly connected, which means along any infinite sequence σ with $\mathrm{Inf}(\sigma) = [\![\mathbf{x}]\!]$ the token will eventually remain in one the three places q_1, q_4 or bit_2. However s_3 needs a token from bit_2 and t_5 a token from q_4. Therefore we can conclude that there is no transition sequence containing only s_3 and t_5 infinitely often, so $\mathbf{x}$ is not realizable. Based on this reasoning we show the following in [EM15] to refine T-surinvariants.

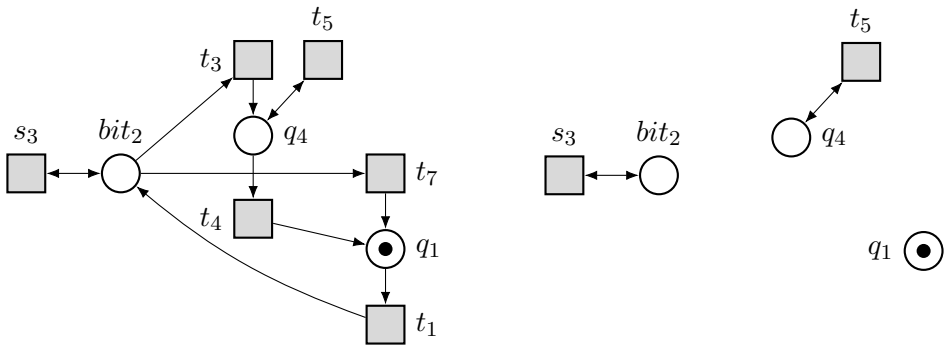
Lemma 3.25 ([EM15]). *Let $(N, \mathbf{m_0})$ be a system and $\mathbf{x}$ a T-surinvariant of N. If N has a P-component (P', T') such that $\mathbf{m_0}(P') = 1$, and the subnet $(P', T' \cap [\![\mathbf{x}]\!])$ is not strongly connected, then $\mathbf{x}$ is not realizable.*

To find P-components that show that a given T-surinvariant $\mathbf{x}$ is not realizable, we first construct a constraint over two vectors $\mathbf{p} \in \mathbb{B}^P, \mathbf{t} \in \mathbb{B}^T$ to find a P-component $(P', T') = ([\![\mathbf{p}]\!], [\![\mathbf{t}]\!])$ with $\mathbf{m_0}(P') = 1$.

$$\begin{aligned} \mathrm{FindPComponent}_{\mathbf{m_0}}(\mathbf{p}, \mathbf{t}) \stackrel{\text{def}}{=} \mathbf{m_0} \cdot \mathbf{p} = 1 \\ \wedge \bigwedge_{p \in P} (\mathbf{p}(p) = 1 \rightarrow \mathbf{t} \geq {}^\bullet p + p^\bullet) \\ \wedge \bigwedge_{t \in T} (\mathbf{t}(t) = 1 \rightarrow ({}^\bullet t \cdot \mathbf{p} = 1 \wedge t^\bullet \cdot \mathbf{p} = 1)) \end{aligned} \tag{3.26}$$

The second line here encodes the part $T' = [\![P'^\bullet]\!] \cup [\![{}^\bullet P']\!]$. Next, we construct a constraint that is satisfiable iff the subnet $(P', T' \cap [\![\mathbf{x}]\!])$ is not strongly connected. For this, we add additional variables $\mathbf{t_1}, \mathbf{t_2} \in \mathbb{B}^{[\![\mathbf{x}]\!]}$ representing a partition $(T_1, T_2) = ([\![\mathbf{t_1}]\!], [\![\mathbf{t_2}]\!])$ of $T' \cap [\![\mathbf{x}]\!]$ such that there are no arcs $(t_1, p), (p, t_2)$ in (P', T') for some $t_1 \in T_1, p \in P', t_2 \in T_2$.

$$\begin{aligned} \mathrm{NSC}_{\mathbf{x}}(\mathbf{p}, \mathbf{t}, \mathbf{t_1}, \mathbf{t_2}) \stackrel{\text{def}}{=} \mathbf{t_1} \neq \mathbf{0} \wedge \mathbf{t_2} \neq \mathbf{0} \wedge \bigwedge_{t \in [\![\mathbf{x}]\!]} \mathbf{t}(t) = \mathbf{t_1}(t) + \mathbf{t_2}(t) \\ \wedge \bigwedge_{\substack{t_1 \in [\![\mathbf{x}]\!] \\ p \in [\![t_1^\bullet]\!] \\ t_2 \in [\![p^\bullet]\!] \cap [\![\mathbf{x}]\!]}} (\mathbf{t_1}(t_1) = 0 \vee \mathbf{p}(p) = 0 \vee \mathbf{t_2}(t_2) = 0) \end{aligned} \tag{3.27}$$



(a) P-component (P', T') **(b)** Subnet $(P', T' \cap [\![\mathbf{x}]\!])$

Fig. 3.5 P-component and subnet of the Petri net for Lamport's algorithm.

We then use the combined constraint

$$\text{FindPComponent}_{\mathbf{m_0}}(\mathbf{p}, \mathbf{t}) \wedge \text{NSC}_{\mathbf{x}}(\mathbf{p}, \mathbf{t}, \mathbf{t_1}, \mathbf{t_2}) \tag{3.28}$$

to find a P-component showing that a T-surinvariant $\mathbf{x}$ obtained from a solution of the constraint (3.23) is not realizable. If that constraint is satisfiable, we extract a P-component (P', T') and the partition (T_1, T_2) from the solution, and construct the following refinement constraint, excluding $\mathbf{x}$ and any other T-invariant where the intersection $T' \cap [\![\mathbf{x}]\!]$ yields the same partition of (P', T').

$$\text{PComponent}_{P',T',T_1,T_2}(\mathbf{x}) \stackrel{\text{def}}{=} \left(\sum_{t \in T_1} \mathbf{x}(t) \geq 1 \wedge \sum_{t \in T_2} \mathbf{x}(t) \geq 1 \right) \rightarrow \sum_{t \in T' \setminus (T_1 \cup T_2)} \mathbf{x}(t) \geq 1 \tag{3.29}$$

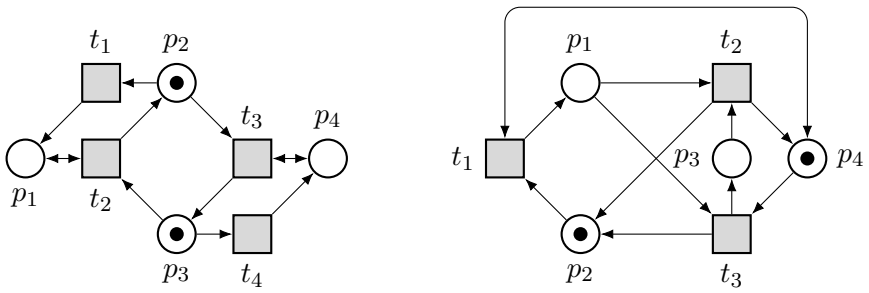
Assuming we have already found a set $\mathcal{P}$ of P-components and their respective partitions, we construct the following constraint by adding the refinement constraints to the T-surinvariant constraint (3.23).

$$\text{TInvariant}_{\varphi}(\mathbf{x}) \wedge \bigwedge_{(P',T',T_1,T_2) \in \mathcal{P}} \text{PComponent}_{P',T',T_1,T_2}(\mathbf{x}) \tag{3.30}$$

As long as the constraint is satisfiable with a vector $\mathbf{x}$, we try to find a new P-component and partition using constraint (3.28). If we succeed, we add it to the set $\mathcal{P}$, and repeat the process. If we do not find a P-component, we give up. Once the main constraint (3.30) is unsatisfiable, we have proven that the liveness property φ holds, since we have shown that there is no realizable T-surinvariant $\mathbf{x}$ where $\neg\varphi([\![\mathbf{x}]\!])$ holds. We call this method RefPComp.

In our example, with the P-component in Figure 3.5a and the partition $(T_1, T_2) = (\{s_3\}, \{t_5\})$, we then construct the following refinement constraint (3.29):

$$(\mathbf{x}(s_3) \geq 1 \wedge \mathbf{x}(t_5) \geq 1) \rightarrow \mathbf{x}(t_1) + \mathbf{x}(t_3) + \mathbf{x}(t_4) + \mathbf{x}(t_7) \geq 1$$



(a) Net without P-components **(b)** Net without unmarked traps

Fig. 3.6 Terminating systems for which refinement is insufficient.

After adding this constraint and solving (3.30), we find another semi-positive T-surinvariant $\mathbf{x}' = \wr s_3, t_1, t_1, t_2, t_3, t_4, t_5, t_6, t_7 \wr$ of the net in Figure 2.1b. In a second refinement step, we can show that this T-surinvariant is also not realizable by the P-component with $P' = \{p_2, p_3, nbit_1\}$ and the partition $(T_1, T_2) = (\{s_3\}, \{t_4, t_6\})$. We add the refinement constraint

$$(\mathbf{x}(s_3) \geq 1 \wedge \mathbf{x}(t_4) + \mathbf{x}(t_6) \geq 1) \rightarrow \mathbf{x}(s_1) + \mathbf{x}(s_2) + \mathbf{x}(s_4) \geq 1$$

after which the constraint (3.30) becomes unsatisfiable, so we have proven that the fairness property for the first process of Lamport's algorithm holds.

3.3.3 Refinement with Traps

The method for liveness with refinement by P-components can prove more properties than before, but is still fails for simple examples as the net in Figure 3.6a, which is derived from a subnet of a leader election algorithm. This net has no infinite transition sequences enabled at the given marking, but the net has the semi-positive T-invariant $\mathbf{x} = \wr t_2, t_3 \wr$ and no P-components, so our method fails. We address this with a refinement using traps. We use the following property shown in [EM15]:

Lemma 3.31 ([EM15]). *Let $(N, \mathbf{m_0})$ be a system and let $\mathbf{x}$ be a realizable T-surinvariant. Then some marking $\mathbf{m}$ reachable from $\mathbf{m_0}$ in N marks every trap of the subnet $(P', T') = (\llbracket \llbracket \mathbf{x} \rrbracket^\bullet \rrbracket, \llbracket \mathbf{x} \rrbracket)$.*

The idea is that if $\mathbf{x}$ is realizable by a sequence σ with $\mathrm{Inf}(\sigma) = \llbracket \mathbf{x} \rrbracket$, then eventually all transitions in this subnet and only those will occur, thus marking every trap. Now if we can show that no marking $\mathbf{m}$ that marks every trap of (P', T') is reachable, then we have shown that $\mathbf{x}$ is not realizable. This is therefore a safety property, so we can reuse the method SAFETY from Section 3.2 with the marking equation to overapproximate reachability. Since (P', T') may have exponentially many traps, we use an iterative approach as follows.

Let $(P', T') = (\llbracket \llbracket \mathbf{x} \rrbracket^\bullet \rrbracket, \llbracket \mathbf{x} \rrbracket)$. We maintain a set $\mathcal{Q}$ of traps of (P', T') where initially $\mathcal{Q} = \emptyset$. We first check if all reachable markings already leave some trap of $\mathcal{Q}$ unmarked by checking satisfiability of the constraint $\text{MarkingEquation}_\psi(\mathbf{m_0}, \mathbf{m}, \mathbf{x})$, with $\psi(\mathbf{m}) = \bigvee_{Q \in \mathcal{Q}} \mathbf{m}(Q) = 0$, i.e. the marking constraint (3.5) from the method SAFETY. If it is unsatisfiable, then no reachable marking marks all traps of $\mathcal{Q}$ and thus also not all traps of (P', T'), so $\mathbf{x}$ is not realizable. Otherwise, we extract a marking $\mathbf{m}$ from the solution. Next, we look for a proper trap of (P', T') unmarked at $\mathbf{m}$ and not already in $\mathcal{Q}$ with the constraint $\text{Trap}_{\llbracket x \rrbracket^\bullet, \mathbf{m}, \llbracket \mathbf{x} \rrbracket}(\mathbf{q})$, reusing our generic trap constraint (3.11). If the constraint is unsatisfiable, then we give up. If the constraint is satisfiable, extract a trap $Q = \llbracket \mathbf{q} \rrbracket \cap P'$ from a solution, add Q to $\mathcal{Q}$ and repeat the process. Note that $\llbracket \mathbf{q} \rrbracket \cap P'$ is a trap in (P', T') since $\llbracket \mathbf{q} \rrbracket$ is an $\llbracket \mathbf{x} \rrbracket$-trap and $P' = \llbracket \llbracket \mathbf{x} \rrbracket^\bullet \rrbracket$. Moreover $Q \notin \mathcal{Q}$ since it is unmarked at $\mathbf{m}$ and $\mathbf{m}$ marks all traps of $\mathcal{Q}$.

Once we have found a set $\mathcal{Q}$ that shows that $\mathbf{x}$ is not realizable, we add the following refinement constraint. It not only excludes T-surinvariants with the same support as $\mathbf{x}$, but instead specifies that if $\llbracket \mathbf{x} \rrbracket^\bullet$ marks all traps in $\mathcal{Q}$, then $\llbracket \mathbf{x} \rrbracket$ must also contain a transition emptying a trap in $\mathcal{Q}$, which of course cannot belong to T'.

$$\text{TrapRefinement}_{\mathcal{Q}}(\mathbf{x}) \stackrel{\text{def}}{=} \left(\bigwedge_{Q \in \mathcal{Q}} \mathbf{x}(\llbracket {}^\bullet Q \rrbracket) \geq 1 \right) \rightarrow \left(\bigvee_{Q \in \mathcal{Q}} \mathbf{x}(\llbracket Q^\bullet \rrbracket \setminus \llbracket {}^\bullet Q \rrbracket) \geq 1 \right) \tag{3.32}$$

We can combine both refinement with P-components and with traps. After we have found a set $\mathcal{P}$ of P-components and a set of sets of traps $\mathcal{S}$, we use the following extension of the previous constraint (3.30).

$$\text{TInvariant}_\varphi(\mathbf{x}) \wedge \bigwedge_{(P', T', T_1, T_2) \in \mathcal{P}} \text{PComponent}_{P', T', T_1, T_2}(\mathbf{x}) \wedge \bigwedge_{\mathcal{Q} \in \mathcal{S}} \text{TrapRefinement}_{\mathcal{Q}}(\mathbf{x}) \tag{3.33}$$

We call this method REFPCOMPTRAPS. Similarly to minimization of the invariant in Section 3.2, we also offer an option to minimize the intermediate refinement objects by always choosing a P-component (P', T') satisfying constraint (3.28) such that $|P'|$ is minimized and intermediate traps Q such that $|Q|$ is minimized. This method is denoted by REFPCOMPTRAPSMIN.

We note that refinement by traps is more generic than refinement by P-components: for any T-surinvariant $\mathbf{x}$, P-component (P', T') with $\mathbf{m_0}(P') = 1$ and partition T_1, T_2 of $T' \cap \llbracket \mathbf{x} \rrbracket$, both $P' \cap \llbracket T_1^\bullet \rrbracket$ and $P' \cap \llbracket T_2^\bullet \rrbracket$ are traps in $(\llbracket \llbracket \mathbf{x} \rrbracket^\bullet \rrbracket, \llbracket \mathbf{x} \rrbracket)$, and we can show with the marking equation that both are never marked simultaneously. However P-components generally exclude more T-surinvariants, so we first look for a P-component to exclude a given $\mathbf{x}$ and, if that fails, then look for a set of traps to exclude it.

For the net in Figure 3.6a where refinement with P-components fails, we have the T-surinvariant $\mathbf{x} = \wr t_2, t_3 \wr$. In the subnet $(\llbracket \llbracket \mathbf{x} \rrbracket^\bullet \rrbracket, \llbracket \mathbf{x} \rrbracket) = (\{p_1, p_2, p_3, p_4\}, \{t_2, t_3\})$, the

sets $Q_1 = \{p_1\}$, $Q_2 = \{p_4\}$ and $Q_3 = \{p_2, p_3\}$ are traps, which we find using our iterative method. Using the marking equation, we can show that they can never all be marked simultaneously. With $\mathcal{Q} = \{Q_1, Q_2, Q_3\}$, we then add the following refinement constraint (3.32):

$$\Big(\mathbf{x}(t_1) + \mathbf{x}(t_2) \geq 1 \wedge \mathbf{x}(t_3) + \mathbf{x}(t_4) \geq 1 \wedge \mathbf{x}(t_2) + \mathbf{x}(t_3) \geq 1\Big) \to \mathbf{x}(t_1) + \mathbf{x}(t_4) \geq 1$$

The net has no semi-positive T-surinvariant also satisfying this constraint, which shows that the system is terminating. Our method with trap refinement is however still incomplete: the Petri net in Figure 3.6b has the T-surinvariant $\mathbf{x} = \wr t_1, t_1, t_2, t_3 \wr$ where $(\llbracket \llbracket \mathbf{x} \rrbracket^\bullet \rrbracket, \llbracket \mathbf{x} \rrbracket)$ is the whole net, but all traps of the net are already marked initially.

3.3.4 Certificates for Termination

For our liveness method, we can consider the final set of unsatisfiable constraints (3.33) as a certificate for the given liveness property. One can verify that the refinement objects are valid P-components and traps individually in polynomial time, however checking unsatisfiability of the whole Presburger constraint is in NP. We briefly present a method to derive a different, easier to check certificate in special cases, similar to the inductive invariant for SAFETY in Section 3.2.4. This was not presented in [EM15], but explored by the authors and later used in [Blo+17].

We show that we can derive a certificate for termination, i.e. the liveness property with $\varphi = \mathit{false}$, in the case where we do not use any refinement, i.e. the constraint (3.23) is unsatisfiable. The certificate takes the form of a ranking function. Using Farkas' lemma [Sch86], we can show that the constraint (3.23) is unsatisfiable if and only if the following constraint is satisfiable with a vector $\mathbf{y} \in \mathbb{Z}^P$:

$$\mathbf{y} \geq \mathbf{0} \wedge \mathbf{y} \cdot \mathbf{C} \leq -\mathbf{1} \tag{3.34}$$

Now if (3.23) is unsatisfiable, then we can extract a vector $\mathbf{y}$ from a solution to the constraint (3.34) and construct the ranking function $R : \mathbb{Z}^P \mapsto \mathbb{N}$ with $R(\mathbf{m}) \stackrel{\text{def}}{=} \mathbf{y} \cdot \mathbf{m}$. The ranking function R satisfies the property that for any markings $\mathbf{m}, \mathbf{m}'$ of N, if $\mathbf{m} \to \mathbf{m}'$, then $R(\mathbf{m}) > R(\mathbf{m}')$, which easily follows as $\mathbf{y}$ satisfies the constraint (3.34). Thus for any initial marking $\mathbf{m_0}$, the length of a sequence σ with $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$ for some $\mathbf{m}$ is bounded by $R(\mathbf{m_0})$, so there is no infinite transition sequence enabled at $\mathbf{m_0}$. This proves that the system is terminating, even for any initial marking. The property of R can be verified in polynomial time by inspecting it for the single steps $\mathbf{m} \xrightarrow{t} \mathbf{m}'$ of each transition t of the Petri net.

3.3.5 Experimental Evaluation

We implemented the methods presented in this section in the tool PETRINIZER, which takes as input a system and a liveness properties and executes the method LIVENESS,

with optional refinement by P-components or additionally traps, both with optional minimization of the refinement structures.

We have the following goals for the experimental evaluation:

(1) Evaluate the degree of completeness of the methods.

(2) Evaluate the usefulness and necessity of refinement with P-component and traps and their minimization.

(3) Compare the performance of PETRINIZER with other state-of-the-art tools.

For the last goal, we found no tools that directly support potentially infinite-state systems modeled by Petri nets and liveness properties of our form. For some finite-state systems and liveness properties with a fairness assumption, we compare PETRINIZER with the model checker SPIN [Hol97].

We evaluate our tool on benchmarks from five different sources. Each instance is a system combined with a liveness property. We have two suites from workflow nets for business processes [Aal03], the first one containing 1386 instances from IBM business process models [Fah+09] and the second containing 590 instances from SAP reference models [Don+07]. The third suite are 50 instances from the analysis of Erlang programs. For these first three suites, we consider termination as the liveness property. The fourth suite are classic asynchronous programs, including a leader election algorithm [DKR82], a snapshot algorithm [Bou87] and three mutual exclusion algorithms [Lam86; Pet81; Szy88]. All algorithms are scalable in the number of processes. We scale each of the five algorithms from $n = 2$ to 6 processes, resulting in 25 instances. For the first two algorithms we consider repeated liveness as the property, i.e. that electing a leader or taking a snapshot happens infinitely often, and for the mutual exclusion algorithms the property that the first process enters the critical section infinitely often. All properties include a fairness assumption on the scheduler. As the fifth suite, we collected five instances with the termination property from the literature.

All experiments were performed on the same machine, equipped with an Intel Core i7-4810MQ CPU at 2.8 GHz and 16 GB of memory. Execution time was limited to 2 hours and memory to 16 GB. The results of the experimental evaluation, originally conducted in [EM15], are given in Figure 3.7 and Tables 3.8 and 3.9[3].

For goal (1) and (2), we see in Table 3.8 that without refinement we can already show termination for all but two instances of the IBM and SAP suite and 27 of the 33 terminating instances of the Erlang suite, but we cannot show liveness for any instance of the asynchronous and literature suite. Using P-components allows us to prove liveness for 14 instances of the asynchronous and 3 of the literature suite, and

[3] A minor error in Table 3.8 of [EM15], mistakenly swapping the labels of IBM and SAP, has been corrected here.

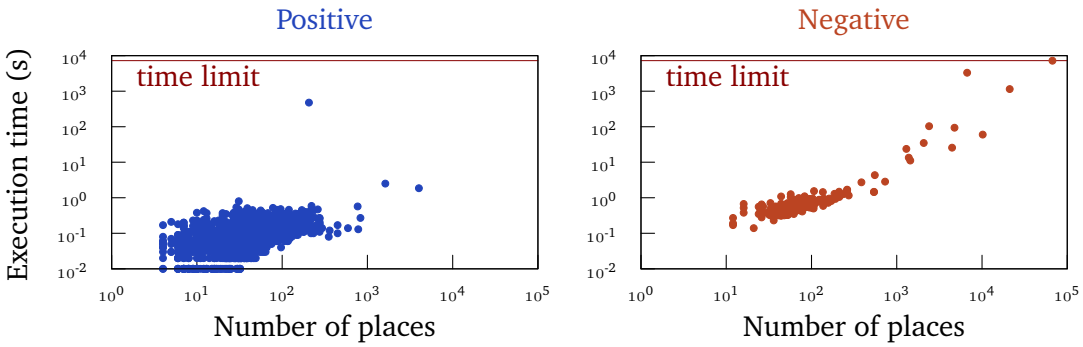


Fig. 3.7 Execution time of PETRINIZER in dependence on the number of place for the benchmark suites IBM, SAP, Erlang and Literature. Each dot represents an execution of the configuration REFPCOMPTRAPS for one instance.

Benchmark	LIVENESS	REFPCOMP	REFPCOMPTRAPS	Terminating
IBM	1263	1263	1264	1264
SAP	571	571	572	572
Erlang	27	27	27	33
Asynchronous	0	14	20	25
Literature	0	3	5	5
Total	1861	1878	1888	1899

Tab. 3.8 Fairly terminating instances with rate of success by PETRINIZER.

additionally using traps proves termination for the remaining IBM, SAP and literature instances, and shows the liveness property for 6 more instances of the asynchronous suite. Overall we can show the property for 1888 of the 1899 instances, and for at least 80% of each individual suite.

For further evaluation of goal (2), we see in Table 3.9 the results for the asynchronous suite. Both P-components and traps are used and necessary for some instances. A large number of refinement steps can cause a large overhead in the execution time. Here, minimization of the refinement components reduces the number of refinement steps and the execution time for the leader election, Lamport's and Peterson's algorithm. For the snapshot and Szymanski's algorithm, there is no reduction and a slight increase in execution time.

To evaluate the performance for goal (3), we first see in Figure 3.7 the individual performance of PETRINIZER on all instances of the IBM, SAP, Erlang and literature suites, divided by whether it succeeded in proving the property. In the positive cases, it always terminates in under 3 seconds except for one case, where we need 320

		Petrinizer						
		RefPCompTraps			RefPCompTrapsMin			SPIN
Benchmark	n	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{S}\rvert$	Time	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{S}\rvert$	Time	Time
Leader election by Dolev, Klawe and Rodeh [DKR82]	2	0	4	2.53	0	4	2.30	0.69
	3	0	6	8.45	0	6	9.03	0.74
	4	0	8	35.5	0	8	38.4	15.7
	5	0	13	206	0	10	154	MO
	6	0	17	1104	0	12	728	MO
Snapshot algorithm by Bougé [Bou87]	2	2	0	0.35	2	0	0.30	0.31
	3	3	0	0.50	3	0	0.81	0.72
	4	4	0	0.60	4	0	0.91	10.3
	5	5	0	0.73	5	0	1.41	218
	6	6	0	1.82	6	0	1.63	MO
Lamport's 1-bit algorithm for mutual exclusion [Lam86]	2	2	0	0.50	3	0	0.43	0.69
	3	6	0	1.26	6	0	1.63	0.69
	4	12	0	2.83	13	0	5.50	0.92
	5	27	0	9.34	18	0	11.3	10.4
	6	26	0	13.4	23	0	20.6	MO
Peterson's mutual exclusion algorithm [Pet81]	2	1	0	0.37	1	0	0.41	0.69
	3	13	0	6.57	7	0	8.55	0.71
	4	21	0	65.9	18	0	92.5	1.16
	5	285	0	2289	36	0	911	43.5
	6	-	-	TO	-	-	TO	MO
Szymanski's mutual exclusion algorithm [Szy88]	2	21	6	10.9	26	6	17.6	0.70
	3	Property cannot be proven by RefPCompTraps for $n \geq 3$.						0.80
	4							5.83
	5							347
	6							MO

Tab. 3.9 Comparison of refinement for liveness with and without minimization and comparison of execution time with SPIN on the asynchronous suite. The execution times are given in seconds, where further TO denotes exceeding the time limit and MO exceeding the memory limit.

refinement steps and 8 minutes. In the negative cases, it is slower and reaches the time limit in one case, but still terminates within 3 seconds for all nets with up to 1000 places. For the asynchronous suite and in comparison with SPIN, we see in Table 3.9 that SPIN is faster on smaller systems and for Peterson's algorithm, but reaches the memory limit for the instances with 5 or 6 processes, where we outperform it for the leader election, snapshot and Lamport's algorithm.

Our procedure LIVENESS is therefore quite effective in proving liveness and fair termination properties. As we avoid building an explicit model of the system, we can often outperform SPIN from the point on where an explicit model of the system becomes too large to analyze. With our different refinement methods, the procedure also has a high degree of completeness on the benchmark suites we analyzed.

3.4 Related Work

The marking equation (also known as state equation or flow equation), as well as traps, siphons (also known as cotraps), T-invariants and P-components are all classical techniques for the analysis of Petri nets [DE95; Mur89; Rei13].

Our method for verification of safety properties is based on the work in [EM00], which introduces the combination of the marking equation with trap constraints. It gives a way to test the constraint for all traps by a reduction to a mixed-integer linear programming problem, which however suffers from precision problems, as the encoding in [EM00] introduces a minimization constraint with an optimal value very close to zero. This work also introduces the iterative approach for adding traps presented in [Esp+14]. However, while [EM00] uses integer linear programming tools to solve the constraints, we employ an SMT solver, for which there have many advances regarding the efficiency for solving large problems in practice. Further, precision problems vanish by using constraints with a discrete domain.

In [WW12], the marking equation is used in combination with state-space exploration for solving the reachability problem. It also employs an iterative approach, but solutions to the current set of constraints are then used to guide the exploration of the state-space, generating new constraints when the search fails. In contrast, we use solutions to directly derive new constraints to exclude unreachable markings. More recently, [Thi20] uses a large collection of constraint-based rules, including the marking equation, traps and siphons, in combination with an SMT solver to either directly verify safety properties of Petri nets or reduce the net while preserving the property. They also combine this with a partial state-space exploration which can be guided by the Parikh vector of a solution.

In [Esp+14] only the refinements with initially marked traps is used and later in [Blo+17] the refinement with U-traps and U-siphons. In between, other work employs

similar refinements. [BKP15] uses a constraint-based approach with the marking equation to analyze workflow nets and a refinement with traps and siphons in the subnets induced by solution to the marking equation, similar to our U-traps and U-siphons. [ALW16] also uses the marking equation for program analysis and extends it with a connectivity constraint on the subnet for the solution. [Blo+16] uses a constraint-based approach based on continuous reachability, and shows that it subsumes the basic trap and siphon constraints of [Esp+14]. The continuous reachability approach can also be extended to Petri nets with control states [BH17], more commonly known as vector addition systems with states (VASS).

Our method for verification of liveness properties is based on the work in [EM97], which uses T-invariants to show liveness properties, together with a refinement by P-components. It focuses on the class of 1-safe systems and therefore finite-state systems, while our approach also works on infinite-state systems. As in [EM00], the work in [EM97] relies on a solver for integer programming, and the experimental evaluation includes only two examples.

In [CA95], constraint-based methods are used for the analysis of safety and liveness properties of concurrent systems, where the constraints are solved using integer programming. For safety properties they employ the marking equation. For liveness properties, an execution is decomposed into a finite prefix and a perpetual interval for the remaining infinite part, where the marking equation is used for each part together with inequalities for the infinite part, together resembling T-surinvariants. They also use a decomposition into strongly connected components of each system component, which are essentially P-components. However there the components have to be manually given, while we discover them automatically.

In [LSW06], constraints resembling T-surinvariants are used to show livelock-freedom of asynchronous reactive systems, a liveness property. They employ a refinement procedure based on the analysis of possible cycles in a process. This also eliminates disconnected cycles similarly to P-components, but their method requires enumeration of all cycles, of which there may be exponentially many.

Finally, we want to mention the work by Leroux [Ler10; Ler11], who presents a semi-decision procedure for the complement of the reachability problem based on inductive Presburger invariants. The central result states that if a marking $\mathbf{m}$ is not reachable from $\mathbf{m_0}$, then there exists an inductive invariant I expressible by a Presburger constraint such that $I(\mathbf{m_0})$ holds, but $I(\mathbf{m})$ does not hold. Therefore inductive Presburger invariants are in a sense a complete class of certificates for safety properties. However, there is no known upper bound on the size of constraints describing these invariants, and the proofs for their existence also do not yield a procedure to construct them other than using exhaustive enumeration.

3.5 Open Problems

For fair termination problems, there is a straightforward polynomial-time reduction to the (general) reachability problem [AH11]. Therefore a procedure for verifying safety properties can also be used to verify liveness properties. Moreover, this is possible not only for fair termination but also for a larger class of liveness properties [GM12]. If would therefore be interesting to compare our method SAFETY on the system obtained by this reduction with our method LIVENESS on the original system, and explore whether there are other interesting liveness properties not handled by LIVENESS. Further, one could investigate if the trap refinement for LIVENESS would be subsumed by the trap refinement for SAFETY in the transformed system, since both use similar constraints.

Another direction to explore would be how our method LIVENESS can be adapted to directly handle more complex liveness properties with different fairness assumptions, such as the variant of the fair termination problem in [GM12], which is equivalent to a reachability problem. With other fairness assumptions, the marking equation or T-invariants cannot directly be used, since they only capture information about what transitions are used in an infinite sequence, but not which markings are visited or which transitions are enabled along the sequence. In Chapter 4 we will see one adaption of LIVENESS to a specific different fairness constraint in the context of population protocols.

4 Verification of Population Protocols

Population protocols [Ang+06] are a model of distributed computing where anonymous agents interact via rendezvous transitions, changing only their local state during the pairwise interactions, with the goal of deciding a property of their initial configuration. The series of interactions is guided by a scheduler and defines the executions of the protocol. A scheduler selects pairs of agents arbitrarily, but must do so in a fair manner, where it may not avoid certain interactions forever. This is similar to a stochastic scheduler as in that any configuration that is always reachable will eventually be reached. This strong fairness assumption is one key property of population protocols in comparison to other distributed systems. Another property is that, while there may be an arbitrary number of agents initially, no agents may be created or destroyed by interactions, and thus the size of a configuration along an execution remains constant.

An important problem for population protocols is deciding whether a protocol correctly computes some given predicate. Using the close relation of population protocols to Petri nets, Esparza et al. [Esp+17] have shown that this problem is decidable, but also as hard as the reachability problem. This means that any complete decision procedure is bound to need a non-elementary amount of memory on some protocols. Therefore as in Chapter 3, we investigate incomplete decision procedures for these problems. We also give properties of the class of protocols where our procedures are complete.

Example

Before formally introducing population protocols, we give a *majority protocol* as an example, shown in Figure 4.1 in form of a Petri net. Each agent of the protocol can be in one of the four states A, B, a, b, shown as places. The possible pairwise interactions

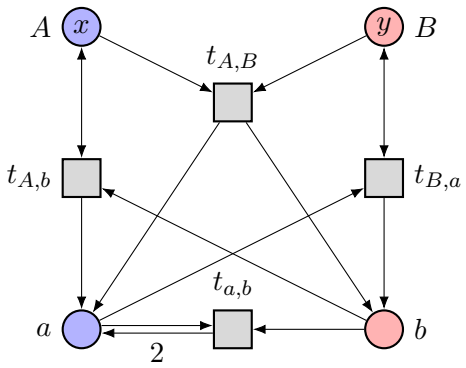


Fig. 4.1 Majority protocol shown in form of a Petri net.

are given by the transitions of the net, which move agents in the states of their preset to the states in their postset. Agents in states A and a have output "yes" (indicated in blue) and agents in states B and b have output "no" (indicated in red). Further, we consider agents in states A and B as *active* and agents in states a and b as *passive*.

The goal of the agents in the protocol is to compute whether $x \geq y$ holds for an input $x, y \in \mathbb{N}$ by reaching a common consensus on the answer given by their output. Initially, x agents are placed in state A and y agents are placed in state B. In a fair execution of the protocol, eventually all active agents of opposing opinions will interact via transition $t_{A,B}$ and become passive. If an active agent remains, then we had a majority of x or y, and this agent will convert all passive agents to the correct output via transitions $t_{A,b}$ or $t_{B,a}$. If no active agent is remaining, then $x = y$ held initially, and transition $t_{a,b}$ will move all passive agents to a. In all cases, we will reach a configuration where all agents have the same output and no transition will change the output again. By our reasoning, we can show that for any fair execution and any numbers x and y, all agents will eventually agree on the correct output for whether $x \geq y$ holds. Therefore the protocol correctly computes $x \geq y$.

Formal definition

For a set A, denote by $A^{\langle 2 \rangle} \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbb{N}^A \mid |\mathbf{x}| = 2\}$ the set of vectors in $\mathbb{N}^A$ of size 2. A *population protocol* is a tuple $\mathcal{P} = (Q, T, X, I, O)$, where Q is a finite set of states, $T \subseteq Q^{\langle 2 \rangle} \times Q^{\langle 2 \rangle}$ is a set of transitions with $(\mathbf{x}, \mathbf{x}) \in T$ for all $\mathbf{x} \in Q^{\langle 2 \rangle}$, X is the finite input alphabet, $I : X \to Q$ is the input mapping, and $O : Q \to \mathbb{B}$ is the output function. A *configuration* of a population protocol is a vector $\mathbf{c} \in \mathbb{N}^Q$ with $|\mathbf{c}| \geq 2$, interpreted as a multiset of agents with $\mathbf{c}(q)$ agents in a state q. A transition $(\mathbf{x}, \mathbf{y}) \in T$ is *silent* if $\mathbf{x} = \mathbf{y}$.

We associate each population protocol $\mathcal{P} = (Q, T, X, I, O)$ with the Petri net $N_\mathcal{P} = (Q, T, F)$ where $F(q, (\mathbf{x}, \mathbf{y})) = \mathbf{x}(q)$ and $F((\mathbf{x}, \mathbf{y}), q) = \mathbf{y}(q)$ for each $q \in Q$ and $(\mathbf{x}, \mathbf{y}) \in T$. As configurations are markings of $N_\mathcal{P}$, we lift all reachability relations on markings of $N_\mathcal{P}$ to configurations, as well as the notions for presets and postsets to states and transitions of $\mathcal{P}$.

Let $\mathcal{P} = (Q, T, X, I, O)$ be a population protocol. We associate to every input $\mathbf{x} \in \mathbb{N}^X$ the vector $I(\mathbf{x}) \in \mathbb{N}^Q$ with $I(\mathbf{x}) \stackrel{\text{def}}{=} \sum_{x \in X} \mathbf{x}(x) \cdot \wr I(x) \wr$. A configuration $\mathbf{c}$ of $\mathcal{P}$

- is *initial* if $\mathbf{c} = I(\mathbf{x})$ for some input $\mathbf{x} \in \mathbb{N}^X$;
- has *consensus* $b \in \mathbb{B}$ if $O(q) = b$ for all $q \in [\![\mathbf{c}]\!]$;
- is *bottom* if for all $\mathbf{c}'$ with $\mathbf{c} \stackrel{*}{\to} \mathbf{c}'$, we have $\mathbf{c}' \stackrel{*}{\to} \mathbf{c}$; and
- is *terminal* if for all $\mathbf{c}'$ with $\mathbf{c} \to \mathbf{c}'$, we have $\mathbf{c}' = \mathbf{c}$.

We define the output of a configuration $\mathbf{c}$ by the function $O : \mathbb{N}^Q \to \mathbb{B} \cup \{\bot\}$ where $O(\mathbf{c}) = b$ if $\mathbf{c}$ has consensus b and otherwise $O(\mathbf{c}) = \bot$.

An *execution* of a population protocol is an infinite sequence π of configurations where $\pi(i) \to \pi(i+1)$ for all $i \in [\omega]$. An execution π *stabilizes* to an output $b \in \mathbb{B}$ if $O(\mathbf{c}) = b$ for all $\mathbf{c} \in \mathrm{Inf}(\pi)$. An execution π is *fair* if $\mathrm{Inf}(\pi) = \{C' \mid \exists C \in \mathrm{Inf}(\pi) : C \to C'\}$, i.e. every configuration which could be reached by a single step infinitely often is also reached infinitely often.

A protocol *computes* a predicate $\varphi : \mathbb{N}^X \to \mathbb{B}$ if for every input $\mathbf{x} \in \mathbb{N}^X$, every fair execution π with $\pi(1) = I(\mathbf{x})$ stabilizes to $\varphi(\mathbf{x})$. A protocol is *well-specified* if it computes some predicate.

Population protocols compute exactly the Presburger predicates [Ang+06], i.e. for every Presburger predicate φ, there is a population protocol that computes φ, and every predicate computed by some population protocol is a Presburger predicate. We use this to restrict ourselves to Presburger constraints in the following definition of our two central decision problems:

Definition 4.1 (Well-specification problem)**.**

Given: A population protocol $\mathcal{P}$.

Decide: Is $\mathcal{P}$ well-specified?

Definition 4.2 (Correctness problem)**.**

Given: A population protocol $\mathcal{P}$ and a Presburger constraint φ.

Decide: Does $\mathcal{P}$ compute φ?

Existing results for verification of population protocols

The line of research for automatic verification of population protocols was started by Esparza et al. [Esp+15], presented in more detail in [Esp+17]. The authors show that the well-specification and correctness problems are decidable, but as hard as the reachability problem for Petri nets. For the *complements* of both problems, they give a reduction to and a polynomial reduction from the reachability problem. We give a short excerpt of the results for the reduction to reachability in [Esp+17], on which we base our results.

Denote by $\mathcal{B}$ the set of bottom configurations of a population protocol. A population protocol is *not* well-specified if and only if there is an initial configuration $\mathbf{c_0}$ and two bottom configurations $\mathbf{c_1}, \mathbf{c_2} \in \mathcal{B}$ such that:

$$\mathbf{c_0} \xrightarrow{*} \mathbf{c_1} \wedge \mathbf{c_0} \xrightarrow{*} \mathbf{c_2} \wedge (O(\mathbf{c_1}) = \bot \vee O(\mathbf{c_2}) = \bot \vee O(\mathbf{c_1}) \neq O(\mathbf{c_2})) \tag{4.3}$$

A population protocol is *not* correct with respect to a predicate φ if and only if there is an input $\mathbf{x} \in \mathbb{N}^X$, an initial configuration $\mathbf{c_0}$ with $I(\mathbf{x}) = \mathbf{c_0}$ and a bottom configuration $\mathbf{c_1} \in \mathcal{B}$ such that:

$$\mathbf{c_0} \xrightarrow{*} \mathbf{c_1} \wedge O(\mathbf{c_1}) \neq \varphi(\mathbf{x}) \tag{4.4}$$

The intuition behind these properties is that if we visit a bottom configuration along a fair execution, then by fairness we will also infinitely often visit that configuration again. Further every fair execution will eventually visit some bottom configuration. Therefore whether a fair execution will stabilize to some output is completely defined by the outputs of bottom configurations reachable from its initial configuration.

4.1 Problem Statement

We investigate incomplete decision procedures for the well-specification problem (Definition 4.1) and correctness problem (Definition 4.2) for population protocols, based on the results in [Esp+17]. We want to evaluate the degree of completeness of the procedure, both practically on existing protocols and theoretically, i.e. investigate the properties of the class of protocols for which the procedure is successful.

Since before starting the work in [Blo+17] presented in this chapter, there was no existing implementation of either a complete or incomplete decision procedure for these problems, another goal was to develop a first tool implementing such a procedure. Such a tool can then provide the basis for development of further techniques for the analysis of population protocols.

4.2 New Contributions

In [Blo+17], we present incomplete decision procedures for the well-specification and correctness problems, based on the characterizations stated in the properties (4.3) and (4.4). A direct translation of these properties faces two difficulties. First, to check $\mathbf{c_0} \xrightarrow{*} \mathbf{c_1}$, we have that the reachability relation for population protocols, as for Petri nets, is in general not semi-linear [Ang+07; HP79], and relying on solving the reachability problem for Petri nets has an inhibiting complexity. Second, we would need a representation of the set $\mathcal{B}$ of bottom configurations. While [Esp+17] shows that $\mathcal{B}$ is semi-linear and effectively constructible, a Presburger constraint describing this set might be very large: [BF97; Ler13] show that $\mathcal{B}$ is definable by a constraint of size k-exponential in the size of the protocol for some constant k, i.e. we only have an elementary upper bound. No lower bound is known for population protocols, but we conjecture an exponential lower bound due to EXPSPACE-hardness results for the related problems of reversible reachability [Ler13] and cyclicity of markings [DL15] in Petri nets.

For solving the first problem for reachability, we revisit the constraint-based methods for our procedure Safety from Section 3.2. This gives us a semi-linear overapproximation of the reachability relation. We then use this to show that only configurations having the correct consensus are reachable. These methods are called StrongConsensus for well-specification and Strong-φ-Consensus for correctness.

For solving the second problem of a representation of the bottom configurations $\mathcal{B}$, we restrict ourselves to *silent* protocols. These are protocols where all fair executions eventually only visit terminal configurations, and where thus all bottom configurations are already terminal. We then need to show the liveness property that the protocol is actually silent. Here we also use an overapproximation, and revisit the procedure LIVENESS from Section 3.3. We further introduce a notion called *layers* to incorporate the fairness assumptions for population protocols. This method is called LAYEREDTERMINATION.

Both methods translate to checking satisfiability of Presburger constraints, for which we employ an SMT solver. From positive results, we can then infer that well-specification or correctness holds.

4.2.1 Layered Termination

We start by presenting the procedure LAYEREDTERMINATION for showing that a protocol is silent. Silent protocols were first introduced in [DGS99], and many protocols from the literature are already silent. Formally, an execution π of a population protocol is *silent* if there exists an $i \in [\omega]$ such that $\pi(i)$ is terminal, and a population protocol $\mathcal{P}$ is silent if every fair execution of $\mathcal{P}$ is silent.

Only restricting ourselves to silent protocols does not make the well-specification or correctness problem easier. As shown in [Blo+17], the well-specification problem for silent protocols is still as hard as the reachability problem for Petri nets, and this can also be adapted to the correctness problem. However, it is easy to see that every bottom configuration of a silent protocol is terminal, which we later use to give a simple constraint describing the set $\mathcal{B}$ of bottom configurations.

To check that a protocol is silent, we also introduce an incomplete decision procedure. A first sufficient approximation would be to check that a protocol has no non-silent execution, fair or not. This is expressible as a fair termination property for our procedure LIVENESS from Section 3.3. Moreover, since there would need to be no non-silent execution from any configuration, initial or not, this would hold if and only if the Petri net for the protocol without silent transitions has no infinite transition sequence enabled at any marking. Recalling Theorem 3.22 and its characterization by a ranking function in Section 3.3.4, this can be checked in polynomial time.

However, this procedure completely ignores the fairness assumption and is usually too weak to show that a protocol is silent. For the majority protocol with the Petri net in Figure 4.1, we have $\wr A, A, B, B\wr \xrightarrow{t_{A,B}} \wr A, B, a, b\wr \xrightarrow{t_{A,b}} \wr A, B, a, a\wr \xrightarrow{t_{B,a}} \wr A, B, a, b\wr$, so the infinite sequence $t_{A,B} \cdot (t_{A,b} \cdot t_{B,a})^\omega$ produces a non-silent execution, which is however not fair as the step $\wr A, B, a, b\wr \xrightarrow{t_{A,B}} \wr a, a, b, b\wr$ is never executed. We therefore introduce a refinement of this method, where we partition the fair executions of a protocol into *layers*, inspired by the typical design of protocols in the literature.

Definition 4.5. For a population protocol $\mathcal{P} = (Q, T, X, I, O)$ and a set of transitions $U \subseteq T$, denote by $\mathcal{P}[U] \stackrel{\text{def}}{=} (Q, U \cup S, X, I, O)$ the subprotocol with transitions U, where S is the set of silent transitions of $\mathcal{P}$. A population protocol $\mathcal{P}$ satisfies *layered termination* if there is an ordered partition $T_1, T_2, \ldots, T_n$ of its transitions such that the following two properties hold for every $i \in [n]$:

(a) Every (fair or unfair) execution of $\mathcal{P}[T_i]$ is silent.

(b) For any configurations $\mathbf{c}, \mathbf{c}'$, if $\mathbf{c} \xrightarrow{*}_{T_i} \mathbf{c}'$ and $\mathbf{c}$ is terminal in $\mathcal{P}[T_1 \cup T_2 \cup \ldots \cup T_{i-1}]$, then $\mathbf{c}'$ is also terminal in $\mathcal{P}[T_1 \cup T_2 \cup \ldots \cup T_{i-1}]$.

In other words, the first property denotes that there is no non-silent execution only containing transitions from a single layer, and the second property states that transitions of a layer may not re-enable transitions of an earlier layer. Now if a protocol satisfies layered termination, then every fair execution will eventually disable all transitions of the first layer, since by property (a) we can and by fairness we will always reach a terminal configuration of that layer. By property (b) then the transitions of that layer will stay disabled. Iterating this process, eventually all non-silent transitions are disabled. Therefore the protocol is silent.

Our example protocol for majority in Figure 4.1 satisfies layered termination with the two layers $T_1 = \{t_{A,B}, t_{B,a}\}$ and $T_2 = \{t_{A,b}, t_{a,b}\}$. Every transition of T_1 reduces the total number of agents in A, B and a, and every transition of T_2 reduces the number of agents in b. Thus every execution of $\mathcal{P}[T_1]$ or $\mathcal{P}[T_2]$ is silent. The set of configurations terminal in $\mathcal{P}[T_1]$ are those where there is no agent in B or no agent in A and a. In both cases, transitions of T_1 remain disabled by transitions of T_2.

Theorem 4.6 ([Blo+17])**.** *Deciding if a population protocol satisfies layered termination is in NP.*

The NP procedure for layered termination first guesses the numbers of layers $n \in [|T|]$ and then the ordered partition $T_1, T_2, \ldots, T_n$. It then verifies properties (a) and (b) of Definition 4.5 in polynomial time. For property (a), we use the characterization of the method LIVENESS for non-existence of an infinite transition sequence through satisfiability of the constraint (3.34), instantiated for the appropriate Petri net of the subprotocol for this layer without silent transitions. For property (b), we show in [Blo+17] that it holds for a given layer T_i if and only if for all $t \in T_i$ and non-silent $u \in T_1 \cup T_2 \cup \ldots \cup T_{i-1}$, there exists $u' \in T_1 \cup T_2 \cup \ldots \cup T_{i-1}$ such that we have:

$$ {}^\bullet u' \leq {}^\bullet t + ({}^\bullet u \ominus t^\bullet) \tag{4.7} $$

Above constraint is derived from the characterization that a configuration $\mathbf{c}$ enables $t \cdot u$ iff $\mathbf{c} \geq {}^\bullet t + ({}^\bullet u \ominus t^\bullet)$, and $\mathbf{c}$ is not terminal in $\mathcal{P}[T_1 \cup T_2 \cup \ldots \cup T_{i-1}]$ iff $\mathbf{c} \geq {}^\bullet u'$ for some $u' \in T_1 \cup \ldots \cup T_{i-1}$. By elimination of $\mathbf{c}$, constraint (4.7) then shows that in any terminal configuration no $t \in T_i$ can re-enable a transition u from a previous layer.

Overall, we construct a constraint as follows to check if a population protocol satisfies layered termination with a given number $n \in [|T|]$ of layers. We introduce a vector $\mathbf{b} \in \mathbb{N}^T$ representing the partition such that $T_i = \{t \in T \mid \mathbf{b}(t) = i\}$ for each $i \in [n]$, and for each $i \in [n]$ introduce a vector $\mathbf{y}_i \in \mathbb{N}^P$ for the constraint (3.34) of the ranking function for the layer T_i. Denote by NS the set of non-silent transitions of the protocol. We then obtain the constraint below, where the second line corresponds to property (b) by the characterization through (4.7) and the third line to property (a) by the characterization through (3.34):

$$\begin{aligned} &\bigwedge_{t \in T} (1 \leq \mathbf{b}(t) \leq n) \\ \wedge &\bigwedge_{\substack{t \in T \\ u \in NS}} \left(\mathbf{b}(t) > \mathbf{b}(u) \to \bigvee_{u' \in T} (\mathbf{b}(t) > \mathbf{b}(u') \wedge {}^\bullet u' \leq {}^\bullet t + ({}^\bullet u \ominus t^\bullet)) \right) \\ \wedge &\bigwedge_{i=1}^{n} \left(\mathbf{y}_i \geq \mathbf{0} \wedge \bigwedge_{t \in NS} (\mathbf{b}(t) = i \to \mathbf{y}_i \cdot \Delta(t) \leq -1) \right) \end{aligned} \tag{4.8}$$

The constraint is satisfiable if and only if the protocol satisfies layered termination with n layers. Since most protocols that satisfy layered termination do so with at most two layers, we check the constraint for increasing $n = 1, 2, \ldots, |T|$ until we can conclude whether layered termination holds. This is then the procedure LayeredTermination.

4.2.2 Strong Consensus

We now return to checking well-specification or correctness by the characterization through (4.3) and (4.4), but under the assumption that the given population protocol is silent. As noted in the preceding section, in this case the set of bottom configurations equals the set of terminal configurations. A configuration is terminal if and only if only silent transitions are enabled. We apply this to first define some utility constraints over a configuration $\mathbf{c} \in \mathbb{N}^Q$ and an input $\mathbf{x} \in \mathbb{N}^X$, which we will use later:

$$\begin{aligned} \mathrm{Init}(\mathbf{x}, \mathbf{c}) &\stackrel{\text{def}}{=} |\mathbf{c}| \geq 2 \wedge I(\mathbf{x}) = \mathbf{c} && (\mathbf{c} \text{ is initial and corresponds to the input } \mathbf{x}) \\ \mathrm{Term}(\mathbf{c}) &\stackrel{\text{def}}{=} \bigwedge_{t \in NS} \neg(\mathbf{c} \geq {}^\bullet t) && (\mathbf{c} \text{ is terminal}) \\ \mathrm{Out}_0(\mathbf{c}) &\stackrel{\text{def}}{=} \sum_{q \in O^{-1}(1)} \mathbf{c}(q) = 0 \\ \mathrm{Out}_1(\mathbf{c}) &\stackrel{\text{def}}{=} \sum_{q \in O^{-1}(0)} \mathbf{c}(q) = 0 && (\mathbf{c} \text{ has output } b \in \{0, 1, \bot\}) \\ \mathrm{Out}_\bot(\mathbf{c}) &\stackrel{\text{def}}{=} \neg\mathrm{Out}_0(\mathbf{c}) \wedge \neg\mathrm{Out}_1(\mathbf{c}) \end{aligned}$$

Further assume that we have a Presburger constraint $\mathrm{PotReach}(\mathbf{c}, \mathbf{c}')$ describing an overapproximation of the reachability relation, i.e. $\mathrm{PotReach}(\mathbf{c}, \mathbf{c}')$ implies $\mathbf{c} \stackrel{*}{\to} \mathbf{c}'$ for all configurations $\mathbf{c}, \mathbf{c}' \in \mathbb{N}^Q$. We will later define this constraint.

Now, for a given silent population protocol, we construct the following constraint as a weaker version of (4.3), which, if unsatisfiable, implies that the protocol is well-specified:

$$\begin{aligned}&\mathrm{Init}(\mathbf{x}, \mathbf{c_0}) \wedge \mathrm{Term}(\mathbf{c_1}) \wedge \mathrm{Term}(\mathbf{c_2}) \wedge \mathrm{PotReach}(\mathbf{c_0}, \mathbf{c_1}) \wedge \mathrm{PotReach}(\mathbf{c_0}, \mathbf{c_2}) \\ &\wedge \left(\mathrm{Out}_\perp(\mathbf{c_1}) \vee \mathrm{Out}_\perp(\mathbf{c_2}) \vee \bigvee_{b \in \mathbb{B}} (\mathrm{Out}_b(\mathbf{c_1}) \wedge \neg \mathrm{Out}_b(\mathbf{c_2})) \right)\end{aligned} \tag{4.9}$$

If we are further given a predicate φ over the possible inputs, then we construct the following constraint as a weaker version of (4.4), which, if unsatisfiable, implies that the protocol is correct with respect to φ:

$$\mathrm{Init}(\mathbf{x}, \mathbf{c_0}) \wedge \mathrm{Term}(\mathbf{c_1}) \wedge \mathrm{PotReach}(\mathbf{c_0}, \mathbf{c_1}) \wedge \bigvee_{b \in \mathbb{B}} (\varphi(\mathbf{x}) = b \wedge \neg \mathrm{Out}_b(\mathbf{c_1})) \tag{4.10}$$

We require that the predicate φ is given as a constraint such that both $\varphi(\mathbf{x}) = 0$ and $\varphi(\mathbf{x}) = 1$ can be translated to existential Presburger constraints of polynomial size. In this case, above constraints are also existential Presburger constraints. In [Blo+17], we use predicates given as Boolean combinations of *threshold* constraints, which are our linear constraints, and *remainder* constraints, which are constraints of the form $\mathbf{a} \cdot \mathbf{x} \equiv c \pmod m$ for some vector $\mathbf{a} \in \mathbb{N}^A$ and numbers $c, m \in \mathbb{N}$ with $m \geq 2$. This is sufficient to express all Presburger predicates.

It remains to define the constraint $\mathrm{PotReach}(\mathbf{c}, \mathbf{c}')$. For this we revisit the method Safety from Section 3.2, using the marking equation with refinement by U-traps and U-siphons. Ignoring the safety property, e.g. by setting it to false, the constraint $\mathrm{PotReach}(\mathbf{c}, \mathbf{c}')$ should hold if and only if the constraint (3.10) is satisfiable with $\mathbf{m_0} = \mathbf{c}$, $\mathbf{m} = \mathbf{c}'$, the set $\mathcal{Q}$ of all U-traps and the set $\mathcal{R}$ of all U-siphons of the Petri net associated to the protocol.

Using this definition, we say that a protocol where constraint (4.9) is unsatisfiable satisfies *strong consensus*, and a protocol where (4.10) is unsatisfiable for a predicate φ satisfies *strong-φ-consensus*.

Theorem 4.11 ([Blo+17]). *Checking whether a protocol satisfies strong consensus or strong-φ-consensus is in co-NP.*

For this result, we cannot use a direct representation of the constraint for PotReach, since the Petri net of the protocol may have an exponential number of U-traps or U-siphons. We prove it instead by giving an NP procedure for the complement that starts by guessing $[\![\mathbf{c}]\!], [\![\mathbf{c}']\!], [\![\mathbf{x}]\!]$ and first checks the U-trap and U-siphon property with $U = [\![\mathbf{x}]\!]$ using the polynomial algorithm from [DE95] briefly described in Section 3.2.3, and then proceeds to check satisfiability of the marking equation together with the remaining constraints of (4.9) or (4.10) for vectors with these supports.

Above result gives a reduction to a polynomially sized Presburger constraint. However a direct translation of the algorithm in [DE95] would produce a quadratically sized constraint, and since we expect to need few U-traps or U-siphons in practice, explicitly adding them could produce a smaller constraint and be more efficient. Therefore we use an iterative approach to find U-traps and U-siphons and construct PotReach as described for the method SAFETYBYREFINEMENT in Section 3.2.3. We use all four refinement steps with U-traps and U-siphons, adding them iteratively to show that PotReach$(\mathbf{c}, \mathbf{c}')$ does not hold for previous solutions to our constraints (4.9) or (4.10), or giving up if no refinement is found. If the constraint is unsatisfiable, then we know that well-specification or correctness, respectively, holds. These incomplete decision procedures are called STRONGCONSENSUS and STRONG-φ-CONSENSUS, respectively.

As an example, consider the majority protocol as shown in Figure 4.1. Assume we want to prove correctness in the case where the predicate $x \geq y$ holds. We then have $\mathbf{c_0}(A) \geq \mathbf{c_0}(B)$ and $\mathbf{c_0}(a) = \mathbf{c_0}(b) = 0$ for any initial configuration $\mathbf{c_0} = I(x, y)$. With the marking equation, we can show that in any terminal configuration $\mathbf{c_1}$ reachable from $\mathbf{c_0}$ we have $\mathbf{c_1}(B) = 0$ and $\mathbf{c_1}(a) = 0 \vee \mathbf{c_1}(b) = 0$. For a possible satisfying assignment $\mathbf{c_0} = \{\!|A, B|\!\}$ and $\mathbf{c_1} = \{\!|b, b|\!\}$ our procedure finds the trap $\{B, a\}$. Together we then have that either initially $\mathbf{c_0}(B) = 0$ and so $\mathbf{c_0}$ is already terminal with the correct consensus, or $\mathbf{c_0}(B) \geq 1$, but then by the trap $\mathbf{c_1}(a) \geq 1$ for any terminal configuration $\mathbf{c_1}$ reachable from $\mathbf{c_0}$, which further implies $\mathbf{c_1}(b) = 0$, so we also have the correct consensus. The case where the predicate does not hold is similar, and our procedure succeeds in proving correctness for the majority protocol.

4.2.3 The Class of WS^3-protocols

We introduce the class WS^3 of population protocols, where a protocol belongs to WS^3 if it satisfies both layered termination and strong consensus. Therefore every WS^3-protocol is well-specified. We use the complexity class DP [PY84] defined as $\mathsf{DP} = \{L_1 \cap L_2 \mid L_1 \in \mathsf{NP} \wedge L_2 \in \text{co-}\mathsf{NP}\}$. From Theorems 4.6 and 4.11 we have:

Theorem 4.12 ([Blo+17])**.** *The membership and correctness problems for WS^3-protocols are in DP.*

By giving WS^3-protocols for threshold and remainder predicates and constructions for negation and conjunction preserving membership in WS^3, we show the following in [Blo+17]:

Theorem 4.13. *Every Presburger predicate is computed by some WS^3-protocol.*

Therefore the class WS^3 of population protocols has a membership and correctness problem with a reasonable complexity (DP) and is as expressive as the the class of all well-specified protocols, where the membership problem is at least TOWER-hard.

4.2.4 Experimental Evaluation

We implemented the procedures LAYEREDTERMINATION, STRONGCONSENSUS and STRONG-φ-CONSENSUS in a tool called PEREGRINE. It takes as input a population protocol and an optional Presburger constraint φ, and sequentially invokes LAYEREDTERMINATION and either STRONGCONSENSUS or STRONG-φ-CONSENSUS to check well-specification or correctness of the protocol.

We evaluate PEREGRINE on a set of benchmarks from protocols in the literature, most of them parameterized by values of constants in their predicates and thus defining a family of protocols. The first two families are the *threshold* and *remainder* protocols of [Ang+06] for the predicates $\mathbf{a} \cdot \mathbf{x} \geq c$ and $\mathbf{a} \cdot \mathbf{x} = c \pmod m$, respectively, where we denote by $\ell_{\max} \stackrel{\text{def}}{=} (|\mathbf{a}(a_1)|, \ldots, |\mathbf{a}(a_k)|, |c| + 1)$ the maximal value in the constraint. We further have two families for *flock of birds* protocols from [CMS10] and [Clé+11] with the predicate $x \geq c$. Finally we analyze the majority protocol in our example from [AAE06] and the broadcast protocol from [Clé+11], both not parameterized.

All experiments were performed on the same machine, equipped with an Intel Core i7-4810MQ CPU at 2.8 GHz and 16 GB of memory. Execution time was limited to 1 hour and memory to 16 GB. Table 4.2 shows the results of the experimental evaluation, originally conducted in [Blo+17], for showing well-specification by LAYEREDTERMINATION and STRONGCONSENSUS. Generally, STRONGCONSENSUS is faster than LAYEREDTERMINATION, except for the flock of birds protocol from [Clé+11] where we need a linear number of refinement steps with U-traps. We also evaluated showing correctness by LAYEREDTERMINATION and STRONG-φ-CONSENSUS. This is successful for all our examples and faster than showing well-specification except for the remainder protocol, where we exceed the time limit for $m = 70$. The evaluation shows that many standard protocols from the literature belong to the class WS^3, where we can show well-specification and correctness for protocols with hundreds of states and thousands of transitions within an hour.

4.3 Related Work

Previous work on verification of population protocols usually falls into one of two categories: verification only for a finite number of inputs [Clé+11; CMS10; PLD08; Sun+09], or verification using an interactive theorem prover [DM09]. For a fixed finite number of inputs, a protocol can be verified by a model checker or by construction of the finite graph of reachable configurations. Such an approach can show partial correctness or find a counterexample, but fails to completely verify a protocol for all possible inputs. Using an interactive theorem prover, in [DM09] some leader election protocols are verified completely. This approach is only semi-automatic and requires human interaction for every new protocol. In contrast, PEREGRINE is the first tool that is able to automatically verify a large class of protocols for all possible inputs.

Threshold [Ang+06]			
$\ell_{\max}$	$\lvert Q\rvert$	$\lvert T\rvert$	Time
3	28	288	8.0
4	36	478	26.5
5	44	716	97.6
6	52	1002	243.4
7	60	1336	565.0
8	68	1718	1019.7
9	76	2148	2375.9
10	84	2626	TO

Remainder [Ang+06]			
m	$\lvert Q\rvert$	$\lvert T\rvert$	Time
10	12	65	0.4
20	22	230	2.8
30	32	495	15.9
40	42	860	79.3
50	52	1325	440.3
60	62	1890	3055.4
70	72	2555	3176.5
80	82	3320	TO

Flock of birds [CMS10]			
c	$\lvert Q\rvert$	$\lvert T\rvert$	Time
20	21	210	1.5
25	26	325	3.3
30	31	465	7.7
35	36	630	20.8
40	41	820	106.9
45	46	1035	295.6
50	51	1275	181.6
55	56	1540	TO

Flock of birds [Clé+11]			
c	$\lvert Q\rvert$	$\lvert T\rvert$	Time
50	51	99	11.8
100	101	199	44.8
150	151	299	369.1
200	201	399	778.8
250	251	499	1554.2
300	301	599	2782.5
325	326	649	3470.8
350	351	699	TO

Majority [AAE06]			
	$\lvert Q\rvert$	$\lvert T\rvert$	Time
	4	4	0.1

Broadcast [Clé+11]			
	$\lvert Q\rvert$	$\lvert T\rvert$	Time
	2	1	0.1

Tab. 4.2 Results of the experimental evaluation for PEREGRINE. $|Q|$ denotes the number of states, $|T|$ denotes the number of non-silent transitions, and the time is the execution time to prove membership for WS^3 in seconds, where further TO denotes exceeding the time limit.

After publication of [Blo+17], Blondin et al. [BEK18] present a method to analyze the expected termination time of population protocols. This can be used to show that a protocol is silent. The work introduces *stage graphs*, which are directed acyclic graphs where the nodes, called stages, describe sets of configurations closed under reachability, and which are further equipped with a certificate showing that a successor stage will be reached along all fair executions starting in that stage. While layered termination can be regarded as a specific instance of a linear stage graph, in general stage graphs also allow splitting into different successors and description of stages using more general constraints than only disabled transitions. They also allow derivation of asymptotic bounds on the expected number of interactions until termination from the certificates of the stages. However the procedure to construct stage graphs used in [BEK18] can be slower than directly testing layered termination.

In [Blo+20b] the authors, including the thesis author, extend the approach using stage graphs from [BEK18] to show correctness of population protocols. By making use of the results from [Esp+17], they derive a sound and complete procedure to show correctness of population protocols through stage graphs where all stages and certificates are described with Presburger constraints. Essentially, such stage graphs are certificates for correctness. The authors also give a procedure to effectively construct a Presburger stage graph which incorporates and generalizes the methods LayeredTermination and Strong-φ-Consensus from [Blo+17]. The procedure remains incomplete, but can verify a larger class of protocols, including non-silent protocols.

The tool Peregrine has also been extended after the initial release in [Blo+17], resulting in two tool papers [BEJ18b; Esp+20]. In the first paper [BEJ18b], Peregrine is equipped with a graphical interface for specifying and analyzing population protocols. The verification method using Strong-φ-Consensus is extended with diagnosis in case of failure due the protocol not being correct. In this case, reachability of configurations returned from a solution of the constraint (4.10) is verified using a model checker and displayed to the user as a counterexample. The second paper [Esp+20] incorporates the stage graph approach from [Blo+20b], giving users a way to visualize the stage graphs and their constraints, which can be used to further ensure correctness of a protocol.

In [Esp+18], the authors analyze immediate observation population protocols (IO protocols), a class introduced in [Ang+07]. They show that the well-specification problem for IO protocols is PSPACE-hard and in EXPSPACE. They further show in [ERW19] that the correctness problem for IO protocols is PSPACE-complete. This class is therefore another class of population protocols where the well-specification and correctness problems are easier than in the general case. IO protocols can however not compute all Presburger predicates, but only those given by certain counting constraints [Ang+07].

4.4 Open Problems

The class WS^3, while being very expressive and having a relatively efficient correctness check, is not that natural, as it is essentially defined by satisfaction of the constraints for LayeredTermination and StrongConsensus. The membership test is also already as hard as the correctness check. Here one could try to find other classes with a more natural characterization and easier membership test, e.g. by structural properties along the lines of [ERW19; Esp+18], which would still be expressive enough and have an easier correctness check.

Besides expressive power, there are also other desirable properties for a class of population protocols. One would often like to have *succinct* protocols [BEJ18a; Blo+20a], i.e. protocols having few states, or *fast* protocols [AAE08; KU18], i.e. protocols with a low expected number of interactions until a stable consensus is reached. Note that generally only certain trade-offs between space and time are possible [AAG18; Ali+17]. For example, for some predicates the only known fast protocols are non-silent [AGV15] and therefore not in WS^3. This is already partly resolved by the lifting of the techniques presented here to stage graphs in [Blo+20b], which however remains without a good characterization of the class for which it succeeds besides the constraints. While a search for a natural and expressive class that allows for fast and succinct protocols as well as an efficient correctness check is certainly daunting, some combinations of these properties could be a good starting point for future research.

For another research direction, we would like to highlight well-specification. While well-specification of a protocol is a necessary precondition for correctness, knowing that a protocol is well-specified does not directly help in further analysis. A problem to consider is the *promise correctness problem*, which is the problem of deciding correctness for a given well-specified population protocol, without needing to validate well-specification (which is the promise). The complexity of the promise correctness problem is open, especially it is unknown if it is easier or just as hard as the general correctness problem. Another related problem is the *tailor problem* in [Esp+17], which is the problem of obtaining the predicate that a given well-specified population protocol computes. Esparza et al. [Esp+17] give an algorithm for the tailor problem by constructing a certificate for well-specification. We believe that the constraints for StrongConsensus can be used as such a certificate and thus transformed to a Presburger predicate that the protocol computes. Here a further analysis and evaluation of this technique would be interesting. Algorithms for the well-specification and tailor problems would also give another algorithm for the correctness problem by checking equivalence of the computed and given Presburger predicates.

5 Quantitative Analysis of Workflow Nets

Workflow nets are a class of Petri nets used for representing and analyzing business processes [Aal98; AH02; DE00], specified e.g. as BPMN (Business Process Model and Notation), EPC (Event-driven Process Chain) or UML Activity Diagrams. We adapt our techniques and theory for analyzing Petri nets to obtain more efficient analysis techniques for workflow nets, which can then be used to obtain information about the underlying business process.

There is interest in extending business process models such as BPMN with quantitative information [SZS10] which can be used to specify costs or execution times of tasks, and further in analysis of resource requirements for workflow management systems [KAV02]. Recent work [BVT15; BVT16] uses workflow graphs, a class of workflow nets, for analysis of the number of required resources and the execution time of a workflow. In this chapter, we will further look at the complexity of computing some quantitative properties of resources and time in workflow nets.

Example

Figure 5.1 shows a simple workflow net for processing an order by a store. Essentially, a workflow net represents a procedure that starts with a token in a designated input place i and ends with a token in a designated output place o, where transitions are either associated with tasks or specify the control flow for these tasks.

The procedure for processing an order contains 5 tasks, given as transitions with execution times inscribed. After initial processing, taking 1 hour, in parallel the tasks for receiving the payment and packaging the order can be executed. In 20% of the cases, and possibly for multiple units, components of the order may be missing from

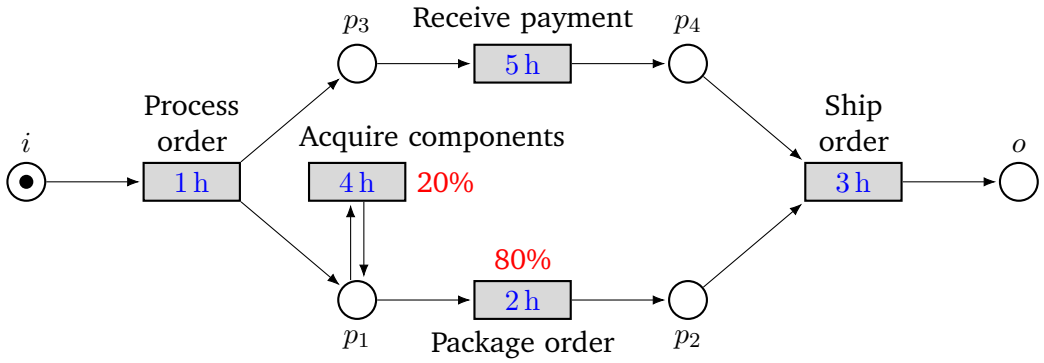


Fig. 5.1 A workflow net for processing an order with quantitative information.

the central inventory and need to be acquired from an external warehouse before packaging, incurring a delay of 4 hours. Only when the tasks for payment and packing have finished the order can be shipped.

Given such a procedure, the store owner may be interested in several questions. First, they might want to know how many employees the store needs to fulfill every order as fast as possible. Second, they might want to know what the expected duration between placement and shipping of an order is, e.g. to optimize logistics or to display this information to customers. In this case, assuming one task is fulfilled by exactly one employee over its full duration, for the first question one can see that two employees suffice, as this is the maximal amount of parallelism. The answer to the second question is already not that obvious, due to the mixing of parallel tasks and choice. We will show that both questions can be answered, but are hard for certain classes of workflow nets.

Formal definition

Definition 5.1. A *workflow net* is a tuple $\mathcal{W} = (N, i, o)$ where $N = (P, T, F)$ is a Petri net, $i \in P$ is the *input place* with ${}^\bullet i = \wr\, \wr$ and $o \in P$ is the *output place* with $o^\bullet = \wr\, \wr$, and for every $x \in P \cup T$, there is a path from i to o along arcs of N passing through x.

We identify any workflow net $\mathcal{W} = (N, i, o)$ with the the Petri net N and the system $(N, \wr i \wr)$. A *run* of a workflow net is a transition sequence σ with $\wr i \wr \xrightarrow{\sigma} \wr o \wr$. A well-designed workflow net should ensure that the system is terminating by having no livelocks or deadlocks, and terminates by putting a single token in o. This is captured by the notion of *soundness*.

Definition 5.2. A workflow $\mathcal{W} = (N, i, o)$ net is *sound* if

- for every $\mathbf{m} \in \mathcal{R}_N(\wr i \wr)$ we have $\wr o \wr \in \mathcal{R}_N(\mathbf{m})$ (*option to complete*);
- for every $\mathbf{m} \in \mathcal{R}_N(\wr i \wr)$, if $\mathbf{m} \geq \wr o \wr$, then $\mathbf{m} = \wr o \wr$ (*proper completion*); and
- for every transition t, there is a run of $\mathcal{W}$ containing t (*no dead transitions*).

Ordinary and 1-safe workflow nets

A marking $\mathbf{m}$ is *1-safe* if $\mathbf{m}(p) \leq 1$ for all $p \in [\![\mathbf{m}]\!]$. A system is 1-safe if all its reachable markings are 1-safe. A Petri net is *ordinary* if the weight of every arc is 1.

In this chapter we will only consider 1-safe workflow nets. Workflow nets that are not 1-safe can have a rather complex timed execution semantics and notion for schedules, which assign transitions along a run starting times respecting their causal dependencies. In a non-1-safe net, an enabled transition might need to distinguish tokens in its preset based on their age, which would require identity of tokens, and we would further need semantics to resolve multi-enabled transitions [BLR13].

Under the assumption that the system is 1-safe, it also suffices to consider only ordinary nets, as every sound 1-safe workflow net must be ordinary.

Free-choice workflow nets

A Petri net $N = (P, T, F)$ is *free-choice* if it is ordinary and $[\![{}^\bullet t_1]\!] \cap [\![{}^\bullet t_2]\!] \neq \emptyset$ implies ${}^\bullet t_1 = {}^\bullet t_2$ for all $t_1, t_2 \in T$. In a free-choice Petri net N, we have the essential property that in any marking $\mathbf{m}$ of N, if $\mathbf{m}$ enables a transition $t \in [\![p^\bullet]\!]$ of a place p, then $\mathbf{m}$ enables all transitions in $[\![p^\bullet]\!]$. Therefore we are *free to choose* the successor transition locally for the token at p.

In this chapter we will often restrict ourselves to free-choice workflow nets. These are especially interesting as many related models for business processes are essentially equivalent to free-choice workflow nets, e.g. workflow graphs [FFV15]. Further, using the theory for free-choice Petri nets, one can derive efficient decision procedures for many problems [Aal97; DE95]. In particular, soundness of free-choice workflow nets and reachability in sound free-choice workflow nets are decidable in polynomial time, and every sound free-choice workflow net is 1-safe.

We note that if one lifts the restriction on a free-choice net to be ordinary and only requires $[\![{}^\bullet t_1]\!] \cap [\![{}^\bullet t_2]\!] \neq \emptyset$ to imply ${}^\bullet t_1 = {}^\bullet t_2$ for all $t_1, t_2 \in T$, then one obtains *equal-conflict* nets with partly similar properties [HDK16; TS93; TS96].

Other classes of Petri nets

In this chapter we will also consider other structural subclasses of Petri nets and workflow nets. A Petri net $N = (P, T, F)$ is a *marked graph* if $|{}^\bullet p| \leq 1$ and $|p^\bullet| \leq 1$ for all places $p \in P$, and is *acyclic* if there is no directed cycle along arcs of N. A Petri net that is not acyclic is *cyclic*. Marked graph Petri nets may be cyclic, but every sound marked graph workflow net is acyclic.

Timed and probabilistic workflow nets

We introduce workflow nets extended with quantitative information about time and probabilities. The formal semantics will be given later as part of our contribution, since they require introduction of some basic models to define concurrency, conflict and probabilities, and also a careful consideration of assumptions is necessary for a sensible semantics.

Definition 5.3. A *timed workflow net (TWN)* is a tuple $(\mathcal{W}, \tau)$ where $\mathcal{W}$ is a 1-safe workflow net and $\tau : T \to \mathbb{N}$ associates an *execution time* to each transition of $\mathcal{W}$.

We will call transitions $t \in T$ with $\tau(t) > 0$ also *tasks* of the workflow. Typically, tasks of a workflow are associated with *resources* (e.g. CPU cores, machines, persons or departments) which are responsible for executing that task. The number of tasks

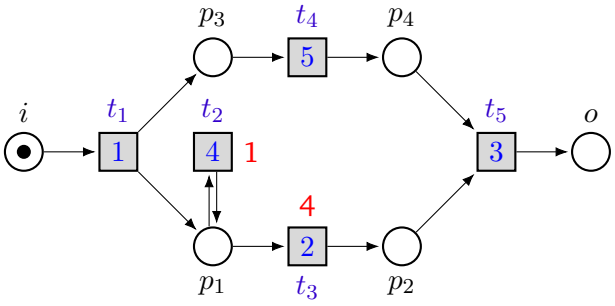


Fig. 5.2 A timed probabilistic workflow net (TPWN)

that can be executed in parallel is then limited by the number of available resources. Therefore, the execution time of a run depends not only depends on the execution times given for the transitions of the run, but also on the number of resources and possible schedules assigning tasks to resources.

Definition 5.4. A *timed probabilistic workflow net (TPWN)* is a tuple $(\mathcal{W}, \tau, w)$ where $(\mathcal{W}, \tau)$ is a TWN and $w : T \to \mathbb{Q}$ associates a *weight* $w(t) > 0$ to each transition $t \in T$.

Probabilities in a TPWN are given by weights, and used to resolve choices between conflicting transitions along a run. Workflow nets are often derived by abstracting over many concrete cases of a process. These cases correspond to runs of the net, from which probabilities can be derived and then used to predict behavior of future cases. In combination with time, this leads to the analysis of the expected execution time.

Figure 5.2 shows a TPWN obtained from the workflow net in Figure 5.1 with renamed transitions, an abstract time unit and probabilities translated to weights. Transitions t are inscribed with their execution time $\tau(t)$ in blue and annotated with the weight $w(t)$ in red. Transitions without annotations have weight 1. For example, the sequence $t_1 \cdot t_2 \cdot t_4 \cdot t_3 \cdot t_5$ is a run of the workflow net. If we can schedule the sequential parts for t_2, t_3 and t_4 in parallel using two resources, the run can be executed with time $\tau(t_1) + \max(\tau(t_2) + \tau(t_3), \tau(t_4)) + \tau(t_5) = 10$. We will later define the probability for this run to be $\frac{w(t_2)}{w(t_2)+w(t_3)} \cdot \frac{w(t_3)}{w(t_2)+w(t_3)} = \frac{4}{25}$ due to the choice between t_2 and t_3 for the token at p_1 twice along the sequence.

Remark on presentation of papers

We note that in [MEV18a], we present timed workflow nets with execution times associated to places instead of transitions. For uniformity of our presentation of both [MEV18a] and [MEO19a], we present all results here only for the model with execution times associated to transitions. Both models can easily be transformed into another with only a linear blowup by introducing intermediate places or transitions. Therefore all complexity results presented here hold for both models.

5.1 Problem Statement

For timed workflow nets, we want to give semantics to execution times of runs in relation with the number of resources. We then analyze the problem of computing the minimal number of resources needed to achieve the minimal execution time. We also examine under which conditions a schedule can achieve this execution time with this number of resources. Since a higher number of resources might relax the assumptions on the schedule, we also examine related measures to the number of resources.

For timed probabilistic workflow nets, we start with the assumption that we have a sufficient amount of resources to execute every run in minimal time. We then want to give suitable semantics to the probability distribution of runs. With this, we analyze the problem of computing the expected execution time of a TPWN.

For all the considered quantitative measures of TWNs or TPWNs, we want to derive results on the complexity to compute them and give algorithms that compute them. We then evaluate these algorithms on a set of workflow nets from practical applications.

5.2 New Contributions

5.2.1 Timed Sequences

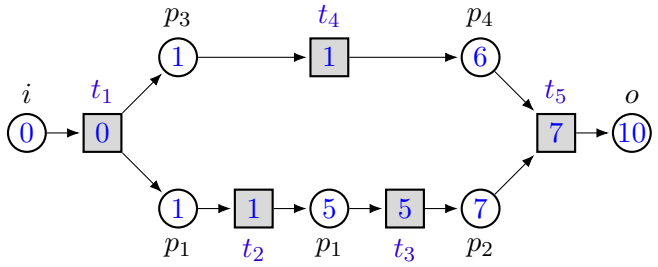
In [MEV18a] we use non-sequential processes to define causal dependencies between transitions and execution times of sequences, for which give a condensed definition from [BF88].

Definition 5.5. A *process* of a 1-safe system $((P, T, F), \mathbf{m_0})$ is a tuple (N', λ) where $N' = (P', T', F')$ is an acyclic marked graph Petri net labeled by $\lambda : P' \cup T' \to P \cup T$.

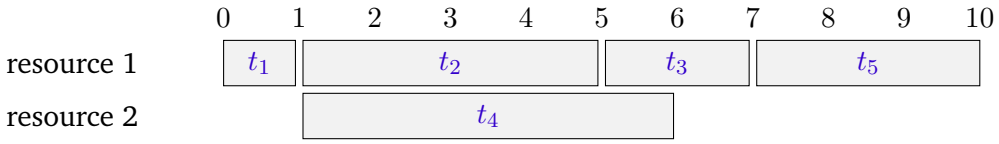
For a process $\pi = (N', \lambda) = ((P', T', F'), \lambda)$ and two elements $x, y \in P' \cup T'$, denote by $x \prec y$ that $x \neq y$ and there is a path from x to y along arcs of N'. An element x is maximal if there is no y with $x \prec y$, and we denote the set of maximal elements of π by $\max(\pi)$.

Definition 5.6. A process $\pi = (N', \lambda)$ of a 1-safe system $\mathcal{S} = (N, \mathbf{m_0})$ is *associated* to a transition sequence σ of $\mathcal{S}$ if it can be constructed inductively a follows.

- If $\sigma = \epsilon$, then the process containing a place p' with $\lambda(p') = p$ for each $p \in [\![\mathbf{m_0}]\!]$ and no other places or transitions is associated to σ.
- If $\sigma = \sigma' \cdot t$ for a transition t and a sequence σ' associated to a process π, then the extension π_t of π by t obtained by adding a new transition t' with $\lambda(t') = t$ and a new place p' with $\lambda(p') = p$ for each $p \in [\![t^\bullet]\!]$, where $[\![{}^\bullet t']\!] = [\![{}^\bullet t \circ \lambda]\!]$, $[\![t'^\bullet]\!] = [\![t^\bullet \circ \lambda]\!]$ and all $p' \in [\![{}^\bullet t']\!]$ are maximal in π, is a process associated to σ.



(a) Process with values by f_σ.



(b) Schedule given by f_σ.

Fig. 5.3 Process and schedule for a run of the timed workflow net in Figure 5.2.

Intuitively, the process associated to a sequence can be obtained by unfolding its transitions, using the existing maximal places for the presets and creating new places for the postsets. As usual, we identify processes only differing in the names of their places and transitions as isomorphic. From [BF88, Theorem 3.4.9] we have that every transition sequence σ of a 1-safe system has exactly one associated process, modulo isomorphism, which we denote by $\Pi(\sigma)$. We further have that if σ leads to a marking $\mathbf{m}$, then $[\![\mathbf{m}]\!] = \{\lambda(p') \mid p' \in \max(\Pi(\sigma))\}$, i.e. the 1-safe marking $\mathbf{m}$ is identified by the labels of maximal places of $\Pi(\sigma)$.

As an example, Figure 5.3a (ignoring the numbers for now) shows the process $\Pi(\sigma)$ for the run $\sigma = t_1 \cdot t_2 \cdot t_4 \cdot t_3 \cdot t_5$ of the timed workflow net in Figure 5.2. Places and transitions are annotated with their labels and not their names, so p_1 is present twice, as it becomes marked by t_1 and t_2. As $t_1 \prec t_2$, the occurrence of t_2 causally depends on t_1, while the occurrences of e.g. t_2 and t_4 are independent.

We use processes to define if a sequence of a timed workflow net can be executed within a given time and a number of resources.

Definition 5.7. A transition sequence σ of a timed workflow net $\mathcal{W} = (N, \tau)$ can be executed within time $t \in \mathbb{N}$ and a number of resources $k \in \mathbb{N}$ if, for the process $\Pi(\sigma) = ((P', T', F'), \lambda)$, there is a function $f : T' \to \mathbb{N}$ such that

(a) for every $t'_1, t'_2 \in T'$: if $t'_1 \prec t'_2$ then $f(t'_1) + \tau(\lambda(t'_1)) \leq f(t'_2)$;

(b) for every $t' \in T'$: $f(t') + \tau(\lambda(t')) \leq t$; and

(c) for every $0 \leq u < t$ there are at most k transitions $t' \in T'$ such that $f(t') \leq u < f(t') + \tau(\lambda(t'))$.

A function f satisfying (a) is a *schedule*, and a function satisfying (a),(b),(c) is a (k, t)-*schedule* of σ.

A schedule describes the starting time of each transition of a run, respecting the execution times and the causal order given by the process. The numbers inscribed in the transitions in Figure 5.3a give the values of some schedule f_σ of the run σ. Figure 5.3b gives an alternative view of the schedule f_σ extended with execution times, which shows that it is a $(2, 10)$-schedule.

As the process $\Pi(\sigma) = ((P', T', F'), \lambda)$ associated to a sequence σ is ayclic, using the well known Bellman's equation we can easily compute the longest path to an element with the execution times as lengths, defined by $f_\sigma : P' \cup T' \to \mathbb{N}$ as follows:

$$f_\sigma(x) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x \in P' \text{ and } {}^\bullet x = \wr\int \\ f_\sigma(t') + \tau(\lambda(t')) & \text{if } x \in P' \text{ and } {}^\bullet x = \wr t' \int \\ \max\{f_\sigma(p') \mid p' \in [\![{}^\bullet x]\!]\} & \text{if } x \in T' \end{cases}$$

We can use f_σ as a schedule by its restriction to T', but it additionally records the *arrival time* of tokens in places. Further, we define a function $\mu : T^* \to \mathbb{N}_\perp^P$ with $\mathbb{N}_\perp \stackrel{\text{def}}{=} \mathbb{N} \cup \{\perp\}$ such that, for a sequence σ of $\mathcal{W}$ leading to the marking $\mathbf{m}$, we have $\mu(\sigma)(p) \stackrel{\text{def}}{=} f_\sigma(p')$ for all $p \in [\![\mathbf{m}]\!]$ and the unique $p' \in \max(\Pi(\sigma))$ with $\lambda(p') = p$, and $\mu(\sigma)(p) \stackrel{\text{def}}{=} \perp$ for all $p \notin [\![\mathbf{m}]\!]$. The function $\mu(\sigma)$ records the arrival time of the current token in marked places, using the value $\perp$ to indicate unmarked places. Denote by $[\![\mu(\sigma)]\!] \stackrel{\text{def}}{=} \{p \in P \mid \mu(\sigma) \neq \perp\}$ the set $[\![\mathbf{m}]\!]$. Finally, we define the minimal execution time of σ by $tm(\sigma) \stackrel{\text{def}}{=} \max\{\mu(\sigma)(p) \mid p \in [\![\mu(\sigma)]\!]\}$, i.e. the last arrival time of a token.

The picture in Figure 5.3a is now complete, showing all values of f_σ inscribed into places and transitions. As the token in o arrives at time 10, we have $tm(\sigma) = 10$. The minimality of $tm(\sigma)$ follows easily from the definition of f_σ, and we have:

Lemma 5.8 ([MEV18a]). *A transition sequence σ of a timed workflow net can be executed within time $tm(\sigma)$ with $|\sigma|$ resources, and cannot be executed faster with any number of resources.*

5.2.2 Resource Threshold

We define the resource threshold of a run and of a timed workflow net itself. The resource threshold of a run gives the minimal number of resources needed to execute this run as fast as possible, and the resource threshold of a workflow net gives the minimal number of resources needed to execute *every run* as fast as possible.

Definition 5.9. The *resource threshold* of a TWN $\mathcal{W}$ and a run σ of $\mathcal{W}$, denoted by $RT(\sigma)$, is the smallest number k such that σ can be executed within time $tm(\sigma)$ with k resources. A schedule of σ *realizes* the resource threshold if it is a $(RT(\sigma), tm(\sigma))$-schedule.

The *resource threshold* of a TWN $\mathcal{W}$, denoted by $RT(\mathcal{W})$, is defined by $RT(\mathcal{W}) = \max\{RT(\sigma) \mid \sigma \text{ is a run of } \mathcal{W}\}$. A schedule of $\mathcal{W}$ is a function that assigns to every

transition sequence σ of $\mathcal{W}$ a schedule of σ. A schedule of $\mathcal{W}$ is *optimal* with k resources if it assigns to every run σ of $\mathcal{W}$ a $(k, tm(\sigma))$-schedule. A schedule of $\mathcal{W}$ *realizes* the resource threshold if it is optimal with $RT(\mathcal{W})$ resources.

For the run in Figure 5.3 of the TWN in Figure 5.2 it is easy to see that the resource threshold is 2, since t_4 and t_2, t_3 need to be executed in parallel to achieve the execution time 10. The net itself has infinitely many runs, only differing in the number of times t_2 is executed. However one can also see that every run can be executed as fast as possible with 2 resources, so the resource threshold of the net itself is also 2.

Computing the resource threshold is NP-hard

We show in [MEV18a] that computing the resource threshold $RT(\mathcal{W})$ of a TWN $\mathcal{W}$ is NP-hard, even if the workflow net is a sound marked graph, and therefore acyclic with only a single process associated to its runs. This means that computing $RT(\sigma)$ for a single run σ is also NP-hard. We obtain the result by a reduction from a classic job shop scheduling problem [Ull75], which is the problem of deciding if a number of jobs with causality constraints given by a partial order can be executed within time t on k machines. This mirrors existence of a (k, t)-schedule in Definition 5.7, and can be easily translated into a suitable timed workflow net.

Theorem 5.10 ([MEV18a]). *The following problem is NP-complete:*

> ***Given**: An sound marked graph TWN $\mathcal{W}$, and a number k.*
> ***Decide**: Does $RT(\mathcal{W}) \leq k$ hold?*

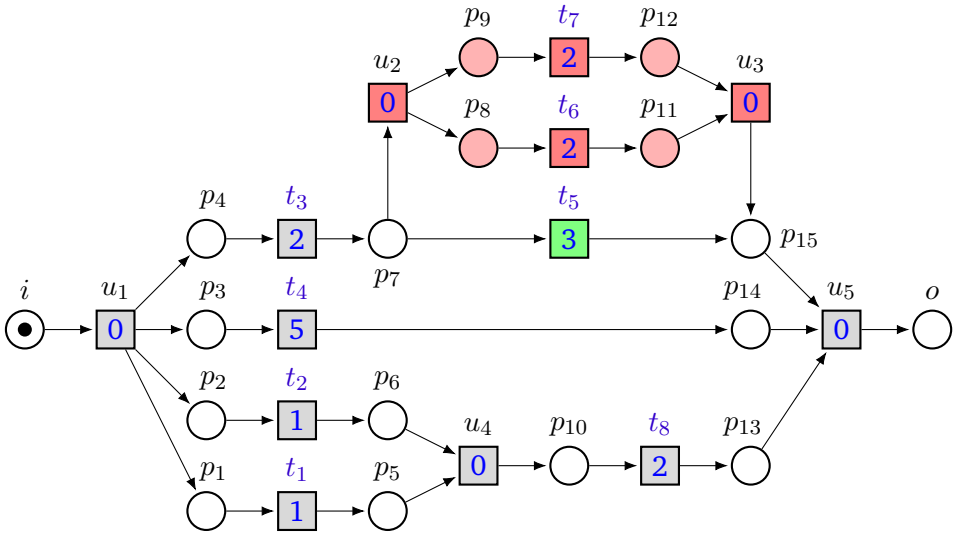
No online scheduler may realize the resource threshold

Having $RT(\mathcal{W})$ resources guarantees that every run of the TWN $\mathcal{W}$ *can* be executed with $RT(\mathcal{W})$ resources. However, a schedule of $\mathcal{W}$ might need to know the complete run to assign an optimal schedule to it, and behave arbitrarily for any of its prefixes. In many cases, one wants to directly schedule tasks as soon as they are ready, without knowing how future choices are resolved and thus not knowing how a sequence will be extended to a run. We formalize this notion by an *online schedule* of a TWN.

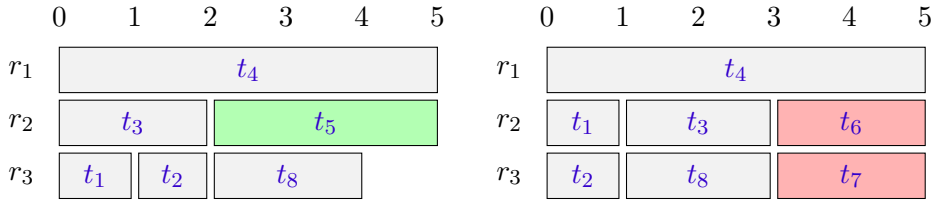
Definition 5.11. For a transition sequence $\sigma = t_1 \cdot \ldots \cdot t_n$, denote by $\sigma_\Pi = t'_1 \cdot \ldots \cdot t'_n$ the sequence of transitions of $\Pi(\sigma) = (N', \lambda)$ corresponding to its construction by Definition 5.6, i.e. especially $t'_i \prec t'_j$ implies $i < j$ and $\lambda(t'_i) = t_i$ for all $i, j \in [n]$.

A schedule f of a TWN $\mathcal{W}$ is an *online schedule* if for any two transition sequences τ, σ of $\mathcal{W}$ such that τ is a prefix of σ, i.e. $\tau(i) = \sigma(i)$ for all $i \in [|\tau|]$, we have $f(\tau)(\tau_\Pi(i)) = f(\sigma)(\sigma_\Pi(i))$ for all $i \in [|\tau|]$.

Intuitively, an online schedule may not reassign the starting times of earlier transitions when a process is extended with a new transition. We have that online schedules may not always realize the resource threshold:



(a) Timed workflow nets with runs σ_g and σ_r indicated in green and red.



(b) Schedule f_g of run σ_g. **(c)** Schedule f_r of run σ_r.

Fig. 5.4 A TWN without an online $(3,5)$-schedule and $(3,5)$-schedules for two of its runs.

Proposition 5.12 ([MEV18a]). *There is an acyclic, sound free-choice TWN for which no online schedule realizes the resource threshold.*

We give the example to show Proposition 5.12 presented in [MEV18a], adapted to execution times on transitions. Figure 5.4a shows an acyclic, sound and free-choice TWN, and Figures 5.4b and 5.4c show two schedules f_g and f_r for the runs $\sigma_g = u_1 \cdot t_1 \cdot t_2 \cdot t_3 \cdot t_4 \cdot t_5 \cdot u_4 \cdot t_8 \cdot u_5$ and $\sigma_r = u_1 \cdot t_1 \cdot t_2 \cdot t_3 \cdot t_4 \cdot u_2 \cdot t_6 \cdot t_7 \cdot u_3 \cdot u_4 \cdot t_8 \cdot u_5$. We omit the schedules for the transitions with execution time 0, since they can always be scheduled directly without requiring more resources. Since every transition occurs at most once in a run, we identify the transitions in the processes $\Pi(\sigma_r)$ and $\Pi(\sigma_g)$ by their labels. All runs of the net are associated with either the process $\Pi(\sigma_r)$ or $\Pi(\sigma_g)$, and as both can be executed within time 5 and 3 resources, and not faster with more resources, the workflow net has the resource threshold 3.

The runs have the common prefix $\sigma = u_1 \cdot t_1 \cdot t_2 \cdot t_3 \cdot t_4$. However, we have $f_g(t_3) = 0 \neq 1 = f_r(t_3)$. We show in [MEV18a] that any $(3,5)$-schedule f_g of σ_g needs to satisfy $f_g(t_3) = 0$, and any $(3,5)$-schedule f_r of σ_r needs to satisfy $f_r(t_3) \geq 1$. Therefore there can be no online $(3,5)$-schedule f of the net, which would need to satisfy both $f(\sigma)(t_3) = 0$ and $f(\sigma)(t_3) \geq 1$, since σ can be extended to either σ_g or σ_r.

5.2.3 Concurrency Threshold

Due to the negative results for the resource threshold, we consider another measure, the *concurrency threshold*, introduced in [BVT16]. The concurrency threshold is derived from the maximum number of tasks, i.e. timed transitions, that could potentially be executed in parallel. We use a definition using marked places which avoids the necessity of defining when transitions are concurrent and which gives a relation to the standard Petri net reachability problem.

Definition 5.13. Let $\mathcal{W} = (N, \tau)$ be a TWN. We define the set $D_{\mathcal{W}}$ of places in the presets of timed transitions and the *concurrency threshold* $CT(\mathcal{W})$ of $\mathcal{W}$ as follows:

$$D_{\mathcal{W}} \stackrel{\text{def}}{=} \{p \in [\![{}^\bullet t]\!] \mid t \in T : \tau(t) > 0\} \qquad CT(\mathcal{W}) \stackrel{\text{def}}{=} \max\{\mathbf{m}(D_{\mathcal{W}}) \mid \mathbf{m} \in \mathcal{R}_{\mathcal{W}}\}$$

Any TWN can be easily transformed into an equivalent TWN $\mathcal{W}$ where $|{}^\bullet t| = 1$ for each transition t with $\tau(t) > 0$. In this case, $CT(\mathcal{W})$ exactly corresponds to the number of transitions which can be executed in parallel.

If we do not know in advance how choices in a run will be resolved and also do not know the exact execution times before a task is scheduled, only whether $\tau(t) = 0$ or $\tau(t) > 0$, then any optimal online schedule needs at least $CT(\mathcal{W})$ resources. To show this, consider a sequence σ leading to a marking $\mathbf{m}$ with $\mathbf{m}(D_{\mathcal{W}}) = CT(\mathcal{W})$ concurrently enabling $CT(\mathcal{W})$ task transitions. If less than $CT(\mathcal{W})$ resources are available, then any online schedule of $\Pi(\sigma)$ would introduce a delay if we extend $\Pi(\sigma)$ with every transition concurrently enabled at $\mathbf{m}$ and give them a sufficiently large execution time. On the other hand, the very simple schedule given by f_σ, which schedules each task as soon as it becomes enabled, is an optimal online schedule using at most $CT(\mathcal{W})$ resources.

So under the assumptions that execution times are not completely known, which is common for business processes where often only a range of times or a random distribution is known, the concurrency threshold gives the minimal number of resources needed for an optimal online schedule. We also have:

Lemma 5.14 ([MEV18a])**.** *For any TWN $\mathcal{W}$, we have $RT(\mathcal{W}) \leq CT(\mathcal{W})$.*

In the TWN in Figure 5.4a, the concurrency threshold is 5, as we can reach the marking $\mathbf{m} = \wr p_1, p_2, p_3, p_8, p_9 \wr$. With the given times, actually 4 resources would suffice for an optimal online schedule. However, if we change the execution times such that $\tau(t_1) = \tau(t_2) = \tau(t_6) = \tau(t_7) = 3$, any optimal online schedule needs 5 resources, as then t_1, t_2, t_4, t_6, t_7 all need to be executed in parallel to achieve the minimal time 5.

Complexity of the concurrency threshold for classes of workflow nets

The complexity of computing the concurrency threshold mainly depends on the complexity of computing the reachability relation for the underlying Petri net. Using existing results for Petri nets [DE95; Mur89] and workflow nets [Aal97], we can characterize the reachability relation for many classes of workflow nets. Here, we use the incidence matrix (Definition 3.2), marking equation (Definition 3.3) and traps (Definition 3.6) introduced in Chapter 3. Further, to apply the theory of free-choice Petri nets [DE95] to free-choice workflow nets [Aal97], for a workflow net $\mathcal{W} = (N, i, o)$ we also define the *short-circuited* net $\overline{N}$ obtained from N by adding a new transition $\bar{t}$ with ${}^\bullet\bar{t} = \wr o \int$ and $\bar{t}^\bullet = \wr i \int$.

Proposition 5.15. *Let $\mathcal{W} = (N, i, o)$ be a workflow net with $N = (P, T, F)$ and $\mathbf{m}$ be a marking of N.*

- *If $\mathbf{m}$ is reachable from $\wr i \int$, then there is a vector $\mathbf{x} \in \mathbb{N}^T$ with $\mathbf{m} = \wr i \int + \mathbf{C} \cdot \mathbf{x}$.*
- *If $\mathcal{W}$ is a marked graph, then $\mathbf{m}$ is reachable from $\wr i \int$ if and only if there is a vector $\mathbf{x} \in \mathbb{Q}^T$ with $\mathbf{x} \geq 0$ and $\mathbf{m} = \wr i \int + \mathbf{C} \cdot \mathbf{x}$.*
- *If $\mathcal{W}$ is acyclic, then $\mathbf{m}$ is reachable from $\wr i \int$ if and only if there is a vector $\mathbf{x} \in \mathbb{N}^T$ with $\mathbf{m} = \wr i \int + \mathbf{C} \cdot \mathbf{x}$,*
- *If $\mathcal{W}$ is sound and free-choice, then $\mathbf{m}$ is reachable from $\wr i \int$ if and only if $\mathbf{m}$ marks every proper trap of $\overline{N}$ and there is a vector $\mathbf{x} \in \mathbb{Q}^T$ with $\mathbf{x} \geq 0$ and $\mathbf{m} = \wr i \int + \mathbf{C} \cdot \mathbf{x}$.*

We use these characterizations to derive a constraint approximating the concurrency threshold. For a TWN $\mathcal{W} = (((P, T, F), i, o), \tau)$ and $K \in \{\mathbb{Q}, \mathbb{Z}\}$, define the following linear optimization constraint:

$$\ell_K^{\mathcal{W}} \stackrel{\text{def}}{=} \max\{\mathbf{m}(D_{\mathcal{W}}) \mid \mathbf{m} \in K^P, \mathbf{x} \in K^T : \mathbf{m} \geq 0 \wedge \mathbf{x} \geq 0 \wedge \mathbf{m} = \wr i \int + \mathbf{C} \cdot \mathbf{x}\} \quad (5.16)$$

We identify the constraint $\ell_{\mathbb{Q}}^{\mathcal{W}}$ or $\ell_{\mathbb{Z}}^{\mathcal{W}}$ with the unique solution to the linear optimization problem defined by it. From the characterizations in Proposition 5.15 we have:

Theorem 5.17 ([MEV18a])**.** *Let $\mathcal{W}$ be a TWN. We have:*

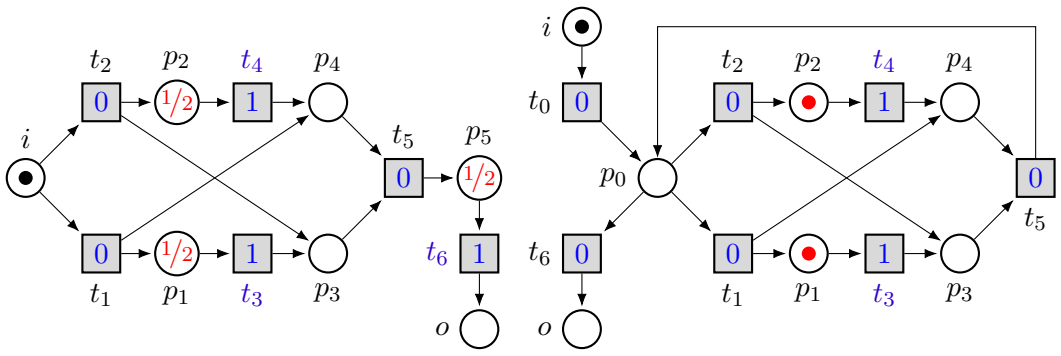
- *$\ell_{\mathbb{Q}}^{\mathcal{W}} \geq \ell_{\mathbb{Z}}^{\mathcal{W}} \geq CT(\mathcal{W})$;*
- *if $\mathcal{W}$ is a marked graph, then $\ell_{\mathbb{Z}}^{\mathcal{W}} = \ell_{\mathbb{Q}}^{\mathcal{W}} = CT(\mathcal{W})$;*
- *if $\mathcal{W}$ is acyclic, then $\ell_{\mathbb{Q}}^{\mathcal{W}} \geq \ell_{\mathbb{Z}}^{\mathcal{W}} = CT(\mathcal{W})$.*

All the inequalities above can be strict, as shown by the following theorem:

Theorem 5.18 ([MEV18a])**.**

(a) There is a sound, acyclic and free-choice TWN $\mathcal{W}$ such that $\ell_{\mathbb{Q}}^{\mathcal{W}} > CT(\mathcal{W})$.

(b) There is a sound and free-choice TWN $\mathcal{W}$ such that $\ell_{\mathbb{Z}}^{\mathcal{W}} > CT(\mathcal{W})$.



(a) Acyclic TWN $\mathcal{W}_a$ with $\ell_{\mathbb{Q}}^{\mathcal{W}_a} > CT(\mathcal{W}_a)$ **(b)** TWN $\mathcal{W}_b$ with $\ell_{\mathbb{Z}}^{\mathcal{W}_b} > CT(\mathcal{W}_b)$

Fig. 5.5 Sound and free-choice TWNs where the constraints for the concurrency threshold are inexact.

Proof. For (a), consider the sound, acyclic and free-choice TWN $\mathcal{W}_a$ in Figure 5.5a. With $\mathbf{m} = (0, 1/2, 1/2, 0, 0, 1/2, 0)$ and $\mathbf{x} = (1/2, 1/2, 0, 0, 1/2, 0)$, we have $\ell_{\mathbb{Q}}^{\mathcal{W}_a} = 3/2$, but $CT(\mathcal{W}_a) = 1$.

For (b), consider the sound and free-choice TWN $\mathcal{W}_b$ in Figure 5.5b. With $\mathbf{m} = \wr p_1, p_2 \wr$ and $\mathbf{x} = \wr t_0, t_1, t_2, t_5 \wr$. we have $\ell_{\mathbb{Z}}^{\mathcal{W}_b} = 2$, but $CT(\mathcal{W}_b) = 1$. Note that $\mathbf{m}$ does not mark the trap $\wr i, p_0, p_3, p_4, o \wr$ in $\overline{N}$. □

Theorem 5.17 shows that $CT(\mathcal{W})$ can be computed in polynomial time for marked graph workflow nets $\mathcal{W}$, as it reduces to a linear optimization problem over the rationals. However, unless P = NP, there is no polynomial-time algorithm computing the concurrency threshold of general acyclic workflow nets. This holds even if the net is sound and free-choice, where in contrast the reachability problem for a given marking is decidable in polynomial time.

Theorem 5.19 ([MEV18a]). *The following problem is NP-complete:*

> ***Given**: A sound, acyclic and free-choice TWN $\mathcal{W}$, and a number k.*
> ***Decide**: Does $CT(\mathcal{W}) \geq k$ hold?*

Proof. Membership in NP follows from the characterization of reachability for sound free-choice workflow nets in Proposition 5.15, and holds even if the net is not acyclic. We refer to [MEV18a] and its full version [MEV18b] for a detailed description of the reduction to show NP-hardness, but briefly give an overview how choices and constraints are encoded. The reduction is from the maximum independent set problem, i.e. the problem of deciding if an undirected graph $G = (V, E)$ has an independent set of vertices of size k. It employs several gadgets which are copies of the net in Figure 5.5a, only that the connections between p_3, p_4 and t_5 are changed. For each edge, copies of t_1 and t_2 give a choice between one of the vertices of the edge, and may "store" one token in p_1 or p_2 to increase the concurrency threshold by 1 for each edge, while putting another token in p_3 or p_4, which are associated to the vertices of the edge. For each vertex, a copy of t_5 is connected to its associated places p_3 or p_4 of the

gadgets for its adjacent edges, encoding the independence constraint. If the vertex is independent, then by choosing it in the gadget for each of its edges, t_5 can be used to put a token in the respective copy of p_5, further increasing the concurrency threshold by 1 for each independent vertex. The workflow net then satisfies $CT(\mathcal{W}) \geq |E| + k$ if and only if G has an independent set of vertices of size k. □

Approximating the concurrency threshold

By solving the linear optimization problem for the constraint given by (5.16) over the rationals or over the integers, we obtain two algorithms that compute an upper bound on the concurrency threshold. We identify $\ell^{\mathcal{W}}_{\mathbb{Q}}$ or $\ell^{\mathcal{W}}_{\mathbb{Z}}$ with their algorithms. By Theorem 5.17, $\ell^{\mathcal{W}}_{\mathbb{Q}}$ is exact for marked graphs and $\ell^{\mathcal{W}}_{\mathbb{Z}}$ is exact for acyclic workflow nets. For sound free-choice workflow nets that are cyclic, also $\ell^{\mathcal{W}}_{\mathbb{Z}}$ may be inexact. However, by the characterization in Proposition 5.15, we could extend the constraint-based method to an exact algorithm by an iterative refinement with traps as in Section 3.2.2 to discard markings $\mathbf{m}$ in the solution that do not mark any proper trap of $\overline{N}$. This was not considered in [MEV18a], but would be a straightforward extension.

We implemented the algorithms $\ell^{\mathcal{W}}_{\mathbb{Q}}$ or $\ell^{\mathcal{W}}_{\mathbb{Z}}$ in a tool called MACAW built on top of the mixed-integer linear programming solver CBC from the COIN-OR project [Lou03]. We compare our tool with the Petri net model checker LOLA [Wol18]. Fixing a number k, we check the lower bound $CT(\mathcal{W}) \geq k$ by querying LOLA if *some* reachable marking $\mathbf{m}$ satisfies $\mathbf{m}(D_{\mathcal{W}}) \geq k$, and check the upper bound $CT(\mathcal{W}) \leq k$ by querying LOLA if *all* reachable markings $\mathbf{m}$ satisfy $\mathbf{m}(D_{\mathcal{W}}) \leq k$.

We evaluate the tools on the 642 sound workflow nets of the IBM benchmark suite, which contains 1386 nets for industrial business processes extracted from the IBM WebSphere Business Modeler [Fah+09]. All 642 nets are free-choice, and 409 of those are marked graphs. Of the remaining 233 nets, 193 are acyclic and 40 cyclic. The workflow nets have no timing information, and we extend them to timed workflow nets by setting $\tau(t) = 1$ for all transitions t. In preliminary experiments, we computed the exact concurrency threshold of all workflow nets and set $k = CT(\mathcal{W})$ for the evaluation of LOLA.

All experiments were performed on the same machine, equipped with an Intel Core i7-6700K CPU at 4.0 GHz and 32 GB of memory. Execution time was unlimited and memory limited to 32 GB. Table 5.6 shows the results of the experimental evaluation, originally conducted in [MEV18a]. For MACAW, both $\ell^{\mathcal{W}}_{\mathbb{Q}}$ or $\ell^{\mathcal{W}}_{\mathbb{Z}}$ always terminate within milliseconds, with only a negligible difference in execution time between both. For LOLA, the computation of the lower bound for all nets only takes a few seconds, however the computation of the upper bound exceeds the memory limit for 3 nets, and takes about 5 minutes for one net, which has a few million reachable states. Comparing the values obtained by $\ell^{\mathcal{W}}_{\mathbb{Q}}$ or $\ell^{\mathcal{W}}_{\mathbb{Z}}$, surprisingly both give the exact value for the concurrency threshold of all nets, even the cyclic ones.

	Net size				Analysis time (sec)			
					MACAW		LOLA	
	$\lvert P\rvert$	$\lvert T\rvert$	$\lvert\mathcal{R}_{\mathcal{W}}\rvert$	$CT(\mathcal{W})$	$\ell_{\mathbb{Q}}^{\mathcal{W}}$	$\ell_{\mathbb{Z}}^{\mathcal{W}}$	$CT(\mathcal{W}) \geq k$	$CT(\mathcal{W}) \leq k$
Median	21	14	16	3	0.01	0.01	0.01	0.01
Mean	28.4	18.6	$3 \cdot 10^{14}$	3.7	0.01	0.01	0.01	0.58*
Max	262	284	$2 \cdot 10^{17}$	66	0.03	0.03	1.18	307.76*

Tab. 5.6 Statistics on the size of nets and analysis time of the concurrency threshold. The results are aggregated over all 642 sound nets of the IBM suite. The times marked with * exclude the 3 nets where LoLA exceeds the memory limit.

Our constraint-based algorithms are therefore often orders of magnitude faster than methods using state-space exploration. While being potentially inexact, they produce the exact value for all instances of a standard suite of free-choice workflow nets.

5.2.4 Probabilistic Sequences

We now turn to the analysis of timed probabilistic workflow nets (TPWNs) and define probabilistic semantics for them. We would like to have that the probability for extending a sequence with an enabled transition t only depends on the weights of transitions that are *in conflict* with t, i.e. those which are also enabled and share an input place with t. In this case, the occurrence of an *independent* transition u, i.e one not in conflict with t, should not change the probability for t to occur. In [MEO19a], following the work of [EHS17], we turn to *confusion-free* TPWNs, and give them semantics based on Markov decision processes (MDPs).

Definition 5.20. Let $\mathcal{W}$ be a 1-safe workflow net and $\mathbf{m}$ a 1-safe marking of $\mathcal{W}$.

For a transition t enabled at $\mathbf{m}$, the set of transitions *in conflict* with t at $\mathbf{m}$, denoted by $C(t, \mathbf{m})$, are the transitions u also enabled at $\mathbf{m}$ with $[\![{}^\bullet t]\!] \cap [\![{}^\bullet u]\!] \neq \emptyset$.

A set U of transitions is a *conflict set of* $\mathbf{m}$ if $U = C(t, \mathbf{m})$ for some transition t enabled at $\mathbf{m}$. The conflict sets of $\mathbf{m}$ are given by $\mathcal{C}(\mathbf{m}) \stackrel{\text{def}}{=} \{C(t, \mathbf{m}) \mid t \in T : {}^\bullet t \geq \mathbf{m}\}$.

$\mathcal{W}$ is *confusion-free* if for every reachable marking $\mathbf{m}$ of $\mathcal{W}$ and every pair of transitions t, u enabled at $\mathbf{m}$ with $u \notin C(t, \mathbf{m})$ we have $C(t, \mathbf{m}) = C(t, \mathbf{m} - {}^\bullet u) = C(t, \mathbf{m} + \Delta(u))$.

It follows easily that any 1-safe free-choice workflow net is confusion-free and satisfies $C(t, \mathbf{m}) = [\![[\![{}^\bullet t]\!]^\bullet]\!]$ for a transition t enabled at a marking $\mathbf{m}$. We further have:

Lemma 5.21 ([EHS17])**.** *Let $\mathcal{W}$ be a 1-safe confusion-free workflow net and $\mathbf{m}$ a reachable marking of $\mathcal{W}$. The conflict sets $\mathcal{C}(\mathbf{m})$ are a partition of the set of transitions enabled at $\mathbf{m}$.*

Markov decision process semantics

We refer to [MEO19a] and its full version [MEO19b] for the full definition for the semantics of TPWNs and give only a brief overview here. A Markov decision process (MDP) is a tuple $\mathcal{M} = (Q, q_0, Steps)$, where Q is a finite set of states, q_0 is the initial state, and $Steps : Q \to 2^{dist(Q)}$ is the probability transition function. For a TPWN $\mathcal{W}$, we define an MDP $MDP_{\mathcal{W}}$, where the states are either markings $\mathbf{m}$ or pairs $(\mathbf{m}, t)$ for a transition t enabled at a marking $\mathbf{m}$. Initially we have $q_0 = \wr i \wr$.

Intuitively, the scheduler of $MDP_{\mathcal{W}}$ (different from the schedule in Definition 5.7) non-deterministically chooses at each marking $\mathbf{m}$ a conflict set $C \in \mathcal{C}(\mathbf{m})$, after which a transition $t \in C$ is chosen at random, moving to the state $(\mathbf{m}, t)$. Afterwards, we deterministically with probability 1 move to the state $\mathbf{m} + \Delta(t)$. The probability to move from a marking $\mathbf{m}$ with a chosen conflict set $C \in \mathcal{C}(\mathbf{m})$ to a state $(\mathbf{m}, t)$ with $t \in C$ is given by is given by $P_{\mathbf{m},C}(t) \stackrel{\text{def}}{=} \frac{w(t)}{w(C)}$, where $w(C) \stackrel{\text{def}}{=} \sum_{t \in C} w(t)$. From a marking $\mathbf{m}$ which enables no transitions, we deterministically with probability 1 move back to $\mathbf{m}$.

A *path* of an MDP is an infinite sequence of states starting in the initial state. For a scheduler S of $MDP_{\mathcal{W}}$, denote by $Paths^S$ the set of paths of $MDP_{\mathcal{W}}$ compatible with S. For a path $\pi \in Paths^S$, denote by $Prob^S(\pi)$ the probability of π, using the usual probability measure for MDPs. We associate to each sequence of states π a transition sequence $\Sigma(\pi)$ obtained by projection on the components t of pairs $(\mathbf{m}, t)$ in π and removal of all states $\mathbf{m}$ in π. For a fixed scheduler S, the transition sequence $\Sigma(\pi)$ for a path $\pi \in Paths^S$ again uniquely defines π. We therefore give an alternative definition of a scheduler using transition sequences.

Definition 5.22. A *scheduler* of $MDP_{\mathcal{W}}$ for a TPWN $\mathcal{W}$ is a function $\gamma : T^* \to 2^T$ assigning to each transition sequence σ of $\mathcal{W}$ leading to a marking $\mathbf{m}$ with $\mathcal{C}(\mathbf{m}) \neq \emptyset$ a conflict set $\gamma(\sigma) \in \mathcal{C}(\mathbf{m})$. A transition sequence σ of $\mathcal{W}$ is *compatible* with a scheduler γ if for all partitions $\sigma = \sigma_1 \cdot t \cdot \sigma_2$ for some transition t, we have $t \in \gamma(\sigma_1)$.

Figure 5.7 shows the MDP for the TPWN in Figure 5.2. We show states by black nodes, abbreviating $(\mathbf{m}, t)$ by t, and show possible choices of the scheduler by white nodes. This MDP has infinitely many schedulers, each differing in how many times they choose $\{t_2, t_3\}$ at $\wr p_1, p_3 \wr$ before eventually (or never) choosing $\{t_4\}$.

5.2.5 Expected Execution Time

We are now ready to combine probabilistic sequences with timed sequences to define the expected execution time of a TPWN. From now on we assume that we have a sufficient number of resources to execute any sequence in minimal time using the online schedule f_σ.

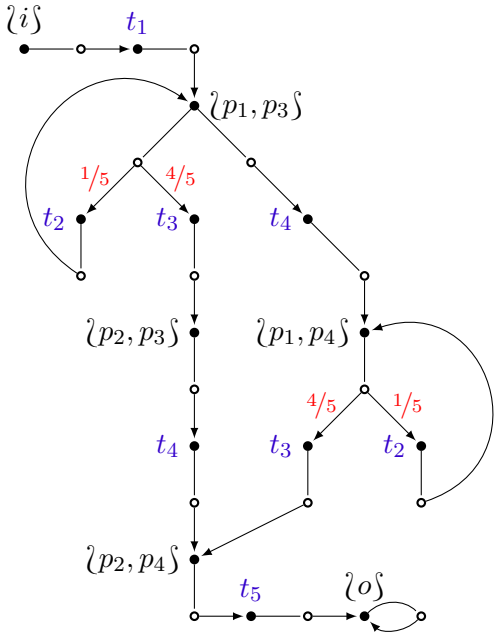


Fig. 5.7 $MDP_{\mathcal{W}}$ for the TPWN $\mathcal{W}$ in Figure 5.2.

Definition 5.23. Let $\mathcal{W}$ be a TPWN, $\mathbf{m}$ a reachable marking of $\mathcal{W}$ and π a path of $MDP_{\mathcal{W}}$. The time needed to reach $\mathbf{m}$ along π, denoted $tm(\mathbf{m}, \pi)$, is defined as follows: If π does not visit $\mathbf{m}$, then $tm(\mathbf{m}, \pi) = \infty$, otherwise, $tm(\mathbf{m}, \pi) = tm(\Sigma(\pi'))$ where π' is the shortest prefix of π ending at $\mathbf{m}$. Given a scheduler S of $\mathbf{m}$, the expected time until reaching $\mathbf{m}$ is defined as

$$ET^S_{\mathbf{m}}(\mathcal{W}) \stackrel{\text{def}}{=} \sum_{\pi \in Paths^S} tm(\mathbf{m}, \pi) \cdot Prob^S(\pi).$$

The *expected execution time* of $\mathcal{W}$ under S is defined as $ET^S(\mathcal{W}) \stackrel{\text{def}}{=} ET^S_{\wr o \wr}(\mathcal{W})$, i.e. the time to reach the final marking.

By adapting the results in [EHS17] for probabilistic workflow nets with costs to TPWNs, we show the following in [MEO19a], which allows us to speak of *the* expected execution time of confusion-free TPWNs.

Theorem 5.24 ([MEO19a])**.** *Let $\mathcal{W}$ be a confusion-free TPWN.*

(a) There exists a value $ET(\mathcal{W})$ such that $ET(\mathcal{W}) = ET^S(\mathcal{W})$ for every scheduler S of $MDP_{\mathcal{W}}$.

(b) $ET(\mathcal{W})$ is finite if and only if $\mathcal{W}$ is sound.

Intuitively, this result holds because a scheduler only determines the logical order in which transitions occur in a sequence, and in a confusion-free net the scheduler can neither change which transitions could actually occur nor change their probabilities.

Earliest-first scheduler

Theorem 5.24 shows that for confusion-free TPWNs, if we can compute $ET^S(\mathcal{W})$ for some scheduler, then we can compute $ET(\mathcal{W})$. For acyclic workflow nets, we could do this by enumeration of the finite set of runs. However, for cyclic nets, this does not work. Even though we have a finite set of reachable markings, there may be infinitely many runs with arbitrarily large execution times. In the following, we give a scheduler that can be used to derive a finite abstraction of its Markov chain, allowing computation of the expected execution time of confusion-free TPWNs.

Recall the definition of $\mu : T^* \to \mathbb{N}_\perp^P$ in Section 5.2.1, which associates to any transition sequence the last time a token arrived in a place marked by a sequence, and where further we have $[\![\mu(\sigma)]\!] = [\![\mathbf{m}]\!]$ if σ leads to $\mathbf{m}$. For a TPWN, the *earliest first* scheduler is the scheduler $\gamma : T^* \to 2^T$ defined as follows:

$$\gamma(\sigma) \stackrel{\text{def}}{=} \operatorname*{arg\,min}_{C \in \mathcal{C}([\![\mu(\sigma)]\!])} \max_{p \in [\![{}^\bullet C]\!]} \mu(\sigma)(p) \tag{5.25}$$

The scheduler γ is therefore defined by μ, and could be implemented with a memory for the current value $\mu(\sigma) \in \mathbb{N}_\perp^P$, updated on occurrence of a transition. Then $MDP_{\mathcal{W}}$ can be unfolded under the scheduler γ into a Markov chain (MC). On such a Markov chain, we can compute the probability of reaching a marking. As we have $tm(\sigma) = \max_{p \in [\![\mu(\sigma)]\!]} \mu(\sigma)(p)$, the execution time to reach a marking is also given by the state of the scheduler.

Unfortunately, for cyclic workflow nets, this may result in an infinite-state Markov chain. Figure 5.8a shows a part of this chain for the MDP of Figure 5.7. In [MEO19a], we derive a *finite abstraction* ν of μ, which results in a finite-state Markov chain sufficient to compute the expected execution time.

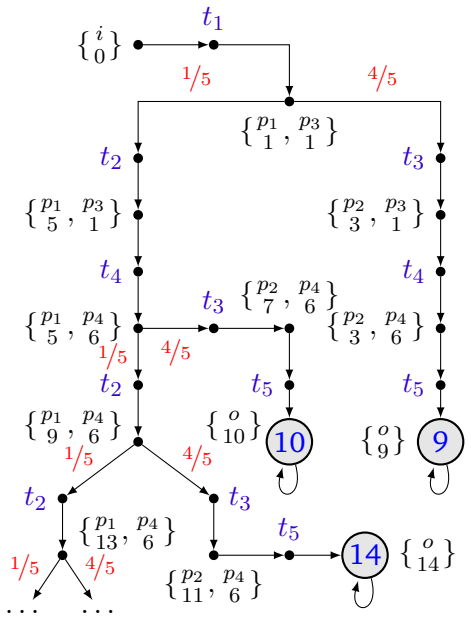
Theorem 5.26 ([MEO19a]). *Let $\mathcal{W} = (N, \tau, w)$ be a confusion-free TPWN, $H \stackrel{\text{def}}{=} \max_{t \in T} \tau(t)$ be the maximal execution time of a transition, and let $[H]_\perp \stackrel{\text{def}}{=} \{\perp, 0\} \cup [H] \subseteq \mathbb{N}_\perp$. There are functions $\nu : T^* \to [H]_\perp^P$, $f : [H]_\perp^P \times T \to [H]_\perp^P$ and $r : [H]_\perp^P \to \mathbb{N}$ such that for every transition sequence σ compatible with γ and for every $t \in T$ enabled by σ:*

$$\gamma(\sigma) = \operatorname*{arg\,min}_{C \in \mathcal{C}([\![\nu(\sigma)]\!])} \max_{p \in [\![{}^\bullet C]\!]} \nu(\sigma)(p) \tag{5.27}$$

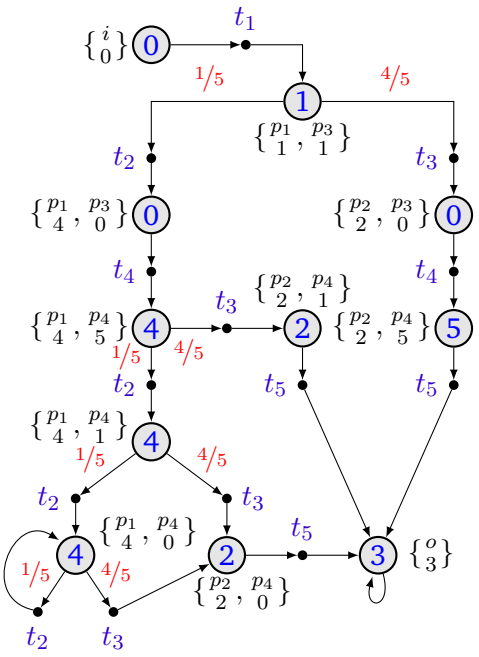
$$\nu(\sigma \cdot t) = f(\nu(\sigma), t) \tag{5.28}$$

$$tm(\sigma) = \max_{p \in [\![\nu(\sigma)]\!]} \nu(\sigma)(p) + \sum_{k=1}^{|\sigma|} r(\nu(\sigma(1) \cdot \sigma(2) \cdot \ldots \cdot \sigma(k))) \tag{5.29}$$

Unlike μ, the range of ν is finite. Equation (5.27) allows us to use ν as a scheduler, Equation (5.28) allows us to update the values of ν locally, and Equation (5.29) allows us to compute the execution time of a sequence by a series of local rewards.



(a) Infinite MC for scheduler using μ, with states labeled by $\mu(\sigma)$ and final states inscribed with $tm(\sigma)$.



(b) Finite MC for scheduler using ν, with states labeled by $\nu(\sigma)$ and inscribed with $r(\nu(\sigma))$.

Fig. 5.8 Two Markov chains for the "earliest-first" scheduler.

We give the definitions of ν, f and r, together with auxiliary functions $st : T^* \to \mathbb{N}$, $upd : \mathbb{N}^P_\perp \times T \to \mathbb{N}^P_\perp$ and an operator $\ominus : \mathbb{N}^P_\perp \times \mathbb{N} \to \mathbb{N}^P_\perp$.

$$\nu(\sigma) \stackrel{\text{def}}{=} \mu(\sigma) \ominus st(\sigma) \qquad st(\sigma) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \sigma = \epsilon \\ \max\limits_{p \in [\![^\bullet t]\!]} \mu(\sigma')(p) & \text{if } \sigma = \sigma' \cdot t \end{cases}$$

$$f(\mathbf{x}, t) \stackrel{\text{def}}{=} upd(\mathbf{x}, t) \ominus \max_{p \in {}^\bullet t} \mathbf{x}(p) \qquad upd(\mathbf{x}, t)(p) \stackrel{\text{def}}{=} \begin{cases} \max\limits_{q \in [\![^\bullet t]\!]} \mathbf{x}(q) + \tau(t) & \text{if } p \in [\![t^\bullet]\!] \\ \perp & \text{if } p \in [\![^\bullet t]\!] \setminus [\![t^\bullet]\!] \\ \mathbf{x}(p) & \text{if } p \notin [\![^\bullet t]\!] \cup [\![t^\bullet]\!] \end{cases}$$

$$r(\mathbf{x}) \stackrel{\text{def}}{=} \min_{C \in \mathcal{C}([\![\mathbf{x}]\!])} \max_{p \in [\![^\bullet C]\!]} \mathbf{x}(p) \qquad (\mathbf{x} \ominus n)(p) \stackrel{\text{def}}{=} \begin{cases} \max(\mathbf{x}(p) - n, 0) & \text{if } \mathbf{x}(p) \neq \perp \\ \perp & \text{if } \mathbf{x}(p) = \perp \end{cases}$$

Intuitively, $\nu(\sigma)$ reduces the values of $\mu(\sigma)$ by the starting time of the last transition, and only keeps the remaining arrival times of token according to a local reference frame, truncating negative values at 0. For our example TPWN in Figure 5.2, we compare in Figure 5.8 the Markov chain for μ (a) with the Markov chain obtained using ν (b). For the sequence $\sigma = t_1 \cdot t_2 \cdot t_4 \cdot t_3 \cdot t_5$ also recall the times of f_σ shown in Figure 5.3. After the prefix $\tau = t_1 \cdot t_2 \cdot t_4$, we reach the marking $\wr p_1, p_4 \int$. We have $\mu(\tau) = \{^{p_1}_{5}, ^{p_4}_{6}\}$ and $\nu(\tau) = \mu(\tau) \ominus 1 = \{^{p_1}_{4}, ^{p_4}_{5}\}$, since the starting time of t_4 is 1. Now $\gamma(\tau) = \{t_2, t_3\}$,

and if t_3 occurs, we have $\tau(t_3) = 2$, the starting time $f_\sigma(t_3) = 5$ and local starting time $\nu(\tau)(p_1) = 4$, leading to the state $\nu(\tau \cdot t_3) = upd(\nu(\tau), t_3) \ominus 4 = \{\!\{ {}^{p_2}_{6}, {}^{p_4}_{5} \}\!\} \ominus 4 = \{\!\{ {}^{p_2}_{2}, {}^{p_4}_{1} \}\!\}$. The reference frame is now at time point 5, at which the token in p_2 will arrive in 2 time units by t_3 and the token in p_6 in 1 time unit by t_4. For σ, the time $tm(\sigma)$ can be recovered by the sum of rewards $0 + 1 + 0 + 4 + 2 + 3 = 10$ along its associated path in the Markov chain in Figure 5.8b.

The proof of Theorem 5.26 crucially depends on the TPWN being confusion-free, which guarantees that if the earliest-first scheduler chooses a conflict set with a certain starting time, then all transitions in the conflict set have the same starting time, and further that no future transition of the run will have a smaller starting time. Therefore it is sound to "forget" the past before that time point, and advance the local reference frame, leaving the duration that passed as a reward in the Markov chain.

Computing the expected execution time

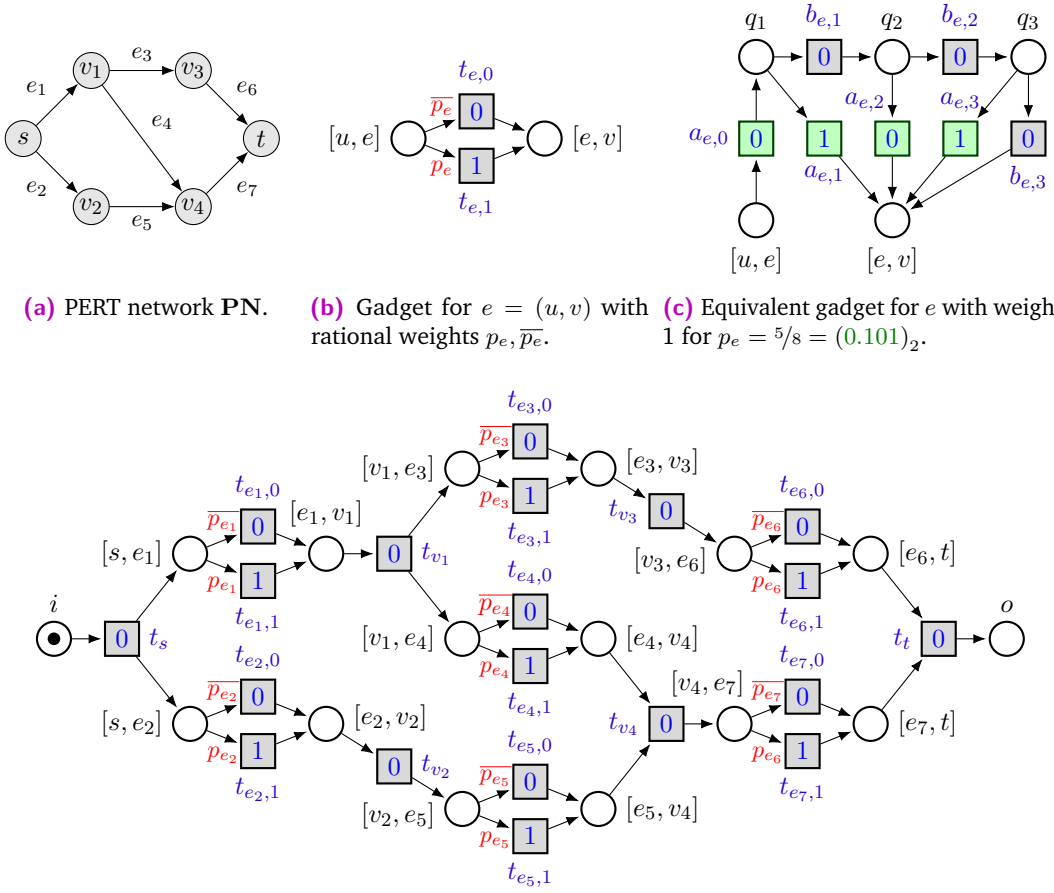
Using the earliest-first scheduler and its finite abstraction ν, we can now give an algorithm to compute the expected execution time. Given a confusion-free TPWN $\mathcal{W} = (N, \tau, w)$, we construct the Markov chain of $MDP_\mathcal{W}$ under ν, which is guaranteed to be finite. If $\mathcal{W}$ is sound, then we know that we will reach a state $\mathbf{x}$ with $\mathbf{x} = \wr o \wr$ with probability one from the initial state.

From the theory of Markov chains with rewards, we know that the expected reward until reaching such a state is the unique solution to a system of linear equations, defined as follows. Let $\mathbf{X}$ be the set of states $\mathbf{x} = \nu(\sigma) \in [H]^P_\perp$ reached in the Markov chain for some sequence σ of $\mathcal{W}$. For $\mathbf{x} \in \mathbf{X}$, let $C(\mathbf{x}) \stackrel{\text{def}}{=} \arg\min_{C \in \mathcal{C}([\![\mathbf{x}]\!])} \max_{p \in [\![\bullet C]\!]} \mathbf{x}(p)$ be the conflict set scheduled by ν at $\mathbf{x}$. We then construct a constraint over a vector $\mathbf{e} : \mathbf{X} \to \mathbb{Q}$ by a conjunction of the following linear constraints for each $\mathbf{x} \in \mathbf{X}$:

$$\begin{aligned} \mathbf{e}(\mathbf{x}) &= r(\mathbf{x}) + \sum_{t \in C(\mathbf{x})} \frac{w(t)}{w(C(\mathbf{x}))} \cdot \mathbf{e}(f(\mathbf{x}, t)) && \text{if } [\![\mathbf{x}]\!] \neq \wr o \wr \\ \mathbf{e}(\mathbf{x}) &= \max_{p \in [\![\mathbf{x}]\!]} \mathbf{x}(p) && \text{if } [\![\mathbf{x}]\!] = \wr o \wr \end{aligned} \tag{5.30}$$

The number of states $\mathbf{X} \subseteq [H]^P_\perp$ and therefore the constraint is at most exponentially larger than $\mathcal{W}$, even with times given in binary, and we can compute a solution in time polynomial in the size of the constraint. We can also test if $\mathcal{W}$ is sound in exponential time by exploration of the reachable markings. If it is unsound, we have $ET(\mathcal{W}) = \infty$. If it is sound, we compute the unique solution $\mathbf{e}$ of the constraint (5.30), where we have $ET(\mathcal{W}) = \mathbf{e}(\nu(\epsilon))$.

Theorem 5.31 ([MEO19a]). *Let $\mathcal{W}$ be a confusion-free TPWN. Then $ET(\mathcal{W})$ is either ∞ or a rational number and can be computed in single exponential time.*



(a) PERT network $\mathbf{PN}$. **(b)** Gadget for $e = (u, v)$ with rational weights $p_e, \overline{p_e}$. **(c)** Equivalent gadget for e with weights 1 for $p_e = 5/8 = (0.101)_2$.

(d) Timed probabilistic workflow net $\mathcal{W}_{\mathbf{PN}}$.

Fig. 5.9 A PERT network and its corresponding timed probabilistic workflow net.

Complexity of computing the expected execution time

In [MEO19a] we show that computing an approximation of the expected execution time of a free-choice TPWN is #P-hard, even in the restricted setting where the net is sound and acyclic, all times are 0 or 1 and all probabilities in the associated MDP are $1/2$ or 1. As e.g. computing the number of satisfying assignments of a Boolean formula (#SAT) is a #P-complete problem, a polynomial-time algorithm to compute the expected execution time would imply P = NP.

Theorem 5.32 ([MEO19a]). *The following problem is #P-hard:*

> ***Given**: A sound, acyclic and free-choice TPWN $\mathcal{W} = (N, \tau, w)$ where all transitions t satisfy $w(t) = 1$, $\tau(t) \in \{0, 1\}$ and $|[\![^\bullet t]\!]^\bullet| \leq 2$, and an $\epsilon > 0$.*
> ***Compute**: A rational r such that $r - \epsilon < ET(\mathcal{W}) < r + \epsilon$.*

We give a short overview of the reduction to show #P-hardness, illustrated on a small example in Figure 5.9. The reduction takes an instance of a *PERT network* $\mathbf{PN}$, which is a directed acyclic graph with a single source vertex s and single sink vertex t. A small PERT network is shown in Figure 5.9a. The PERT network is

	Net info & size				Analysis time (ms)			$\lvert\mathbf{X}\rvert$		
Net	cyclic	$\lvert P\rvert$	$\lvert T\rvert$	$\lvert\mathcal{R}_{\mathcal{W}}\rvert$	$[1]$	$[10^3]$	$[10^6]$	$[1]$	$[10^3]$	$[10^6]$
m1.s30_s703	no	264	286	6117	40.3	44.6	43.8	304	347	347
m1.s30_s596	yes	214	230	623	21.6	24.4	23.6	208	232	234
b3.s371_s1986	no	235	101	$2 \cdot 10^{17}$	16.8	16.4	16.5	101	102	102
b2.s275_s2417	no	103	68	$2 \cdot 10^5$	14.2	17.8	15.9	355	460	431

Tab. 5.10 Results for computing the expected execution time of the IBM suite. We give the analysis times and size of the state space $\lvert\mathbf{X}\rvert$ for the 4 nets with the highest analysis times, for each copy for the three intervals $[1], [10^3], [10^6]$ of possible weights and execution times.

further a *two-state stochastic* network where each edge e is given a probability p_e, encoded in binary, defining a Bernoulli random variable X_e with $P(X_e = 1) = p_e$ and $P(X_e = 0) = \overline{p_e} \stackrel{\text{def}}{=} 1 - p_e$. These independent random variables induce a distribution over annotations of **PN** with edge lengths 0 or 1, where for one annotation the *project duration* is the length of the longest path from s to t, and the expected value over all annotations is the *expected project duration*. From [Hag88; PB83], we have that computing an approximation to the expected project duration is a #P-hard problem.

The reduction then produces a TPWN $\mathcal{W}_{\mathbf{PN}}$ as follows. First, for each edge e, we create a gadget as shown in Figure 5.9b to simulate the random experiment by X_e for the edge length, with the probabilities p_e and $\overline{p_e}$ as rational weights. Then, these gadgets are combined into a single TPWN as shown in Figure 5.9d, which mirrors the graph structure by adding transitions connecting edges through their common vertices. A run of $\mathcal{W}_{\mathbf{PN}}$ corresponds to a random valuation of all X_e, and the execution time of the run to the length of the longest path in the PERT network. Finally, to only use weights 1, we replace the gadgets of Figure 5.9b by a sequence of coin flips producing the same distribution, illustrated in Figure 5.9c for $p_e = 5/8$.

Evaluation of computing the expected execution time

We implemented the algorithm for computing the expected execution time of a TPWN as a package named WORKFLOWNETANALYZER in the process mining toolkit PROM [Ver+10]. We evaluate it on two sets of benchmarks. The first are the 642 sound and free-choice workflow nets of the IBM benchmark suite [Fah+09]. As these do not have timing or probabilistic information, we create three copies of each net in the suite, and annotate each copy with times and weights chosen uniformly at random from different intervals: (i) $w(t) = \tau(t) = 1$, (ii) $w(t), \tau(t) \in [10^3]$ and (iii) $w(t), \tau(t) \in [10^6]$. The second suite is a workflow net obtained from the academic winner of the BPI Challenge 2017 [Rod+17] for analyzing a financial process, which we will identify as the BPI net. For this net we use real-word data from the challenge [Don17] to

				Analysis time		
Distribution	$\|T\|$	$ET(\mathcal{W})$	$\|\mathbf{X}\|$	Total	Construction	Solving
Deterministic Mean	19	24 d 1 h	33	40 ms	18 ms	22 ms
Histogram/12 h	141	24 d 18 h	4054	244 ms	232 ms	12 ms
Histogram/6 h	261	24 d 21 h	15522	2.1 s	1.8 s	0.3 s
Histogram/4 h	375	24 d 22 h	34063	10 s	6 s	4 s
Histogram/2 h	666	24 d 23 h	122785	346 s	52 s	294 s
Histogram/1 h	1117	—	422614	—	12.7 min	MO

Tab. 5.11 Results for computing the expected execution time of the BPI net. We give the expected execution time, analysis time with time for constructing and solving the constraint, and size of the state space $|\mathbf{X}|$ for various distributions of transition times. MO denotes denotes exceeding the memory limit.

annotate it with quantitative information. For each transition, this gives us a probability which we use as the weight and a discrete distribution of execution times. As the range of the distributions is too large to analyze completely, in the evaluation we either simply use its mean or a coarser histogram distribution with differently sized buckets. Our implementation and benchmarks have been published as a peer-reviewed artifact [MEO19c] which can be used to reproduce the experimental results.

All experiments were performed on the same machine, equipped with an Intel Core i7-6700K CPU at 4.0 GHz and 32 GB of memory. Execution time was unlimited and memory limited to 32 GB. The results of the experimental evaluation, originally conducted in [MEO19a], are shown in Table 5.10 for 4 of the IBM workflow nets and in Table 5.11 for the BPI net. The expected execution time of all the 642 IBM workflow nets can be computed in milliseconds, and does not seem strongly correlated with the number of reachable markings or the intervals of the execution times. For the BPI net, we can analyze it within minutes for an already rather precise approximation of the execution times, however after exceeding 1000 transitions the size of the constraint becomes too large to solve exactly.

The performance of our algorithm mainly depends on the number and range of choices along runs of the workflow net, i.e. the sizes of the conflict sets, and the corresponding range of possible execution times. However, due to our results of only needing to consider the earliest-first scheduler, the algorithm does not need to explore all possible reachable markings and can also handle workflow nets with a large state space.

5.3 Related Work

Our starting point for the analysis of the resource and concurrency threshold was the work by Botezatu et al. [BVT15; BVT16]. They introduce the notion of the concurrency threshold and give an algorithm to compute it by a decomposition algorithm, which also uses the linear optimization constraint for marked graphs, but resorts to computation of the reachability graph for complex fragments. In contrast, we give a simpler NP algorithm for all sound free-choice workflow nets, and experimentally show that the polynomial approximation algorithm is already exact in practice. They also show NP-hardness for deciding existence of an optimal schedule with a fixed number of resources, which is however slightly different to our problem of computing the resource threshold.

Botezatu et al. [BVT16] also consider the computation of the expected execution time. They show that the threshold decision problem is NP-hard. However, their reduction requires execution times to be given in binary, while we prove #P-hardness where all times are 0 or 1. They also do not give any decidability result.

Our starting point for the analysis of the expected execution time was the work by Esparza et al. [EHS16; EHS17]. They introduce probabilistic workflow nets (PWNs) and analyze the problem of computing the *expected cost* for PWNs where transitions t are annotated with a cost $c(t)$. Similarly to expected time, the expected cost for confusion-free PWNs is independent of the scheduler. In contrast to the expected time, the expected cost can be computed in polynomial time, as shown in [EHS17] by a set of reduction rules. The algorithm does not transfer to the expected time, as time and cost behave differently in the presence of concurrency: the cost of two concurrent transitions t_1, t_2 is $c(t_1)+c(t_2)$, while the execution time is $\max(\tau(t_1), \tau(t_2))$. In the probabilistic case, for two independent random variables X_1, X_2, we have $\mathbb{E}(X_1 + X_2) = \mathbb{E}(X_1) + \mathbb{E}(X_2)$, however $\mathbb{E}(\max(X_1, X_2)) \geq \max(\mathbb{E}(X_1), \mathbb{E}(X_2))$ with inequality in general.

Probabilistic workflow nets where time is continuous and execution times are exponentially distributed can be translated to Generalized Stochastic Petri Nets (GSPNs) [Fer95; MCB84]. These are popular due to their nice computational properties, as the memoryless property of the exponential distribution allows a reduction to a continuous-time Markov chain for further analysis. This also allows for exact computation of the expected execution time. However, the construction of the Markov chain might require exploration of all reachable markings of the net. Further, exponential distributions are not always desired or insufficient for a precise model of the execution times. While other distributions can be approximated by phase-type distributions [Mar+85], this does not lead to an exact algorithm and increases the analysis time exponentially.

Aalst et al. [AHR00] consider discrete time stochastic workflow nets where transitions have a probability distribution over discrete execution times. They present an approach to compute the probability distribution for the total execution time by a hierarchical decomposition into sequence, iteration, choice and parallel blocks. However the computation for iteration only gives an approximation, and the decomposition does not capture all free-choice nets. They sketch an algorithm for acyclic free-choice nets based on computation of the probability distributions of arrival times of tokens, which resembles our approach. They mention this could be combined with their decomposition, but it is unclear if this results in a complete algorithm for free-choice nets. Similar work [EP02; Ha+06] also computes distributions using a block decomposition, but these are also incomplete for all free-choice nets and approximative for cyclic nets.

5.4 Open Problems

Proposition 5.12 states that there may be no online schedule realizing the resource threshold. There we require that tasks are executed without interruption after starting. One could also consider *preemptive* online schedules, where tasks may be interrupted and resumed later. This corresponds to timed workflow nets where all execution times of transitions are 0 or 1, unlike our example in Figure 5.4. The scheduling problem for an optimal finish time with a preemptive schedule remains NP-hard [GS78], which should also transfer to computing the resource threshold or deriving an optimal schedule. Still, one could explore whether online schedules are always possible in this case or a counterexample exists.

We show that computing the expected execution time is #P-hard and in EXPTIME (for the threshold decision problem). One could further research the exact complexity of this problem. We have NP-hardness with arbitrary times [BVT16], but no known reduction for NP-hardness with times 0 or 1. In the acyclic case, a PSPACE algorithm should be possible by enumeration of all runs. In the cyclic case, it is unclear if the expected execution time (in binary) even has a size polynomial in the size of the workflow net. For the related problem of computing the expected duration of a PERT network, it is not known whether it is in #P [Hag88]. For non-free-choice workflow nets, the problem of deciding if the expected cost of a confusion-free PWN with only a single run is 0 is PSPACE-hard [EHS17], which directly transfers to the expected execution time. However [EHS17] gives no PSPACE upper bound.

For computing the expected execution time, we restricted ourselves to confusion-free workflow-nets. Workflow nets are not always confusion-free, and confusion allows expressing more complex control flow structures. One could also explore the notion of expected time for general workflow nets. Our definition using MDPs already gives an expected execution time for each scheduler, and one could analyze the maximum or minimum over all schedulers. However, a problem is that the probabilistic semantics

for nets with confusion are not always intuitive or matching the original system. Esparza et al. [EHS17] briefly consider this for the expected cost, where examples of PWNs with confusion are shown where the MDP semantics is still appropriate. Bruni et al. [BMM19] give a reduction of a Petri net with confusion to a confusion-free Petri net model extended with certain *persistent* places, on which a suitable semantics for causality and probabilities is derived using non-sequential processes with persistence. One could investigate if this model could also be suitably extended with time for the analysis of the expected execution time of workflow nets with confusion.

Bibliography

[AAE06] D. Angluin, J. Aspnes, and D. Eisenstat. “Stably computable predicates are semi-linear”. In: *Proceedings of the 25th ACM Symposium on Principles of Distributed Computing, PODC 2006*. Ed. by E. Ruppert and D. Malkhi. ACM, 2006, pp. 292–299. DOI: 10.1145/1146381.1146425.

[AAE08] D. Angluin, J. Aspnes, and D. Eisenstat. “Fast computation by population protocols with a leader”. In: *Distributed Computing* 21.3 (2008), pp. 183–199. DOI: 10.1007/s00446-008-0067-z.

[AAG18] D. Alistarh, J. Aspnes, and R. Gelashvili. “Space-Optimal Majority in Population Protocols”. In: *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*. Ed. by A. Czumaj. SIAM, 2018, pp. 2221–2239. DOI: 10.1137/1.9781611975031.144.

[Aal03] W. M. P. van der Aalst. “Challenges in Business Process Management: Verification of Business Processing Using Petri Nets”. In: *Bulletin of the EATCS* 80 (2003), pp. 174–199.

[Aal+11] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, et al. “Soundness of workflow nets: classification, decidability, and analysis”. In: *Formal Aspects of Computing* 23.3 (2011), pp. 333–363. DOI: 10.1007/s00165-010-0161-4.

[Aal97] W. M. P. van der Aalst. “Verification of Workflow Nets”. In: *Proceedings of the 18th International Conference on Application and Theory of Petri Nets, ICATPN 1997*. Ed. by P. Azéma and G. Balbo. Vol. 1248. Lecture Notes in Computer Science. Springer, 1997, pp. 407–426. DOI: 10.1007/3-540-63139-9_48.

[Aal98] W. M. P. van der Aalst. “The Application of Petri Nets to Workflow Management”. In: *Journal of Circuits, Systems and Computers* 8.1 (1998), pp. 21–66. DOI: 10.1142/S0218126698000043.

[ABQ11] M. F. Atig, A. Bouajjani, and S. Qadeer. “Context-Bounded Analysis For Concurrent Programs With Dynamic Creation of Threads”. In: *Logical Methods in Computer Science* 7.4 (2011). DOI: 10.2168/LMCS-7(4:4)2011.

[AFK88] K. R. Apt, N. Francez, and S. Katz. “Appraising Fairness in Languages for Distributed Programming”. In: *Distributed Computing* 2.4 (1988), pp. 226–241. DOI: 10.1007/BF01872848.

[AGV15] D. Alistarh, R. Gelashvili, and M. Vojnovic. “Fast and Exact Majority in Population Protocols”. In: *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing, PODC 2015*. Ed. by C. Georgiou and P. G. Spirakis. ACM, 2015, pp. 47–56. DOI: 10.1145/2767386.2767429.

[AH02] W. M. P. van der Aalst and K. M. van Hee. *Workflow Management: Models, Methods, and Systems*. Cooperative Information Systems. MIT Press, 2002.

[AH11] M. F. Atig and P. Habermehl. "On Yen's Path Logic for Petri Nets". In: *International Journal of Foundations of Computer Science* 22.4 (2011), pp. 783–799. DOI: 10.1142/S0129054111008428.

[AHR00] W. M. P. van der Aalst, K. van Hee, and H. Reijers. "Analysis of discrete-time stochastic Petri nets". In: *Statistica Neerlandica* 54.2 (2000), pp. 237–255. DOI: 10.1111/1467-9574.00139.

[Aks+17] S. Akshay, S. Chakraborty, A. Das, V. Jagannath, and S. Sandeep. "On Petri Nets with Hierarchical Special Arcs". In: *Proceedings of the 28th International Conference on Concurrency Theory, CONCUR 2017*. Ed. by R. Meyer and U. Nestmann. Vol. 85. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 40:1–40:17. DOI: 10.4230/LIPIcs.CONCUR.2017.40.

[Ali+17] D. Alistarh, J. Aspnes, D. Eisenstat, R. Gelashvili, and R. L. Rivest. "Time-Space Trade-offs in Population Protocols". In: *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*. Ed. by P. N. Klein. SIAM, 2017, pp. 2560–2579. DOI: 10.1137/1.9781611974782.169.

[ALW16] K. Athanasiou, P. Liu, and T. Wahl. "Unbounded-Thread Program Verification using Thread-State Equations". In: *Proceedings of the 8th International Joint Conference on Automated Reasoning, IJCAR 2016*. Ed. by N. Olivetti and A. Tiwari. Vol. 9706. Lecture Notes in Computer Science. Springer, 2016, pp. 516–531. DOI: 10.1007/978-3-319-40229-1_35.

[Ang+04] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. "Computation in networks of passively mobile finite-state sensors". In: *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing, PODC 2004*. Ed. by S. Chaudhuri and S. Kutten. ACM, 2004, pp. 290–299. DOI: 10.1145/1011767.1011810.

[Ang+06] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. "Computation in networks of passively mobile finite-state sensors". In: *Distributed Computing* 18.4 (2006), pp. 235–253. DOI: 10.1007/s00446-005-0138-3.

[Ang+07] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. "The computational power of population protocols". In: *Distributed Computing* 20.4 (2007), pp. 279–304. DOI: 10.1007/s00446-007-0040-2.

[AR09] J. Aspnes and E. Ruppert. "An Introduction to Population Protocols". In: *Middleware for Network Eccentric and Mobile Applications*. Ed. by B. Garbinato, H. Miranda, and L. E. T. Rodrigues. Springer, 2009, pp. 97–120. DOI: 10.1007/978-3-540-89707-1_5.

[BEJ18a] M. Blondin, J. Esparza, and S. Jaax. "Large Flocks of Small Birds: on the Minimal Size of Population Protocols". In: *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*. Ed. by R. Niedermeier and B. Vallée. Vol. 96. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 16:1–16:14. DOI: 10.4230/LIPIcs.STACS.2018.16.

[BEJ18b] M. Blondin, J. Esparza, and S. Jaax. “Peregrine: A Tool for the Analysis of Population Protocols”. In: *Proceedings of the 30th International Conference on Computer Aided Verification, CAV 2018*. Ed. by H. Chockler and G. Weissenbacher. Vol. 10981. Lecture Notes in Computer Science. Springer, 2018, pp. 604–611. DOI: 10.1007/978-3-319-96145-3_34.

[BEK18] M. Blondin, J. Esparza, and A. Kučera. “Automatic Analysis of Expected Termination Time for Population Protocols”. In: *Proceedings of the 29th International Conference on Concurrency Theory, CONCUR 2018*. Ed. by S. Schewe and L. Zhang. Vol. 118. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 33:1–33:16. DOI: 10.4230/LIPIcs.CONCUR.2018.33.

[BF88] E. Best and C. Fernández. *Nonsequential Processes - A Petri Net View*. Vol. 13. EATCS Monographs on Theoretical Computer Science. Springer, 1988. DOI: 10.1007/978-3-642-73483-0.

[BF97] Z. Bouziane and A. Finkel. “Cyclic Petri Net Reachability Sets are Semi-linear Effectively Constructible”. In: *Proceedings of the Second International Workshop on Verification of Infinite State Systems, Infinity 1997*. Ed. by F. Moller. Vol. 9. Electronic Notes in Theoretical Computer Science. Elsevier, 1997, pp. 15–24. DOI: 10.1016/S1571-0661(05)80423-2.

[BH17] M. Blondin and C. Haase. “Logics for continuous reachability in Petri nets and vector addition systems with states”. In: *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*. IEEE Computer Society, 2017, pp. 1–12. DOI: 10.1109/LICS.2017.8005068.

[BKP15] H. Bride, O. Kouchnarenko, and F. Peureux. “Constraint Solving for Verifying Modal Specifications of Workflow Nets with Data”. In: *Revised Selected Papers of the 10th International Andrei Ershov Informatics Conference on Perspectives of System Informatics, PSI 2015*. Ed. by M. Mazzara and A. Voronkov. Vol. 9609. Lecture Notes in Computer Science. Springer, 2015, pp. 75–90. DOI: 10.1007/978-3-319-41579-6_7.

[Blo+16] M. Blondin, A. Finkel, C. Haase, and S. Haddad. “Approaching the Coverability Problem Continuously”. In: *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2016*. Ed. by M. Chechik and J. Raskin. Vol. 9636. Lecture Notes in Computer Science. Springer, 2016, pp. 480–496. DOI: 10.1007/978-3-662-49674-9_28.

[Blo+17] M. Blondin, J. Esparza, S. Jaax, and P. J. Meyer. “Towards Efficient Verification of Population Protocols”. In: *Proceedings of the 36th ACM Symposium on Principles of Distributed Computing, PODC 2017*. Ed. by E. M. Schiller and A. A. Schwarzmann. ACM, 2017, pp. 423–430. DOI: 10.1145/3087801.3087816.

[Blo+20a] M. Blondin, J. Esparza, B. Genest, M. Helfrich, and S. Jaax. “Succinct Population Protocols for Presburger Arithmetic”. In: *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020*. Ed. by C. Paul and M. Bläser. Vol. 154. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 40:1–40:15. DOI: 10.4230/LIPIcs.STACS.2020.40.

[Blo+20b] M. Blondin, J. Esparza, M. Helfrich, A. Kučera, and P. J. Meyer. “Checking Qualitative Liveness Properties of Replicated Systems with Stochastic Scheduling”. In: *Proceedings of the 32nd International Conference on Computer Aided Verification, CAV 2020*. Ed. by S. K. Lahiri and C. Wang. Vol. 12225. Lecture Notes in Computer Science. Springer, 2020, pp. 372–397. DOI: 10.1007/978-3-030-53291-8_20.

[BLR13] H. Boucheneb, D. Lime, and O. H. Roux. “On Multi-enabledness in Time Petri Nets”. In: *Proceedings of the 34th International Conference on Application and Theory of Petri Nets and Concurrency, PETRI NETS 2013*. Ed. by J. M. Colom and J. Desel. Vol. 7927. Lecture Notes in Computer Science. Springer, 2013, pp. 130–149. DOI: 10.1007/978-3-642-38697-8_8.

[BMM19] R. Bruni, H. C. Melgratti, and U. Montanari. “Concurrency and Probability: Removing Confusion, Compositionally”. In: *Logical Methods in Computer Science* 15.4 (2019). DOI: 10.23638/LMCS-15(4:17)2019.

[Bou87] L. Bougé. “Repeated Snapshots in Distributed Systems with Synchronous Communications and their Implementation in CSP”. In: *Theoretical Computer Science* 49 (1987), pp. 145–169. DOI: 10.1016/0304-3975(87)90005-3.

[BVT15] M. Botezatu, H. Völzer, and L. Thiele. “The Complexity of Deadline Analysis for Workflow Graphs with a Single Resource”. In: *Proceedings of the 20th International Conference on Engineering of Complex Computer Systems, ICECCS 2015*. IEEE, 2015, pp. 110–119. DOI: 10.1109/ICECCS.2015.22.

[BVT16] M. Botezatu, H. Völzer, and L. Thiele. “The Complexity of Deadline Analysis for Workflow Graphs with Multiple Resources”. In: *Proceedings of the 14th International Conference on Business Process Management, BPM 2016*. Ed. by M. L. Rosa, P. Loos, and O. Pastor. Vol. 9850. Lecture Notes in Computer Science. Springer, 2016, pp. 252–268. DOI: 10.1007/978-3-319-45348-4_15.

[CA95] J. C. Corbett and G. S. Avrunin. “Using Integer Programming to Verify General Safety and Liveness Properties”. In: *Formal Methods in System Design* 6.1 (1995), pp. 97–123. DOI: 10.1007/BF01384316.

[CEP95] A. Cheng, J. Esparza, and J. Palsberg. “Complexity Results for 1-Safe Nets”. In: *Theoretical Computer Science* 147.1&2 (1995), pp. 117–136. DOI: 10.1016/0304-3975(94)00231-7.

[CH16] D. Chistikov and C. Haase. “The Taming of the Semi-Linear Set”. In: *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*. Ed. by I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi. Vol. 55. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 128:1–128:13. DOI: 10.4230/LIPIcs.ICALP.2016.128.

[CH17] D. Chistikov and C. Haase. “On the Complexity of Quantified Integer Programming”. In: *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*. Ed. by I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl. Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 94:1–94:13. DOI: 10.4230/LIPIcs.ICALP.2017.94.

[Che+17] H. Chen, R. Cummings, D. Doty, and D. Soloveichik. “Speed faults in computation by chemical reaction networks”. In: *Distributed Computing* 30.5 (2017), pp. 373–390. DOI: 10.1007/s00446-015-0255-6.

[Clé+11] J. Clément, C. Delporte-Gallet, H. Fauconnier, and M. Sighireanu. “Guidelines for the Verification of Population Protocols”. In: *Proceedings of the 31st International Conference on Distributed Computing Systems, ICDCS 2011*. IEEE, 2011, pp. 215–224. DOI: 10.1109/ICDCS.2011.36.

[CMS10] I. Chatzigiannakis, O. Michail, and P. G. Spirakis. “Algorithmic Verification of Population Protocols”. In: *Proceedings of the 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS 2010*. Ed. by S. Dolev, J. A. Cobb, M. J. Fischer, and M. Yung. Vol. 6366. Lecture Notes in Computer Science. Springer, 2010, pp. 221–235. DOI: 10.1007/978-3-642-16023-3_19.

[Coo71] S. A. Cook. “The Complexity of Theorem-Proving Procedures”. In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, STOC 1971*. Ed. by M. A. Harrison, R. B. Banerji, and J. D. Ullman. ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047.

[Coo72] D. C. Cooper. “Theorem Proving in Arithmetic without Multiplication”. In: *Machine Intelligence* 7 (1972), pp. 91–99.

[Cze+19] W. Czerwinski, S. Lasota, R. Lazic, J. Leroux, and F. Mazowiecki. “The reachability problem for Petri nets is not elementary”. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*. Ed. by M. Charikar and E. Cohen. ACM, 2019, pp. 24–33. DOI: 10.1145/3313276.3316369.

[DE00] J. Desel and T. Erwin. “Modeling, Simulation and Analysis of Business Processes”. In: *Business Process Management, Models, Techniques, and Empirical Studies*. Ed. by W. M. P. van der Aalst, J. Desel, and A. Oberweis. Vol. 1806. Lecture Notes in Computer Science. Springer, 2000, pp. 129–141. DOI: 10.1007/3-540-45594-9_9.

[DE16] J. Desel and J. Esparza. “Negotiations and Petri Nets”. In: *Transactions on Petri Nets and Other Models of Concurrency* 11 (2016), pp. 203–225. DOI: 10.1007/978-3-662-53401-4_10.

[DE95] J. Desel and J. Esparza. *Free Choice Petri Nets*. USA: Cambridge University Press, 1995.

[DGS99] S. Dolev, M. G. Gouda, and M. Schneider. “Memory Requirements for Silent Stabilization”. In: *Acta Informatica* 36.6 (1999), pp. 447–462. DOI: 10.1007/s002360050180.

[Dic13] L. E. Dickson. “Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors”. In: *American Journal of Mathematics* 35.4 (1913), pp. 413–422. DOI: 10.2307/2370405.

[DKO13] E. D’Osualdo, J. Kochems, and C. L. Ong. “Automatic Verification of Erlang-Style Concurrency”. In: *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*. Ed. by F. Logozzo and M. Fähndrich. Vol. 7935. Lecture Notes in Computer Science. Springer, 2013, pp. 454–476. DOI: 10.1007/978-3-642-38856-9_24.

[DKR82] D. Dolev, M. M. Klawe, and M. Rodeh. “An $O(n \log n)$ Unidirectional Distributed Algorithm for Extrema Finding in a Circle”. In: *Journal of Algorithms* 3.3 (1982), pp. 245–260. DOI: 10.1016/0196-6774(82)90023-2.

[DL15] F. Drewes and J. Leroux. “Structurally Cyclic Petri Nets”. In: *Logical Methods in Computer Science* 11.4 (2015). DOI: 10.2168/LMCS-11(4:15)2015.

[DM09] Y. Deng and J. Monin. “Verifying Self-stabilizing Population Protocols with Coq”. In: *Proceedings of the 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering, TASE 2009*. Ed. by W. Chin and S. Qin. IEEE Computer Society, 2009, pp. 201–208. DOI: 10.1109/TASE.2009.9.

[Don+07] B. F. van Dongen, M. H. Jansen-Vullers, H. M. W. Verbeek, and W. M. P. van der Aalst. “Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants”. In: *Computers in Industry* 58.6 (2007), pp. 578–601. DOI: 10.1016/j.compind.2007.01.001.

[Don17] B. F. van Dongen. *BPI Challenge 2017*. 2017. DOI: 10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b.

[DRB02] G. Delzanno, J. Raskin, and L. V. Begin. “Towards the Automated Verification of Multithreaded Java Programs”. In: *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2002*. Ed. by J. Katoen and P. Stevens. Vol. 2280. Lecture Notes in Computer Science. Springer, 2002, pp. 173–187. DOI: 10.1007/3-540-46002-0_13.

[EHS16] J. Esparza, P. Hoffmann, and R. Saha. “Polynomial Analysis Algorithms for Free Choice Probabilistic Workflow Nets”. In: *Proceedings of the 13th International Conference on Quantitative Evaluation of Systems, QEST 2016*. Ed. by G. Agha and B. V. Houdt. Vol. 9826. Lecture Notes in Computer Science. Springer, 2016, pp. 89–104. DOI: 10.1007/978-3-319-43425-4_6.

[EHS17] J. Esparza, P. Hoffmann, and R. Saha. “Polynomial analysis algorithms for free choice Probabilistic Workflow Nets”. In: *Performance Evaluation* 117 (2017), pp. 104–129. DOI: 10.1016/j.peva.2017.09.006.

[EM00] J. Esparza and S. Melzer. “Verification of Safety Properties Using Integer Programming: Beyond the State Equation”. In: *Formal Methods in System Design* 16.2 (2000), pp. 159–189. DOI: 10.1023/A:1008743212620.

[EM15] J. Esparza and P. J. Meyer. “An SMT-based Approach to Fair Termination Analysis”. In: *Proceedings of the 15th Conference on Formal Methods in Computer-Aided Design, FMCAD 2015*. Ed. by R. Kaivola and T. Wahl. IEEE, 2015, pp. 49–56. DOI: 10.1109/FMCAD.2015.7542252.

[EM97] J. Esparza and S. Melzer. “Model Checking LTL Using Constraint Programming”. In: *Proceedings of the 18th International Conference on Application and Theory of Petri Nets, ICATPN 1997*. Ed. by P. Azéma and G. Balbo. Vol. 1248. Lecture Notes in Computer Science. Springer, 1997, pp. 1–20. DOI: 10.1007/3-540-63139-9_26.

[EP02] J. Eder and H. Pichler. “Duration Histograms for Workflow Systems”. In: *Proceedings of the IFIP TC8 / WG8.1 Working Conference on Engineering Information Systems in the Internet Context, EISIC 2002*. Ed. by C. Rolland, S. Brinkkemper, and M. Saeki. Vol. 231. IFIP Conference Proceedings. Kluwer, 2002, pp. 239–253. DOI: 10.1007/978-0-387-35614-3_14.

[ERW19] J. Esparza, M. A. Raskin, and C. Weil-Kennedy. “Parameterized Analysis of Immediate Observation Petri Nets”. In: *Proceedings of the 40th International Conference on Application and Theory of Petri Nets and Concurrency, PETRI NETS 2019*. Ed. by S. Donatelli and S. Haar. Vol. 11522. Lecture Notes in Computer Science. Springer, 2019, pp. 365–385. DOI: 10.1007/978-3-030-21571-2_20.

[Esp+14] J. Esparza, R. Ledesma-Garza, R. Majumdar, P. Meyer, and F. Niksic. “An SMT-Based Approach to Coverability Analysis”. In: *Proceedings of the 26th International Conference on Computer Aided Verification, CAV 2014*. Ed. by A. Biere and R. Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 603–619. DOI: 10.1007/978-3-319-08867-9_40.

[Esp+15] J. Esparza, P. Ganty, J. Leroux, and R. Majumdar. “Verification of Population Protocols”. In: *Proceedings of the 26th International Conference on Concurrency Theory, CONCUR 2015*. Ed. by L. Aceto and D. de Frutos-Escrig. Vol. 42. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 470–482. DOI: 10.4230/LIPIcs.CONCUR.2015.470.

[Esp+17] J. Esparza, P. Ganty, J. Leroux, and R. Majumdar. “Verification of population protocols”. In: *Acta Informatica* 54.2 (2017), pp. 191–215. DOI: 10.1007/s00236-016-0272-3.

[Esp+18] J. Esparza, P. Ganty, R. Majumdar, and C. Weil-Kennedy. “Verification of Immediate Observation Population Protocols”. In: *Proceedings of the 29th International Conference on Concurrency Theory, CONCUR 2018*. Ed. by S. Schewe and L. Zhang. Vol. 118. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 31:1–31:16. DOI: 10.4230/LIPIcs.CONCUR.2018.31.

[Esp+20] J. Esparza, M. Helfrich, S. Jaax, and P. J. Meyer. “Peregrine 2.0: Explaining Correctness of Population Protocols Through Stage Graphs”. In: *Proceedings of the 18th International Symposium on Automated Technology for Verification and Analysis, ATVA 2020*. Ed. by D. V. Hung and O. Sokolsky. Vol. 12302. Lecture Notes in Computer Science. Springer, 2020, pp. 550–556. DOI: 10.1007/978-3-030-59152-6_32.

[Esp98] J. Esparza. “Reachability in Live and Safe Free-Choice Petri Nets is NP-Complete”. In: *Theoretical Computer Science* 198.1-2 (1998), pp. 211–224. DOI: 10.1016/S0304-3975(97)00235-1.

[Fah+09] D. Fahland, C. Favre, B. Jobstmann, et al. “Instantaneous Soundness Checking of Industrial Business Process Models”. In: *Proceedings of the 7th International Conference on Business Process Management, BPM 2009*. Ed. by U. Dayal, J. Eder, J. Koehler, and H. A. Reijers. Vol. 5701. Springer, 2009, pp. 278–293. DOI: 10.1007/978-3-642-03848-8_19.

[Fer95] A. Ferscha. “Qualitative and Quantitative Analysis of Business Workflows Using Generalized Stochastic Petri Nets”. In: *Proceedings of the 9th Austrian-Hungarian Informatics Conference on Workflow Management: Challenges, Paradigms and Products, Connectivity 1994*. Ed. by G. Chroust and A. Benczúr. R. Oldenbourg Verlag GmbH, 1995, pp. 222–234.

[FFV15] C. Favre, D. Fahland, and H. Völzer. “The relationship between workflow graphs and free-choice workflow nets”. In: *Information Systems* 47 (2015), pp. 197–219. DOI: 10.1016/j.is.2013.12.004.

[FR98] M. J. Fischer and M. O. Rabin. “Super-Exponential Complexity of Presburger Arithmetic”. In: *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Ed. by B. F. Caviness and J. R. Johnson. Springer, 1998, pp. 122–135. DOI: 10.1007/978-3-7091-9459-1_5.

[GM12] P. Ganty and R. Majumdar. “Algorithmic verification of asynchronous programs”. In: *ACM Transactions on Programming Languages and Systems* 34.1 (2012), 6:1–6:48. DOI: 10.1145/2160910.2160915.

[GS66] S. Ginsburg and E. H. Spanier. “Semigroups, Presburger Formulas, and Languages”. In: *Pacific Journal of Mathematics* 16.2 (1966), pp. 285–296. URL: https://projecteuclid.org/euclid.pjm/1102994974.

[GS78] T. Gonzalez and S. Sahni. “Flowshop and Jobshop Schedules: Complexity and Approximation”. In: *Operations Research* 26.1 (1978), pp. 36–52. DOI: 10.1287/opre.26.1.36.

[Ha+06] B. Ha, H. A. Reijers, J. Bae, and H. Bae. “An Approximate Analysis of Expected Cycle Time in Business Process Execution”. In: *Business Process Management Workshops, Proceedings of the 2nd International Workshop on Business Process Design, BPD 2006*. Ed. by J. Eder and S. Dustdar. Vol. 4103. Lecture Notes in Computer Science. Springer, 2006, pp. 65–74. DOI: 10.1007/11837862_8.

[Hac76] M. Hack. “Decidability questions for Petri Nets”. PhD thesis. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1976. URL: http://hdl.handle.net/1721.1/27441.

[Hag88] J. N. Hagstrom. “Computational complexity of PERT problems”. In: *Networks* 18.2 (1988), pp. 139–147. DOI: 10.1002/net.3230180206.

[HDK16] T. Hujsa, J. Delosme, and A. M. Kordon. “On Liveness and Reversibility of Equal-Conflict Petri Nets”. In: *Fundamenta Informaticae* 146.1 (2016), pp. 83–119. DOI: 10.3233/FI-2016-1376.

[Hol97] G. J. Holzmann. “The Model Checker SPIN”. In: *IEEE Transactions on Software Engineering* 23.5 (1997), pp. 279–295. DOI: 10.1109/32.588521.

[HP79] J. E. Hopcroft and J. Pansiot. “On the Reachability Problem for 5-Dimensional Vector Addition Systems”. In: *Theoretical Computer Science* 8 (1979), pp. 135–159. DOI: 10.1016/0304-3975(79)90041-0.

[Kar84] N. Karmarkar. “A new polynomial-time algorithm for linear programming”. In: *Combinatorica* 4.4 (1984), pp. 373–396. DOI: 10.1007/BF02579150.

[KAV02] A. Kumar, W. M. P. van der Aalst, and E. M. W. Verbeek. “Dynamic Work Distribution in Workflow Management Systems: How to Balance Quality and Performance”. In: *Journal of Management Information Systems* 18.3 (2002), pp. 157–194. DOI: 10.1080/07421222.2002.11045693.

[KKW10] A. Kaiser, D. Kroening, and T. Wahl. “Dynamic Cutoff Detection in Parameterized Concurrent Programs”. In: *Proceedings of the 22nd International Conference on Computer Aided Verification, CAV 2010*. Ed. by T. Touili, B. Cook, and P. B. Jackson. Vol. 6174. Lecture Notes in Computer Science. Springer, 2010, pp. 645–659. DOI: 10.1007/978-3-642-14295-6_55.

[KKW12] A. Kaiser, D. Kroening, and T. Wahl. “Efficient Coverability Analysis by Proof Minimization”. In: *Proceedings of the 23rd International Conference on Concurrency Theory, CONCUR 2012*. Ed. by M. Koutny and I. Ulidowski. Vol. 7454. Lecture Notes in Computer Science. Springer, 2012, pp. 500–515. DOI: 10.1007/978-3-642-32940-1_35.

[KKW14] A. Kaiser, D. Kroening, and T. Wahl. “A Widening Approach to Multithreaded Program Verification”. In: *ACM Transactions on Programming Languages and Systems* 36.4 (2014), 14:1–14:29. DOI: 10.1145/2629608.

[Klo+13] J. Kloos, R. Majumdar, F. Niksic, and R. Piskac. “Incremental, Inductive Coverability”. In: *Proceedings of the 25th International Conference on Computer Aided Verification, CAV 2013*. Ed. by N. Sharygina and H. Veith. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 158–173. DOI: 10.1007/978-3-642-39799-8_10.

[KU18] A. Kosowski and P. Uznanski. “Brief Announcement: Population Protocols Are Fast”. In: *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing, PODC 2018*. Ed. by C. Newport and I. Keidar. ACM, 2018, pp. 475–477. DOI: 10.1145/3212734.3212788.

[Lam77] L. Lamport. “Proving the Correctness of Multiprocess Programs”. In: *IEEE Transactions on Software Engineering* 3.2 (1977), pp. 125–143. DOI: 10.1109/TSE.1977.229904.

[Lam86] L. Lamport. “The Mutual Exclusion Problem: Part II - Statement and Solutions”. In: *Journal of the ACM* 33.2 (1986), pp. 327–348. DOI: 10.1145/5383.5385.

[Len83] H. W. Lenstra Jr. “Integer Programming with a Fixed Number of Variables”. In: *Mathematics of Operations Research* 8.4 (1983), pp. 538–548. DOI: 10.1287/moor.8.4.538.

[Ler10] J. Leroux. “The General Vector Addition System Reachability Problem by Presburger Inductive Invariants”. In: *Logical Methods in Computer Science* 6.3 (2010). DOI: 10.2168/LMCS-6(3:22)2010.

[Ler11] J. Leroux. “Vector Addition System Reachability Problem: A Short Self-contained Proof”. In: *Proceedings of the 5th International Conference on Language and Automata Theory and Applications, LATA 2011*. Ed. by A. Dediu, S. Inenaga, and C. Martín-Vide. Vol. 6638. Lecture Notes in Computer Science. Springer, 2011, pp. 41–64. DOI: 10.1007/978-3-642-21254-3_3.

[Ler13] J. Leroux. “Vector Addition System Reversible Reachability Problem”. In: *Logical Methods in Computer Science* 9.1 (2013). DOI: 10.2168/LMCS-9(1:5)2013.

[Lip76] R. J. Lipton. *The Reachability Problem Requires Exponential Space*. Research Report 63. Yale University, Department of Computer Science, 1976. URL: http://cpsc.yale.edu/sites/default/files/files/tr63.pdf.

[LMS20] M. Luttenberger, P. J. Meyer, and S. Sickert. “Practical synthesis of reactive systems from LTL specifications via parity games”. In: *Acta Informatica* 57.1-2 (2020), pp. 3–36. DOI: 10.1007/s00236-019-00349-3.

[Lou03] R. Lougee-Heimer. “The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community”. In: *IBM Journal of Research and Development* 47.1 (2003), pp. 57–66. DOI: 10.1147/rd.471.0057.

[LS19] J. Leroux and S. Schmitz. “Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension”. In: *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019*. IEEE, 2019, pp. 1–13. DOI: 10.1109/LICS.2019.8785796.

[LSW06] S. Leue, A. Stefanescu, and W. Wei. “A Livelock Freedom Analysis for Infinite State Asynchronous Reactive Systems”. In: *Proceedings of the 17th International Conference on Concurrency Theory, CONCUR 2006*. Ed. by C. Baier and H. Hermanns. Vol. 4137. Lecture Notes in Computer Science. Springer, 2006, pp. 79–94. DOI: 10.1007/11817949_6.

[LT17] R. Lazic and P. Totzke. “What Makes Petri Nets Harder to Verify: Stack or Data?” In: *Concurrency, Security, and Puzzles - Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday*. Ed. by T. Gibson-Robinson, P. J. Hopcroft, and R. Lazic. Vol. 10160. Lecture Notes in Computer Science. Springer, 2017, pp. 144–161. DOI: 10.1007/978-3-319-51046-0_8.

[Mar+85] M. A. Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. “On Petri Nets with Stochastic Timing”. In: *Proceedings of the International Workshop on Timed Petri Nets*. IEEE, 1985, pp. 80–87.

[MB08] L. M. de Moura and N. Bjørner. “Z3: An Efficient SMT Solver”. In: *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2008*. Ed. by C. R. Ramakrishnan and J. Rehof. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[MCB84] M. A. Marsan, G. Conte, and G. Balbo. “A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems”. In: *ACM Transactions on Computer Systems* 2.2 (1984), pp. 93–122. DOI: 10.1145/190.191.

[MEO19a] P. J. Meyer, J. Esparza, and P. Offtermatt. “Computing the Expected Execution Time of Probabilistic Workflow Nets”. In: *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2019*. Ed. by T. Vojnar and L. Zhang. Vol. 11428. Lecture Notes in Computer Science. Springer, 2019, pp. 154–171. DOI: 10.1007/978-3-030-17465-1_9.

[MEO19b] P. J. Meyer, J. Esparza, and P. Offtermatt. *Computing the Expected Execution Time of Probabilistic Workflow Nets*. 2019. arXiv: 1811.06961 [cs.LO].

[MEO19c] P. J. Meyer, J. Esparza, and P. Offtermatt. *Artifact and instructions to generate experimental results for TACAS 2019 paper: Computing the Expected Execution Time of Probabilistic Workflow Nets*. 2019. DOI: 10.6084/m9.figshare.7831781.v1.

[MEV18a] P. J. Meyer, J. Esparza, and H. Völzer. “Computing the Concurrency Threshold of Sound Free-Choice Workflow Nets”. In: *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2018*. Ed. by D. Beyer and M. Huisman. Vol. 10806. Lecture Notes in Computer Science. Springer, 2018, pp. 3–19. DOI: 10.1007/978-3-319-89963-3_1.

[MEV18b] P. J. Meyer, J. Esparza, and H. Völzer. *Computing the concurrency threshold of sound free-choice workflow nets*. 2018. arXiv: 1802.08064 [cs.LO].

[ML16] P. J. Meyer and M. Luttenberger. “Solving Mean-Payoff Games on the GPU”. In: *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis, ATVA 2016*. Ed. by C. Artho, A. Legay, and D. Peled. Vol. 9938. Lecture Notes in Computer Science. Springer, 2016, pp. 262–267. DOI: 10.1007/978-3-319-46520-3_17.

[MSL18] P. J. Meyer, S. Sickert, and M. Luttenberger. “Strix: Explicit Reactive Synthesis Strikes Back!” In: *Proceedings of the 30th International Conference on Computer Aided Verification, CAV 2018*. Ed. by H. Chockler and G. Weissenbacher. Vol. 10981. Lecture Notes in Computer Science. Springer, 2018, pp. 578–586. DOI: 10.1007/978-3-319-96145-3_31.

[Mur89] T. Murata. “Petri Nets: Properties, Analysis and Applications”. In: *Proceedings of the IEEE* 77.4 (1989), pp. 541–580. DOI: 10.1109/5.24143.

[NB15] S. Navlakha and Z. Bar-Joseph. “Distributed information processing in biological and computational systems”. In: *Communications of the ACM* 58.1 (2015), pp. 94–102. DOI: 10.1145/2678280.

[Opp78] D. C. Oppen. “A $2^{2^{2^{pn}}}$ Upper Bound on the Complexity of Presburger Arithmetic”. In: *Journal of Computer and System Sciences* 16.3 (1978), pp. 323–332. DOI: 10.1016/0022-0000(78)90021-1.

[Pap81] C. H. Papadimitriou. “On the complexity of integer programming”. In: *Journal of the ACM* 28.4 (1981), pp. 765–768. DOI: 10.1145/322276.322287.

[PB83] J. S. Provan and M. O. Ball. “The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected”. In: *SIAM Journal on Computing* 12.4 (1983), pp. 777–788. DOI: 10.1137/0212053.

[Pet62] C. A. Petri. “Kommunikation mit Automaten”. PhD thesis. Universität Hamburg, 1962. URL: http://nbn-resolving.de/urn:nbn:de:gbv:18-228-7-1602.

[Pet81] G. L. Peterson. “Myths About the Mutual Exclusion Problem”. In: *Information Processing Letters* 12.3 (1981), pp. 115–116. DOI: 10.1016/0020-0190(81)90106-X.

[PLD08] J. Pang, Z. Luo, and Y. Deng. “On automatic verification of self-stabilizing population protocols”. In: *Frontiers of Computer Science in China* 2.4 (2008), pp. 357–367. DOI: 10.1007/s11704-008-0040-9.

[Pre29] M. Presburger. “Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt”. In: *Comptes Rendus du I^{er} Congrès des mathématiciens des pays slaves* (1929), pp. 192–201.

[PY84] C. H. Papadimitriou and M. Yannakakis. “The Complexity of Facets (and Some Facets of Complexity)”. In: *Journal of Computer and System Sciences* 28.2 (1984), pp. 244–259. DOI: 10.1016/0022-0000(84)90068-0.

[Rac78] C. Rackoff. “The Covering and Boundedness Problems for Vector Addition Systems”. In: *Theoretical Computer Science* 6 (1978), pp. 223–231. DOI: 10.1016/0304-3975(78)90036-1.

[Rei13] W. Reisig. *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013. DOI: 10.1007/978-3-642-33278-4.

[Reu90] C. Reutenauer. *The Mathematics of Petri Nets*. USA: Prentice-Hall, 1990.

[Rod+17] A. Rodrigues, C. Almeida, D. Saraiva, et al. *Stairway to value: mining a loan application process*. 2017. URL: https://www.win.tue.nl/bpi/lib/exe/fetch.php?media=2017:bpi2017_winner_academic.pdf.

[Sch16] S. Schmitz. “Complexity Hierarchies beyond Elementary”. In: *ACM Transactions on Computation Theory* 8.1 (2016), 3:1–3:36. DOI: 10.1145/2858784.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons Ltd., 1986.

[Sun+09] J. Sun, Y. Liu, J. S. Dong, and J. Pang. “PAT: Towards Flexible Verification under Fairness”. In: *Proceedings of the 21st International Conference on Computer Aided Verification, CAV 2009*. Ed. by A. Bouajjani and O. Maler. Vol. 5643. Lecture Notes in Computer Science. Springer, 2009, pp. 709–714. DOI: 10.1007/978-3-642-02658-4_59.

[SZS10] K. Saeedi, L. Zhao, and P. R. F. Sampaio. “Extending BPMN for Supporting Customer-Facing Service Quality Requirements”. In: *Proceedings of the 8th IEEE International Conference on Web Services, ICWS 2010*. IEEE, 2010, pp. 616–623. DOI: 10.1109/ICWS.2010.116.

[Szy88] B. K. Szymanski. “A simple solution to Lamport’s concurrent programming problem with linear wait”. In: *Proceedings of the 2nd International Conference on Supercomputing, ICS 1988*. Ed. by J. Lenfant. ACM, 1988, pp. 621–626. DOI: 10.1145/55364.55425.

[Thi20] Y. Thierry-Mieg. “Structural Reductions Revisited”. In: *Proceedings of the 41st International Conference on Application and Theory of Petri Nets and Concurrency, PETRI NETS 2020*. Ed. by R. Janicki, N. Sidorova, and T. Chatain. Vol. 12152. Lecture Notes in Computer Science. Springer, 2020, pp. 303–323. DOI: 10.1007/978-3-030-51831-8_15.

[TS93] E. Teruel and M. S. Suárez. “Liveness and Home States in Equal Conflict Systems”. In: *Proceedings of the 14th International Conference on Application and Theory of Petri Nets 1993, PETRI NETS 1993*. Ed. by M. A. Marsan. Vol. 691. Lecture Notes in Computer Science. Springer, 1993, pp. 415–432. DOI: 10.1007/3-540-56863-8_59.

[TS96] E. Teruel and M. S. Suárez. “Structure Theory of Equal Conflict Systems”. In: *Theoretical Computer Science* 153.1&2 (1996), pp. 271–300. DOI: 10.1016/0304-3975(95)00124-7.

[Ull75] J. D. Ullman. “NP-Complete Scheduling Problems”. In: *Journal of Computer and System Sciences* 10.3 (1975), pp. 384–393. DOI: 10.1016/S0022-0000(75)80008-0.

[Val96] A. Valmari. “The State Explosion Problem”. In: *Lectures on Petri Nets I: Basic Models, ACPN 1996*. Ed. by W. Reisig and G. Rozenberg. Vol. 1491. Lecture Notes in Computer Science. Springer, 1996, pp. 429–528. DOI: 10.1007/3-540-65306-6_21.

[Ver+10] E. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. “ProM 6: The Process Mining Toolkit”. In: *Proceedings of the Business Process Management 2010 Demonstration Track, BPM Demo 2010*. Ed. by M. L. Rosa. Vol. 615. CEUR Workshop Proceedings. CEUR-WS.org, 2010. URL: http://ceur-ws.org/Vol-615/paper13.pdf.

[Wol18] K. Wolf. “Petri Net Model Checking with LoLA 2”. In: *Proceedings of the 39th International Conference on Application and Theory of Petri Nets and Concurrency, PETRI NETS 2018*. Ed. by V. Khomenko and O. H. Roux. Vol. 10877. Lecture Notes in Computer Science. Springer, 2018, pp. 351–362. DOI: 10.1007/978-3-319-91268-4_18.

[WW12] H. Wimmel and K. Wolf. “Applying CEGAR to the Petri Net State Equation”. In: *Logical Methods in Computer Science* 8.3 (2012). DOI: 10.2168/LMCS-8(3:27)2012.

[Yen92] H. Yen. “A Unified Approach for Deciding the Existence of Certain Petri Net Paths”. In: *Information and Computation* 96.1 (1992), pp. 119–137. DOI: 10.1016/0890-5401(92)90059-O.

Appendix I

First Author Publications

Paper A

An SMT-based Approach to Fair Termination Analysis. FMCAD, 2015.

This section has been published as a **peer-reviewed conference paper**. The thesis author is **first author** of the paper.

© Javier Esparza and Philipp J. Meyer.

J. Esparza and P. J. Meyer. "An SMT-based Approach to Fair Termination Analysis". In: *Proceedings of the 15th Conference on Formal Methods in Computer-Aided Design, FMCAD 2015*. Ed. by R. Kaivola and T. Wahl. IEEE, 2015, pp. 49–56. DOI: 10.1109/FMCAD.2015.7542252

Summary

We extend the constraint-based approach for coverability analysis from [Esp+14] to fair termination analysis. We use a method based on T-invariants introduced in [EM97] to prove non-existence of some unfair infinite computations, which also allows for verification of certain LTL properties. We extend the method with an iterative refinement through P-components and traps. The method is incomplete, and we give counterexamples where it fails for both types of refinements. We experimentally evaluate our method on a large of set of workflow nets, models of Erlang programs, and models of asynchronous and distributed programs from the literature. Our approach can show termination for 1863 of the 1874 terminating instances and further fair termination for 21 of the 25 fairly terminating instances. Most instances can be verified within a few seconds, except for a few large instances. We also compare with the tool SPIN on the 25 fairly terminating instances. Here, we often perform better on the larger instances.

Contributions of thesis author

Composition and revision of the manuscript. Joint discussion and development of the theoretical results presented in the paper, with the following notable individual contributions: development of the verification method using T-invariants with refinements by P-components and traps; deriving the counterexamples in Fig. 7; implementation of the verification method; conducting the experimental evaluation.

An SMT-based Approach to Fair Termination Analysis

Javier Esparza
Institut für Informatik
Technische Universität München
Garching bei München, Germany
Email: esparza@in.tum.de

Philipp J. Meyer
Institut für Informatik
Technische Universität München
Garching bei München, Germany
Email: meyerphi@in.tum.de

***Abstract*—Algorithms for the coverability problem have been successfully applied to safety checking for concurrent programs. In a former paper (An SMT-based Approach to Coverability Analysis, CAV14) we have revisited a constraint approach to coverability based on classical Petri net analysis techniques and implemented it on top of state-of-the-art SMT solvers. In this paper we extend the approach to fair termination; many other liveness properties can be reduced to fair termination using the automata-theoretic approach to verification. We use T-invariants to identify potential infinite computations of the system, and design a novel technique to discard false positives, that is, potential computations that are not actually executable. We validate our technique on a large number of case studies.**

I. Introduction

In recent years, verification problems for concurrent shared-memory or asynchronous message-passing software have been attacked by means of Petri net techniques. In particular, it has been shown that safety properties or fair termination can be solved by constructing and analyzing the coverability graph of a Petri net, or some related object [1]–[5]. This renewed interest on the coverability problem has led to numerous algorithmic advances for the construction of coverability graphs [4], [6]–[9].

Despite this success, the coverability problem remains computationally expensive [10], since it involves exhaustive state-space exploration. This motivates the study of cheaper incomplete procedures: algorithms much faster than the construction of the coverability graph, which may prove the property true, but also answer "don't know". In a recent paper, the authors, together with other colleagues, have revisited and further developed tests based on the marking equation and traps, two classical Petri net analysis techniques [11]. These techniques allow one to efficiently compute program invariants expressed as constraints of linear arithmetic [12]–[14]. If the states violating the property also correspond to those satisfying a linear constraint, unsatisfiability of the complete constraint system proves the property true. In the test suite analyzed in [11], 83% of the positive problem instances (that is, the instances for which the property holds) could be proved in this way. Moreover, due to advances in SMT-solving, the constraint systems could be solved at a fraction of the cost of state-exploration techniques. So the technique makes sense as a preprocessing that allows to prove many easy cases at low cost; if the technique fails, then we can always resort to complete state-space exploration methods.

In this paper we extend the approach to liveness properties. As in [11], which revisited and expanded previous work, we revisit an idea initially presented in [15], based on the use of transition invariants. Since liveness is typically harder than safety, and the constraint technology of 1997 was very primitive compared to state-of-the-art SMT-solvers, the work of [15] only explored a rather straightforward test, and only considered one case study. In this paper we improve the test of [15], design different implementations, compare their performance, and validate them on numerous case studies coming from different areas: distributed algorithms, workflow processes, Erlang programs, and asynchronous programs.

We conclude this introduction with a brief outline of our technique. Given an infinite execution σ of a Petri net model, let $inf(\sigma)$ be the set of transitions that occur infinitely often in σ. We consider liveness properties such that whether σ satisfies the property or not depends only on $inf(\sigma)$. (This is not an important restriction because, by taking the product of the Petri net model with a suitable Büchi automaton, every LTL property can be reduced to a property of this kind.) We say that a set T of transitions is *feasible* if $T = inf(\sigma)$ for some σ. We use T-invariants (more precisely, T-surinvariants) to extract Boolean constraints that must be satisfied by every feasible set of transitions. However, these constraints are typically quite weak, and have spurious solutions, that is, they are satisfied by unfeasible sets of transitions. So we design a refinement loop that, given a solution, tries to construct an additional constraint that excludes it. If the refinement procedure terminates, then the model satisfies the property.

The paper is structured as follows. Section II contains basic definitions. Section III introduces the main technique. In Section IV and V, we describe two methods to refine the main technique. Section VI contains the experimental evaluation. Finally, Section VII presents conclusions.

II. Preliminaries

A *net* is a triple (P, T, F), where P is a set of *places*, T is a (disjoint) set of *transitions*, and $F : (P \times T) \cup (T \times P) \to \{0, 1\}$ is the *flow function*. For $x \in P \cup T$, the *pre-set* is ${}^\bullet x = \{y \in P \cup T \mid F(y, x) = 1\}$ and the *post-set* is $x^\bullet = \{y \in P \cup T \mid$

ISBN: 978-0-9835678-5-1. Copyright owned jointly by the authors and FMCAD, Inc.

```
    procedure PROCESS 1                 procedure PROCESS 2
    begin                               begin
      bit_1 := false                      bit_2 := false
      while true do                       while true do
p_1:    bit_1 := true             q_1:      bit_2 := true
p_2:    while bit_2 do skip od    q_2:      if bit_1 then
p_3:    (* critical section *)     q_3:        bit_2 := false
        bit_1 := false            q_4:        while bit_1 do skip od
      od                                      goto q_1
    end                                     fi
                                q_5:      (* critical section *)
                                        bit_2 := false
                                      od
                                    end
```

Fig. 1. Lamport's 1-bit algorithm for mutual exclusion [16].

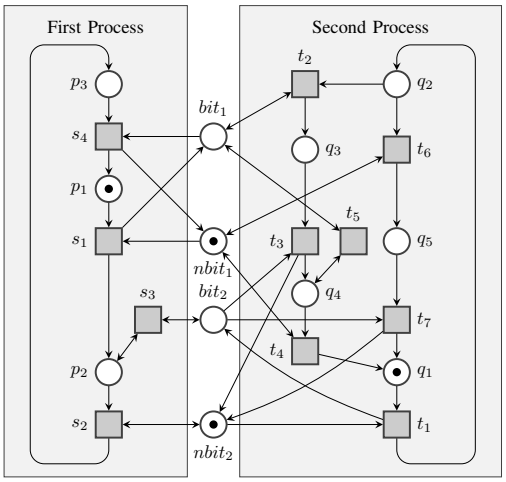


Fig. 2. Petri net for Lamport's 1-bit algorithm.

$F(x,y) = 1\}$. We extend the pre- and post-set to a subset of $P \cup T$ as the union of the pre- and post-sets of its elements. A *subnet* of a Petri net (P,T,F) is a triple (P',T',F') such that $P' \subseteq P$, $T' \subseteq T$, and $F' : (P' \times T') \cup (T' \times P') \to \{0,1\}$ with $F'(x,y) = F(x,y)$. Since F' is completely determined by F,P', and T', we often speak of the subnet (P',T').

A *marking* of a net (P,T,F) is a function $m : P \to \mathbb{N}$. Assuming an enumeration $p_1,\ldots,p_n$ of P, we often identify m and the vector $(m(p_1),\ldots,m(p_n))$. For a subset $P' \subseteq P$ of places, we write $m(P') = \sum_{p \in P'} m(p)$. A *Petri net* is a tuple $N = (P,T,F,m_0)$, where (P,T,F) is a net and m_0 is a marking called the *initial marking*. Petri nets are represented graphically as follows: places and transitions are represented as circles and boxes, respectively. For $x,y \in P \cup T$, there is an arc leading from x to y iff $F(x,y) = 1$. The initial marking is represented by putting $m_0(p)$ black tokens in each place p.

A transition $t \in T$ is *enabled at* m iff $m(p) \geq 1$ for every $p \in {}^\bullet t$. A transition t enabled at m may *fire*, yielding a new marking m' (denoted $m \xrightarrow{t} m'$), where $m'(p) = m(p) + F(t,p) - F(p,t)$.

A sequence of transitions, $\sigma = t_1 t_2 \ldots t_r$ is an *occurrence sequence* of N iff there exist markings $m_1,\ldots,m_r$ such that $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \ldots \xrightarrow{t_r} m_r$. The marking m_r is said to be *reachable* from m_0 by the occurrence of σ (denoted $m_0 \xrightarrow{\sigma} m_r$).

An infinite sequence of transitions, $\sigma = t_1 t_2 \ldots$ is an *infinite occurrence sequence* of N iff every finite prefix of σ is an occurrence sequence of N (denoted $m_0 \xrightarrow{\sigma}$). The set $inf(\sigma)$ contains the transitions occurring infinitely often in σ.

A. Liveness properties

We consider a restricted notion of liveness property. Section II-C briefly sketches how to handle general LTL properties.

A *liveness property* φ of a net $N = (P,T,F,m_0)$ is a Boolean constraint over the free variables T. The property φ holds for an infinite occurrence sequence σ (denoted $\sigma \models \varphi$) iff $I_\sigma \models \varphi$, where $I_\sigma(t) = 1$ if $t \in inf(\sigma)$ else 0. A Petri net N satisfies a property φ (denoted $N \models \varphi$) iff $\sigma \models \varphi$ for every infinite occurrence sequence $m_0 \xrightarrow{\sigma}$. Note that a liveness property is always satisfied if the Petri net has no infinite occurrence sequences. Therefore the property $\varphi = false$ is equivalent to termination of the Petri net. Fair termination properties can be expressed by means of more complex formulas φ.

B. Two examples

As a first example, consider Lamport's 1-bit algorithm for mutual exclusion [16], shown in Fig. 1. Fig. 2 shows a Petri net model for the code. The two grey blocks model the control flow of the two processes. For instance, the token in place p_1 models the current position of process 1 at program location p_1. The four places in the middle of the diagram model the current values of the variables. For instance, a token in place $nbit_1$ indicates that the variable bit_1 is currently set to *false*.

The main liveness property for the processes states that, assuming a fair scheduler that allows both processes to execute actions infinitely often, each process enters the critical section infinitely often. For the first process, this corresponds to the property that every infinite occurrence sequence in which at least one of $s_1,\ldots,s_4$ and one of $t_1,\ldots,t_7$ occur infinitely often, contains infinitely many occurrences of s_2. As a Boolean formula, we get

$$\left(\bigvee_{i=1}^{4} s_i\right) \wedge \left(\bigvee_{j=1}^{7} t_j\right) \Rightarrow s_2$$

For the second process we obtain a similar property.

As a second example, consider the fairly terminating asynchronous program [17] given in Fig. 3. Here, the **post** command is a non-blocking operation for launching a process in parallel. Initially, the process INIT is executed, which sets x to $true$ and launches H. Process H launches new instances of H and G until G sets x to $false$. Assuming a fair scheduler, i.e., one that will execute each process eventually, the program

```
    procedure H          procedure G          procedure INIT
    begin                begin                begin
h:    if x then      g:    x := false           x := true
        post H             end                  post H
        post G                                  end
      fi
    end
```

Fig. 3. Asynchronous program [17].

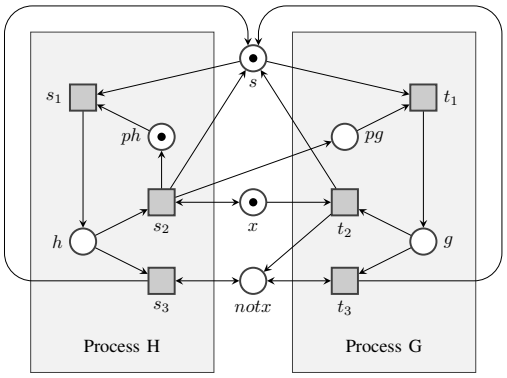


Fig. 4. Petri net for the asynchronous program.

should terminate. This fair termination is the liveness property we want to prove.

Transforming the program into a Petri net gives us the net in Fig. 4. The place s models the scheduler, ph and pg are pending instances of H and G, respectively, and h and g are program locations. The transitions t_1 and s_1 dispatch the processes, while the other transitions exit the processes depending on the value of x. Note that the net is unbounded, as repeatedly firing $s_1 s_2$ puts arbitrarily many tokens in pg.

If the scheduler is fair and continues dispatching instances of H and G infinitely often, the program should terminate, giving us the liveness property $s_1 \wedge t_1 \implies \mathit{false}$, equivalent to $\neg(s_1 \wedge t_1)$.

C. LTL properties

To check general LTL properties we can use the automata-theoretic approach. Given a property φ, we construct the product of the Petri net model of the system and a Büchi automaton for $\neg\varphi$. The product yields a new Petri net with a set of accepting places. The initial net violates the property iff the product net has an infinite sequence σ such that $\mathit{inf}(\sigma)$ contains at least one of the input transitions of the accepting places. A detailed construction can be found in [15].

III. T-SURINVARIANTS

We present a procedure, called LIVENESS, which checks a sufficient condition for a given Petri net to satisfy a liveness property. The condition is unsatisfiability of an appropriate linear arithmetic formula.

Definition 1 (Incidence matrix)**.** The *incidence matrix* C of a Petri net N is a $|P| \times |T|$ matrix given by

$$C(p,t) = F(t,p) - F(p,t)$$

Definition 2 (T-surinvariant)**.** A vector $X : T \to \mathbb{Z}$ is a *T-surinvariant* of a Petri net N iff $C \cdot X \geq 0$. If moreover $C \cdot X = 0$, then X is a *T-invariant*.

A T-surinvariant X is *semi-positive* iff $X \geq 0$ and $X \neq 0$. The *support* of a T-surinvariant X is given by $\|X\| = \{t \in T \mid X(t) > 0\}$.

Loosely speaking, X is a surinvariant if for every place p and for every occurrence sequence $m \xrightarrow{\sigma} m'$, if σ fires each transition t exactly $X(t)$ times, then $m(p) \leq m'(p)$, that is, the number of tokens in p can only increase. The following theorem, where we identify X with the multiset of transitions containing each $t \in T$ exactly $X(t)$ times, shows that the T-surinvariants of a Petri net provide information about its infinite runs.

Theorem 1. *[13], [14] Let σ be an infinite sequence of transitions and N a Petri net. If σ is an infinite occurrence sequence of N, then there is a semi-positive T-surinvariant X satisfying $\|X\| = \mathit{inf}(\sigma)$.*

Proof. Let σ' be a suffix of σ containing only transitions of $\mathit{inf}(\sigma)$, and let $\sigma' = \sigma'_1\sigma'_2\sigma'_3\ldots$ such that each σ'_i contains every transition of $\mathit{inf}(\sigma)$ at least once. Since σ is an infinite occurrence sequence of N, there exist markings $m_1, m_2, m_3, \ldots$ such that $m_1 \xrightarrow{\sigma'_1} m_2 \xrightarrow{\sigma'_2} m_3 \xrightarrow{\sigma'_3} \ldots$. By Dickson's lemma, there exist indices $i < j$ such that $m_i \leq m_j$. Let X be the Parikh vector of $\sigma'_i \ldots \sigma'_{j-1}$, i.e., the vector assigning to each transition its number of occurrences in the sequence. By the definition of the firing rule and the incidence matrix C, for every place p we have $m_j(p) - m_i(p) = \sum_{t \in T} C(p,t)X(t)$ or, in matrix form, $m_j - m_i = C \cdot X$. Since $m_j \geq m_i$, we have $m_j - m_i \geq 0$, and so X is a semi-positive T-surinvariant. Since $\sigma'_i \ldots \sigma'_{j-1}$ contains all transitions of $\mathit{inf}(\sigma)$, we have $\|X\| = \mathit{inf}(\sigma)$. □

However, a T-surinvariant does not guarantee the existence of a corresponding occurrence sequence. Consider the net in Fig. 4. The multiset $X = \{s_1, s_2, t_1, t_3\}$ is a semi-positive T-invariant, but, as we will see later, no infinite occurrence sequence σ satisfies $\mathit{inf}(\sigma) = \{s_1, s_2, t_1, t_3\}$. We say that a T-surinvariant X is *realizable* if there is an infinite occurrence sequence σ with $\|X\| = \mathit{inf}(\sigma)$.

For a T-surinvariant X and a liveness property φ, we denote by $\varphi(X)$ the constraint of linear arithmetic obtained by substituting $X(t) > 0$ for every occurrence of t in φ. So, for instance, if $\varphi = t_1 \vee t_2$, then $\varphi(X) = X(t_1) > 0 \vee X(t_2) > 0$. By Theorem 1, if there is an infinite sequence σ such that $\sigma \models \varphi$, then there is also a semi-positive T-surinvariant X such that $\varphi(X)$ holds. Taking the contrapositive, we have: if no semi-positive T-surinvariant X satisfies $\neg\varphi(X)$, then no sequence σ satisfies $\neg\varphi$, and so $N \models \varphi$. This directly leads to a semi-decision procedure for checking if a liveness property

ISBN: 978-0-9835678-5-1. Copyright owned jointly by the authors and FMCAD, Inc.

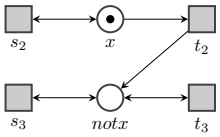


Fig. 5. Subnet of the net of Fig. 4.

φ is a property of a Petri net N: If the following constraints are unsatisfiable, then $N \models \varphi$.

$$\mathcal{C}(N,\varphi) :: \begin{cases} C \cdot X \geq 0 & \text{T-surinvariant condition} \\ X \geq 0 & \text{non-negativity condition} \\ X \neq 0 & \text{non-zero condition} \\ \neg\varphi(X) & \text{property condition} \end{cases} \quad (1)$$

In practice, the procedure is very efficient, but often fails to prove the property. As an example, consider Lamport's algorithm. The negation of the fairness property for the first process yields

$$(s_1 \vee s_2 \vee s_3 \vee s_4) \wedge (t_1 \vee t_2 \vee t_3 \vee t_4 \vee t_5 \vee t_6 \vee t_7) \wedge \neg s_2$$

which corresponds to runs of the system where both processes are executed infinitely often, but where the first process never enters the critical section. However, $X = \{s_3, t_5\}$ is a solution to the constraints (1). The solution corresponds to the processes being stuck in the locations p_2 and q_4 while executing the **skip** commands. For this reason, in the next section we revisit an idea of [15] which leads to a more precise set of constraints.

IV. Refining T-surinvariants with P-components

The method Liveness can be strengthened by discarding T-surinvariants which are not realizable.

Consider again the net of Fig. 4. Recall that $X = \{s_1, s_2, t_1, t_3\}$ is a semi-positive T-invariant. We prove that it is not realizable. Consider the subnet $N' = (P', T')$, where $P' = \{x, notx\}$ and $T' = \{s_2, t_2, s_3, t_3\}$, shown in Fig. 5.

Inspection of the subnet shows that firing a transition does not change the total number of tokens in P'. For example, firing t_2 takes a token from x, but adds a token to $notx$. So this number is always equal to 1, and so it makes sense to speak of "the" token of N'. Assume now that X is realized by some infinite sequence σ, i.e., $inf(\sigma) = \|X\|$. Since both s_2 and t_3 occur infinitely often in σ, there are sequences $\sigma_1, \sigma_2, \sigma_3$ such that $\sigma = \sigma_1\, s_2\, \sigma_2\, t_3 \sigma_3$, and $\sigma_2 \in \|X\|^*$. After the occurrence of s_2 the token of N' is on x, and before the occurrence of t_3 it is on $notx$. But σ_2 cannot "move" the token from x to $notx$, as it does not contain any occurrence of t_2 (because $t_2 \notin \|X\|$). So we reach a contradiction, and σ does not exist.

In the rest of the section we show how to automatically search for proofs of non-realizability like this. We need the notion of a P-component of a net.

Definition 3 (P-component)**.** A *P-component* of a net $N = (P, T, F)$ is a subnet $N' = (P', T')$ such that $P' \neq \emptyset$ and $|t^\bullet \cap P'| = |{}^\bullet t \cap P'| = 1$ for all $t \in T'$ and $T' = P'^\bullet \cup {}^\bullet P'$ (where pre- and post-sets are taken with respect to N).

The subnet of Fig. 5 is a P-component. Note that the number of tokens in a P-component never changes, i.e., $m_0(P) = m(P)$ for all $m_0 \xrightarrow{\sigma} m$. Therefore, if initially a P-component only contains one token, then we know that the token will stay in the P-component.

Lemma 2. *Let X be a T-surinvariant of a Petri net N. If N has a P-component (P', T') such that $m_0(P') = 1$, and the subnet $(P', T' \cap \|X\|)$ is not strongly connected, then X is not realizable.*

Proof. (Sketch.) If $(P', T' \cap \|X\|)$ is not strongly connected, then by the definition of P-component there are two transitions $t_1, t_2 \in T' \cap \|X\|$ such that no path of $(P', T' \cap \|X\|)$ leads from t_1 to t_2. So the token of (P', T') cannot be transported from the output place of t_1 in $(P', T' \cap \|X\|)$ to the input place of t_2 in $(P', T' \cap \|X\|)$ by firing transitions of X only. Since every sequence realizing X must fire both t_1 and t_2 infinitely often, no such sequence exists. □

Lemma 2 provides a refinement condition. To find such a refinement, we encode the condition as a conjunction of linear arithmetic constraints. A pair (P', T') is the set of places and transitions of a P-component such that $m_0(P') = 1$ iff it satisfies these constraints:

$$\forall t \in T' : |t^\bullet \cap P'| = 1 \qquad P'^\bullet \cup {}^\bullet P' = T'$$
$$\forall t \in T' : |{}^\bullet t \cap P'| = 1 \qquad m_0(P') = 1$$

For the strong connectedness condition, we use that a graph (V, E) is not strongly connected iff there is a partition $V = V_1 \uplus V_2$ such that no edge $(v, v') \in E$ satisfies $v \in V_1, v' \in V_2$. In our case, V is the set $T' \cap \|X\|$, and E is the set of pairs (t_1, t_2) such that some place $p \in P'$ satisfies $(t_1, p), (p, t_2) \in F'$. So $(P', T' \cap \|X\|)$ is not strongly connected iff the following constraints are satisfiable:

$$T' \cap \|X\| = T_1 \uplus T_2 \qquad T_1 \neq \emptyset$$
$$(T_1^\bullet \cap P')^\bullet \cap \|X\| \subseteq T_1 \qquad T_2 \neq \emptyset$$

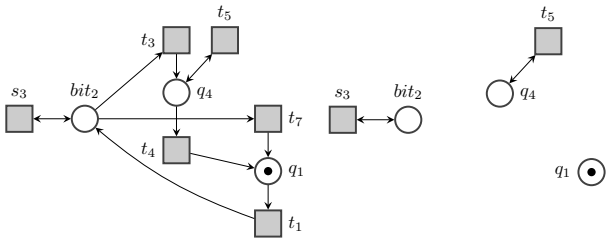
These constraints can be encoded by introducing an array of variables with range $\{0, 1\}$ for each set of places or transitions. For example, the constraint $\forall t \in T' : |t^\bullet \cap P'| = 1$ translates to the linear arithmetic constraint

$$\bigwedge_{t \in T} \left[T'(t) = 1 \implies \sum_{p \in t^\bullet} P'(p) = 1 \right]$$

where $(T'(t1), \ldots, T'(t_n))$ is the array of Boolean variables for the set T'.

If the constraints above are satisfiable for a given T-surinvariant X, then X is not realizable. We can exclude X (and any other T-surinvariant whose support has the same intersection with the P-component as $\|X\|$) by adding the constraint:

ISBN: 978-0-9835678-5-1. Copyright owned jointly by the authors and FMCAD, Inc.



(a) P-component (P', T') (b) Subnet $(P', T' \cap \|X\|)$

Fig. 6. P-component and subnet of the Petri net for Lamport's algorithm.

$$\delta ::= \left[\bigvee_{t \in T_1} t\right] \wedge \left[\bigvee_{t \in T_2} t\right] \implies \bigvee_{t \in T' \setminus \|X\|} t \qquad (2)$$

to the set of constraints (1). We can iterate the process, until either the constraints are unsatisfiable, which means successfully proving the property, or no further P-components can be found to discard a T-surinvariant, which means failure.

For example, for Lamport's algorithm and the fairness property for the first process, the constraints (1) have the solution $X = \{s_3, t_5\}$. However, since $(P', T') = (\{bit_2, q_1, q_4\}, \{s_3, t_1, t_3, t_4, t_5, t_7\})$ is a P-component and $T_1 = \{s_3\}$, $T_2 = \{t_5\}$ satisfy the constraints above, we get that X is not realizable. The P-component (P', T') and the subnet $(P', T' \cap \|X\|)$ are shown in Fig. 6. We can immediately see that the token cannot be transported from the output place of s_3 to the input place of t_5.

We add the refinement constraint

$$s_3 \wedge t_5 \implies t_1 \vee t_3 \vee t_4 \vee t_7$$

to the set (1) and check again for satisfiability. The new set is still satisfiable with the solution $X = \{s_3, t_1, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$. In a second refinement step we find a P-component with $\{nbit_1, p_2, p_3\}$ as set of places, and add the refinement constraint

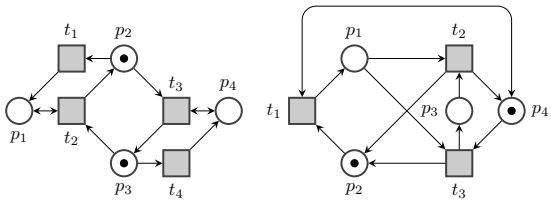
$$s_3 \wedge (t_4 \vee t_6) \implies s_1 \vee s_2 \vee s_4,$$

after which the constraints (1) are unsatisfiable, and we conclude that the fairness property for the first process holds.

For the second example (Fig. 3 and 4), we considered the fair termination property $\varphi = \neg(s_1 \wedge t_1)$. After adding $\neg\varphi = s_1 \wedge t_1$ to the constraints (1), we obtain a solution $X = \{s_1, s_2, t_1, t_3\}$. With the P-component $(P', T') = (\{x, notx\}, \{s_2, s_3, t_2, t_3\})$ and the partition $T_1 = \{s_2\}$ and $T_2 = \{t_3\}$, we can discard this T-invariant as unrealizable and obtain the refinement constraint

$$s_2 \wedge t_3 \implies s_3 \vee t_2,$$

after which the constraints (1) are unsatisfiable and we can prove fair termination.



(a) Net without P-components (b) Net without unmarked traps

Fig. 7. Terminating Petri nets for which refinement is insufficient.

V. Refining T-surinvariants with Traps

For some Petri nets, refinement with P-components is not sufficient for discarding unrealizable T-surinvariants. For example, we cannot prove the properties for the leader election algorithm by Dolev, Klawe and Rodeh [18] or the mutual exclusion algorithm by Szymanski [19]. These nets are too big to give as an example, but consider instead the net in Fig. 7a, which is similar to a subnet of the net for the leader election algorithm. The net has no infinite occurrence sequences and we would like to prove termination. The multiset $X = \{t_2, t_3\}$ is a T-surinvariant, but the net has no P-components, so we cannot refine the constraints. To solve this problem we develop a refinement technique based on traps.

Definition 4 (Trap)**.** A *trap* is a set of places $S \subseteq P$ such that $S^\bullet \subseteq {}^\bullet S$.

It follows immediately from the definition that marked traps stay marked: if a trap S is marked at some marking m, i.e. $m(S) > 0$, then it is also marked at all markings m' reachable from m, because every transition taking tokens from S also adds at least one token to S.

Given a T-surinvariant X, we consider the subnet $(P', T') = (\|X\|^\bullet, \|X\|)$. In the example of Fig. 7a, (P', T') is obtained by removing transitions t_1 and t_4, together with their input and output arcs. Assume $\|X\|$ is realized by an infinite occurrence sequence σ. Then there are sequences σ', σ'' such that $\sigma = \sigma'\sigma''$ and $\sigma'' \in \|X\|^\omega$. Since every place P' has an input transition in $\|X\|$, every place of P' gets marked during the execution of σ'', and therefore every trap of (P', T') becomes eventually marked. So we have the following lemma:

Lemma 3. *Let $N = (P, T, F, m_0)$ be a net and let $\|X\|$ be a realizable T-surinvariant. Then some marking m reachable from m_0 in N marks every trap of the subnet $(P', T') = (\|X\|^\bullet, \|X\|)$.*

By this lemma, if we show that no reachable marking marks every trap of (P', T'), then X is unrealizable. We use an iterative approach. Given a set of traps $\mathcal{Q}$, using the technique of [11] we can construct a set of constraints satisfied by every reachable marking that marks every trap of $\mathcal{Q}$.[1] If the constraints are satisfiable, then we extract from the solution a

[1]The constraints express that a solution m satisfies the marking equation and that $m(S) > 0$ for every trap $S \in \mathcal{Q}$.

marking m that marks all traps in $\mathcal{Q}$. Since m may not mark all traps, we search for a new trap $S \notin \mathcal{Q}$ not marked at m. If we find such S, we set $\mathcal{Q} = \mathcal{Q} \cup \{S\}$ and iterate, otherwise we give up. If the constraints are unsatisfiable, then no reachable marking marks all traps in $\mathcal{Q}$, which implies that X is not realizable. We can then add a new constraint excluding any solution with the same support as X. However, we can do better, and add a stronger constraint. Since we have shown that no infinite occurrence sequence σ can reach a marking that simultaneously marks all traps of $\mathcal{Q}$, we choose a constraint expressing that if $inf(\sigma)$ contains transitions marking all traps of (P', T'), then it must also contain at least one transition that empties a trap of (P', T'). (Of course, such a transition cannot belong to T', it must be a transition of $T \setminus T'$.)

$$\delta ::= \bigwedge_{S \in \mathcal{Q}} \left[\bigvee_{t \in {}^\bullet S} t \right] \implies \bigvee_{S \in \mathcal{Q}} \left[\bigvee_{t \in S^\bullet \setminus {}^\bullet S} t \right] \tag{3}$$

For example, for the Petri net in Fig. 7a, the method LIVENESS returns $X = \{t_2, t_3\}$ as a T-surinvariant. The corresponding subnet is $(P', T') = (\{p_1, p_2, p_3, p_4\}, \{t_2, t_3\})$. Initially, for $\mathcal{Q} = \emptyset$, we can take the initial marking m_0. In m_0, the trap $S_1 = \{p_1\}$ of (P', T') is unmarked. We search for a marking m satisfying the marking equation and $m(p_1) \geq 1$, and obtain as solution $m_1 = (1, 0, 1, 0)$. At this marking the trap $S_2 = \{p_4\}$ is unmarked. So we search for a marking m satisfying the marking equation, $m(p_1) \geq 1$ and $m(p_4) \geq 1$, and obtain as solution $m_2 = (1, 0, 0, 1)$. At this marking the trap $S_3 = \{p_2, p_3\}$ is unmarked. We search for a marking m satisfying the marking equation, $m(p_1) \geq 1$, $m(p_4) \geq 1$, and $m(p_2) + m(p_3) \geq 1$, and obtain that the constraints are unsatisfiable. So we generate the refinement constraint

$$(t_1 \vee t_2) \wedge (t_3 \vee t_4) \wedge (t_2 \vee t_3) \implies t_1 \vee t_4,$$

which excludes $\{t_2, t_3\}$. In fact, the additional constraint turns out to exclude not only $\{t_2, t_3\}$, but all T-surinvariants, which proves termination of the Petri net.

The refinement with traps is a generalization of the refinement with P-components. For a T-surinvariant X, assume there is refinement with a P-component (P', T') and a partition $T_1 \uplus T_2 = T' \cap \|X\|$. In the subnet $(\|X\|^\bullet, \|X\|)$, $S_1 = P' \cap T_1^\bullet$ and $S_2 = P' \cap T_2^\bullet$ are two different traps, and as the P-component has only one token, we can show with the marking equation that S_1 and S_2 cannot be marked at the same time.

Generally, refinement with P-components requires fewer calls to the SMT solver, and is therefore more efficient. In our experiments it is also sufficient for most cases, and if it fails, refinement with traps can be applied afterwards. So we always start with a refinement with P-components, and apply then a refinement with traps if necessary.

Even with both refinements, the method is still incomplete. Consider the Petri net in Fig. 7b, which appears in a Petri net model of the drinking philosopher's problem [20]. The net is terminating, but $X = \{t_1, t_1, t_2, t_3\}$ is a surinvariant (observe that X is a genuine multiset with two copies of t_1). The subnet corresponding to X is the complete net, and every trap is initially marked, so no refinement can be found.

If the property does not hold, our method fails and returns a surinvariant that cannot be excluded by our refinements. For example, for Lamport's algorithm, the fairness property for the second process is not satisfied, where the negation $\neg\varphi$ is:

$$(s_1 \vee s_2 \vee s_3 \vee s_4) \wedge (t_1 \vee t_2 \vee t_3 \vee t_4 \vee t_5 \vee t_6 \vee t_7) \wedge \neg t_6.$$

After two refinement steps, our method returns the T-surinvariant $X = \{s_1, s_2, s_3, s_4, t_1, t_2, t_3, t_4, t_5\}$, which satisfies $\neg\varphi$ and cannot be further refined. In this case we can use guided state-space exploration [21] to try to identify permutations of X that are actual repeatable occurrence sequences. In this case, $\sigma = s_1 t_1 s_3 t_2 t_3 s_2 t_5 s_4 t_4$ is indeed a matching occurrence sequence which can be repeated infinitely and violates the property.

VI. EXPERIMENTAL EVALUATION

We extended our tool *Petrinizer* [11], implemented on top of the SMT solver Z3 [22], with the method LIVENESS. The method can be used without refinement, with only P-component or trap refinement, or with P-component refinement followed by trap refinement. In addition, the refinement structures can be minimized.

For our evaluation, we had three goals. First, we wanted to measure the success rates on a large number of case studies. The second goal was to investigate the usefulness and necessity of P-components, traps and minimization of them. As a third goal, we wanted to measure the performance of the method and compare it with the model checker SPIN[2] [23].

A. Benchmarks

For the evaluation, we used five different benchmark suites from various sources. The first two suites are workflow nets coming from business processes [24]. One is a collection of SAP reference models [25] and the other consists of IBM business process models [26]. We examined the nets for termination. In total, these suites contain 1976 models, out of which 1836 are terminating.

The third suite contains 50 examples that come from the analysis of Erlang programs [5], found on the website of the Soter tool[3]. Out of these, 33 are terminating.

For the fourth suite, we used classic asynchronous programs that can be scaled in the number of processes. These include a leader election algorithm [18], a snapshot algorithm [27] and three mutual exclusion algorithms [16], [19], [28]. Each of the 5 algorithms is scaled from $n = 2$ to 6 processes, resulting in 25 examples. For the former two distributed algorithms, the property is repeated liveness, i.e., infinitely often electing a leader or taking a snapshot infinitely often, while for the latter three mutual exclusion algorithms it is non-starvation for the first process. These properties all contain a fairness assumption for the scheduler, and they hold for all examples.

[2] http://spinroot.com/
[3] http://mjolnir.cs.ox.ac.uk/soter/

TABLE I
FAIRLY TERMINATING EXAMPLES WITH RATE OF SUCCESS BY DIFFERENT REFINEMENT METHODS.

Benchmark	No ref.	Ref. w/P-co.	Ref. w/traps	Terminating
SAP	1263	1263	1264	1264
IBM	571	571	572	572
Erlang	27	27	27	33
Asynchronous	0	14	20	25
Literature	0	3	5	5
Total	1861	1878	1888	1899

Finally, as the fifth suite, we collected 5 examples from the literature on termination and liveness analysis and modeled them as Petri nets. These are the programs from Fig. 2 in [29], Fig. 3 in [30], Fig. 1(b) in [17] and two variants of the Windows NT Bluetooth driver from [31]. These are all terminating programs.

The Petri nets for these benchmarks vary largely in size. The number of places ranges from 4 to 66950, with a mean of 116 and a median of 38. The number of transitions ranges from 3 to 213626, with a mean of 163 and a median of 30.

We try to prove the fairness property of the asynchronous programs and termination for the examples from the other benchmark suites. In total, we have 1899 examples where the property holds.

B. Rate of success on terminating examples

In Table I, the rate of success with different refinement methods is shown. Even without refinement, we can prove termination of all but 2 of the SAP and IBM examples, and of 27 of the 33 Erlang examples. However, without refinement we can prove none of the 30 examples from the other two suites. Refinement with P-components allows us to prove 14 of the asynchronous and 3 of the literature examples. Additional refinement with traps allows us to prove the 2 remaining SAP and IBM examples, 6 more asynchronous examples, and the remaining examples from the literature suite. In total, we can prove termination for 1888 of the 1899 terminating examples, and at least 80% of the terminating examples of each suite.

C. Usefulness of refinement methods and minimization

Table II presents results on the asynchronous benchmark suite and refinement with and without minimization. Minimization of the refinement components can result in better refinement constraints that exclude more T-surinvariants, at the price of a time overhead, since repeated calls to the SMT solver are needed until a minimal component is found. The default method, R_1, is refinement with P-components and traps without any minimization. Refinement method R_2 minimizes P-components (P', T') by $|P'|$ and traps S by $|S|$. Other criteria were also tested, but there was no optimal one working for all benchmarks. For each method, the number of P-components $|\mathcal{R}|$, number of trap refinements $|\mathcal{Q}|$ and total execution time in seconds for proving the property are given.

We observe cases where we need refinement only with P-components (Snapshot), only with traps (Leader election) or with both (Szymanski at $n = 2$). For Szymanski at $n \geq 3$ we cannot prove the property even with both refinement methods.

TABLE II
COMPARISON OF REFINEMENT WITH AND WITHOUT MINIMIZATION AND RUNTIME COMPARISON WITH SPIN. FOR AN EXECUTION, TO DENOTES EXCEEDING THE TIME LIMIT AND MO EXCEEDING THE MEMORY LIMIT.

		Refinement R_1			Ref. w/ min. R_2			SPIN
Benchmark	n	$\|\mathcal{R}\|$	$\|\mathcal{Q}\|$	T (s)	$\|\mathcal{R}\|$	$\|\mathcal{Q}\|$	T (s)	T (s)
Leader election by Dolev, Klawe and Rodeh [18]	2	0	4	2.53	0	4	2.30	0.69
	3	0	6	8.45	0	6	9.03	0.74
	4	0	8	35.5	0	8	38.4	15.7
	5	0	13	206	0	10	154	MO
	6	0	17	1104	0	12	728	MO
Snapshot algorithm by Bougé [27]	2	2	0	0.35	2	0	0.30	0.31
	3	3	0	0.50	3	0	0.81	0.72
	4	4	0	0.60	4	0	0.91	10.3
	5	5	0	0.73	5	0	1.41	218
	6	6	0	1.82	6	0	1.63	MO
Lamport's 1-bit algorithm for mutual exclusion [16]	2	2	0	0.50	3	0	0.43	0.69
	3	6	0	1.26	6	0	1.63	0.69
	4	12	0	2.83	13	0	5.50	0.92
	5	27	0	9.34	18	0	11.3	10.4
	6	26	0	13.4	23	0	20.6	MO
Peterson's mutual exclusion algorithm [28]	2	1	0	0.37	1	0	0.41	0.69
	3	13	0	6.57	7	0	8.55	0.71
	4	21	0	65.9	18	0	92.5	1.16
	5	285	0	2289	36	0	911	43.5
	6	-	-	TO	-	-	TO	MO
Szymanski's mutual exclusion algorithm [19]	2	21	6	10.9	26	6	17.6	0.70
	3							0.80
	4	Property cannot be proven with refinement for $n \geq 3$.						5.83
	5							347
	6							MO

Minimization with method R_2 saves many refinement steps for Peterson and a few for Lamport and Leader election, while for Szymanski the number of steps increases. The time overhead when no steps are saved is not very large (up to $2\times$).

Our method produces a certificate for fair termination consisting of the P-components $\mathcal{R}$ and traps $\mathcal{Q}$. One can use independent methods to check that $\mathcal{R}$ and $\mathcal{Q}$ are indeed P-components and traps, and that the constraints (1) are unsatisfiable. The size of each P-component and trap is limited by the size of the net. The size of the whole certificate depends on the number of refinement steps, however it is usually much more compact than the whole state space.

D. Performance

All experiments were performed on the same machine, equipped with a Intel Core i7-4810MQ CPU at 2.8 GHz and 16 GB of memory, running Linux 3.18.6 in 64-bit mode. Execution time was limited to 2 hours and memory to 16 GB.

Table II shows the execution times of Petrinizer for the asynchronous benchmark suite and a comparison with SPIN. SPIN was used with a fairness strategy enforced and partial order reduction. Only for the snapshot algorithm, partial order reduction was turned off, as it is not supported together with fairness and the rendezvous operations used in the algorithm. For small examples, SPIN is usually faster. However, as n

ISBN: 978-0-9835678-5-1. Copyright owned jointly by the authors and FMCAD, Inc.

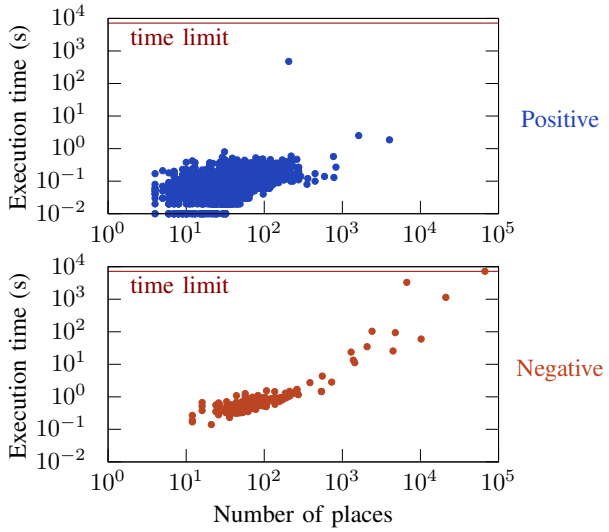


Fig. 8. Execution time in dependence on the number of places for the examples from the benchmark suites SAP, IBM, Erlang and Literature, depending on whether Petrinizer succeeds in proving termination.

grows to 5 or 6, SPIN quickly reaches the memory limit. Here, Petrinizer outperforms SPIN significantly on the examples Leader election, Snapshot and Lamport.

For the other four benchmark suites, Fig. 8 shows the performance of Petrinizer. For the positive examples (i.e., those where we can prove the property), we can prove all but one of the 1868 examples in under 3 seconds. The outlier is from the SAP suite, for which we need 320 refinement steps and 8 minutes. Even the largest positive example from the Erlang suite with 4014 places only needs 1.86 seconds. For the negative examples, Petrinizer performs worse, usually because it performs more refinement steps. However, it terminates in under 3 seconds for all nets with up to 1000 places. Only in one case we reach the time limit of 2 hours (our largest example with 66950 places).

We only need more than 3 refinement steps in one case (an outlier with 320 steps). The number of steps is not correlated to the net size.

VII. Conclusion

Transition invariants and P-components are classical analysis techniques for Petri nets. We have demonstrated that, combined with a state-of-the-art SMT solver, these techniques are very effective in proving fair termination for a large number of common benchmark examples. We have further developed a novel technique based on traps, which allows us to reach a high degree of completeness on these benchmarks. The constraint systems produced by our tool can be used as a certificate of fair termination.

Acknowledgments

We thank Filip Niksic, Corneliu Popeea, and Karsten Wolf for kindly providing examples for our experimental evaluation.

References

[1] A. Kaiser, D. Kroening, and T. Wahl, "Dynamic cutoff detection in parameterized concurrent programs," in *CAV*, 2010, pp. 645–659.
[2] P. Ganty and R. Majumdar, "Algorithmic verification of asynchronous programs," *ACM Trans. Program. Lang. Syst.*, vol. 34, no. 1, p. 6, 2012.
[3] A. Bouajjani and M. Emmi, "Bounded phase analysis of message-passing programs," in *TACAS*, 2012, pp. 451–465.
[4] A. Kaiser, D. Kroening, and T. Wahl, "Efficient coverability analysis by proof minimization," in *CONCUR*, 2012, pp. 500–515.
[5] E. D'Osualdo, J. Kochems, and C.-H. L. Ong, "Automatic verification of Erlang-style concurrency," in *SAS*, 2013, pp. 454–476.
[6] G. Geeraerts, J.-F. Raskin, and L. V. Begin, "Expand, enlarge and check: New algorithms for the coverability problem of WSTS," *J. Comput. Syst. Sci.*, vol. 72, no. 1, pp. 180–203, 2006.
[7] P. Ganty, J.-F. Raskin, and L. Van Begin, "From many places to few: Automatic abstraction refinement for Petri nets," *Fundam. Inform.*, vol. 88, no. 3, pp. 275–305, 2008.
[8] A. Valmari and H. Hansen, "Old and new algorithms for minimal coverability sets," in *Petri Nets*, 2012, pp. 208–227.
[9] J. Kloos, R. Majumdar, F. Niksic, and R. Piskac, "Incremental, inductive coverability," in *CAV*, 2013, pp. 158–173.
[10] C. Rackoff, "The covering and boundedness problems for vector addition systems," *Theor. Comput. Sci.*, vol. 6, pp. 223–231, 1978.
[11] J. Esparza, R. Ledesma-Garza, R. Majumdar, P. Meyer, and F. Niksic, "An SMT-based approach to coverability analysis," in *CAV*, 2014, pp. 603–619.
[12] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.
[13] J. Desel and J. Esparza, *Free Choice Petri Nets*. Cambridge University Press, 1995.
[14] W. Reisig, *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.
[15] J. Esparza and S. Melzer, "Model checking LTL using constraint programming," in *ICATPN*, 1997, pp. 1–20.
[16] L. Lamport, "The mutual exclusion problem - part II: Statement and solutions," *J. ACM*, vol. 33, no. 2, pp. 327–348, 1986.
[17] P. Ganty, R. Majumdar, and A. Rybalchenko, "Verifying liveness for asynchronous programs," in *POPL*, 2009, pp. 102–113.
[18] D. Dolev, M. M. Klawe, and M. Rodeh, "An o(n log n) unidirectional distributed algorithm for extrema finding in a circle," *J. Algorithms*, vol. 3, no. 3, pp. 245–260, 1982.
[19] B. K. Szymanski, "A simple solution to lamport's concurrent programming problem with linear wait," in *ICS*, 1988, pp. 621–626.
[20] K. M. Chandy and J. Misra, "The drinking philosopher's problem," *ACM Trans. Program. Lang. Syst.*, vol. 6, no. 4, pp. 632–646, 1984.
[21] H. Wimmel and K. Wolf, "Applying CEGAR to the Petri net state equation," *Logical Methods in Computer Science*, vol. 8, no. 3, 2012.
[22] L. M. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *TACAS*, 2008, pp. 337–340.
[23] G. J. Holzmann, "The model checker SPIN," *IEEE Trans. Software Eng.*, vol. 23, no. 5, pp. 279–295, 1997.
[24] W. M. P. van der Aalst, "Challenges in business process management: Verification of business processing using petri nets." *Bulletin of the EATCS*, vol. 80, pp. 174–199, 2003.
[25] B. F. van Dongen, M. H. Jansen-Vullers, H. M. W. Verbeek, and W. M. P. van der Aalst, "Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants," *Computers in Industry*, vol. 58, no. 6, pp. 578–601, 2007.
[26] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf, "Instantaneous soundness checking of industrial business process models," in *BPM*, 2009, pp. 278–293.
[27] L. Bougé, "Repeated snapshots in distributed systems with synchronous communications and their implementation in CSP," *Theor. Comput. Sci.*, vol. 49, pp. 145–169, 1987.
[28] G. L. Peterson, "Myths about the mutual exclusion problem," *Inf. Process. Lett.*, vol. 12, no. 3, pp. 115–116, 1981.
[29] B. Cook, A. Podelski, and A. Rybalchenko, "Proving thread termination," in *PLDI*, 2007, pp. 320–330.
[30] A. Podelski and A. Rybalchenko, "Transition invariants," in *LICS*, 2004, pp. 32–41.
[31] S. Qadeer and D. Wu, "KISS: keep it simple and sequential," in *PLDI*, 2004, pp. 14–24.

Paper B

Computing the Concurrency Threshold of Sound Free-Choice Workflow Nets. TACAS, 2018.

This section has been published as a **peer-reviewed conference paper**. The thesis author is **first author** of the paper.

© Philipp J. Meyer, Javier Esparza and Hagen Völzer.

P. J. Meyer, J. Esparza, and H. Völzer. "Computing the Concurrency Threshold of Sound Free-Choice Workflow Nets". In: *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2018*. Ed. by D. Beyer and M. Huisman. Vol. 10806. Lecture Notes in Computer Science. Springer, 2018, pp. 3–19. DOI: 10.1007/978-3-319-89963-3_1

Summary

We consider workflow nets where execution of tasks takes a certain duration and requires a unit of a resource. We analyze the problem of determining the minimum number of resources required to execute a given sound and free-choice workflow net as fast as possible. We show that computing this number is NP-hard even for a very restricted form of free-choice workflow nets, and the fastest execution with this number of resources may not be realizable by an an online scheduler, i.e. requires knowledge on how future choices will be resolved. We then analyze the notion of the *concurrency threshold*, which is the minimum number of resources required to execute the workflow net as fast as possible by any scheduler and with a priori unknown task durations. We show that computing the concurrency threshold is NP-complete, and give two linear programming algorithms to approximate it. We evaluate our algorithms on a set of 642 free-choice workflow nets. They always compute the exact concurrency threshold within milliseconds, while an approach using state-space exploration takes several minutes or exceeds the memory limit in some cases.

Contributions of thesis author

Composition and revision of the manuscript. Joint discussion and development of the theoretical results presented in the paper, with the following notable individual contributions: deriving the complexity results in Theorem 1 and 3 and writing the proofs; design of the example to show Proposition 1; implementation of the approximation algorithms in the tool MACAW; conducting the experimental evaluation.

Computing the Concurrency Threshold of Sound Free-Choice Workflow Nets

Philipp J. Meyer[1(✉)], Javier Esparza[1], and Hagen Völzer[2]

[1] Technical University of Munich, Munich, Germany
{meyerphi,esparza}@in.tum.de
[2] IBM Research, Zurich, Switzerland
hvo@zurich.ibm.com

Abstract. Workflow graphs extend classical flow charts with concurrent fork and join nodes. They constitute the core of business processing languages such as BPMN or UML Activity Diagrams. The activities of a workflow graph are executed by humans or machines, generically called resources. If concurrent activities cannot be executed in parallel by lack of resources, the time needed to execute the workflow increases. We study the problem of computing the minimal number of resources necessary to fully exploit the concurrency of a given workflow, and execute it as fast as possible (i.e., as fast as with unlimited resources).

We model this problem using free-choice Petri nets, which are known to be equivalent to workflow graphs. We analyze the computational complexity of two versions of the problem: computing the resource and concurrency thresholds. We use the results to design an algorithm to approximate the concurrency threshold, and evaluate it on a benchmark suite of 642 industrial examples. We show that it performs very well in practice: It always provides the exact value, and never takes more than 30 ms for any workflow, even for those with a huge number of reachable markings.

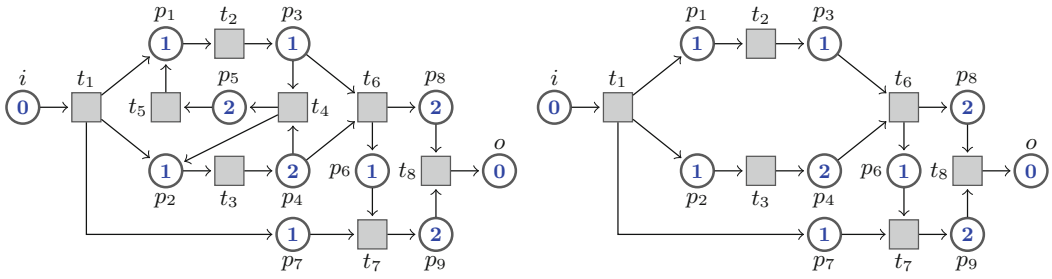
1 Introduction

A *workflow graph* is a classical control-flow graph (or flow chart) extended with concurrent fork and join. Workflow graphs represent the core of workflow languages such as BPMN (Business Process Model and Notation), EPC (Event-driven Process Chain), or UML Activity Diagrams.

In many applications, the activities of an execution workflow graph have to be carried out by a fixed number of *resources* (for example, a fixed number of computer cores). Increasing the number of cores can reduce the minimal runtime of the workflow. For example, consider a simple deterministic workflow (a workflow without choice or merge nodes), which forks into k parallel activities, all of duration 1, and terminates after a join. With an optimal assignment of resources to activities, the workflow takes time k when executed with one resource, time $\lceil k/2 \rceil$ with two resources, and time 1 with k resources; additional resources

© The Author(s) 2018
D. Beyer and M. Huisman (Eds.): TACAS 2018, LNCS 10806, pp. 3–19, 2018.
https://doi.org/10.1007/978-3-319-89963-3_1

bring no further reduction. We call k the *resource threshold*. In a deterministic workflow that forks into two parallel chains of k sequential activities each, one resource leads to runtime $2k$, and two resources to runtime k. More resources do not improve the runtime, and so the resource threshold is 2. Clearly, the resource threshold of a deterministic workflow with k activities is a number between 1 and k. Determining this number can be seen as a scheduling problem. However, most scheduling problems assume a fixed number of resources and study how to optimize the makespan [11,17], while we study how to minimize the number of resources. Other works on resource/machine minimization [5,6] consider interval constraints instead of the partial-order constraints given by a workflow graph.



(a) Sound free-choice workflow net N (b) A run of N

Fig. 1. A sound free-choice workflow net and one of its runs (Color figure online)

Following previous work, we do not directly work with workflow graphs, but with their equivalent representation as *free-choice workflow Petri nets*, which has been shown to be essentially the same model [10] and allows us to directly use a wealth of results of free-choice Petri nets [7]. Figure 1(a) shows a free-choice workflow net. The actual workflow activities, also called *tasks*, which need a resource to execute and which consume time are modeled as the places of the net: Each place p of the net is assigned a time $\tau(p)$, depicted in blue. Intuitively, when a token arrives in p, it must execute a task that takes $\tau(p)$ time units before it can be used to fire a transition. A free choice exists between transitions t_4 and t_6, which is a representation of a choice node (if-then-else or loop condition) in the workflow.

If no choice is present or all choices are resolved, we have a deterministic workflow such as the one in Fig. 1(b). In Petri net terminology, deterministic workflows correspond to the class of marked graphs. Deterministic workflows are common in practice: in the standard suite of 642 industrial workflows that we use for experiments, 63.7% are deterministic. We show that already for this restricted class, deciding if the threshold exceeds a given bound is NP-hard. Therefore, we investigate an over-approximation of the resource threshold, already introduced in [4]: the *concurrency threshold*. This is the maximal number of task places that can be simultaneously marked at a reachable marking. Clearly, if a workflow with concurrency threshold k is executed with k resources, then we can always start the task of a place immediately after a token arrives, and this schedule already

achieves the fastest runtime achievable with unlimited resources. We show that the concurrency threshold can be computed in polynomial time for deterministic workflows.

For workflows with nondeterministic choice, corresponding to free-choice nets, we show that computing the concurrency threshold of free-choice workflow nets is NP-hard, solving a problem left open in [4]. We even prove that the problem remains NP-hard for sound free-choice workflows. Soundness is the dominant behavioral correctness notion for workflows, which rules out basic control-flow errors such as deadlocks. NP-hardness in the sound case is remarkable, because many analysis problems that have high complexity in the unsound case can be solved in polynomial time in the sound case (see e.g. [1,7,8]).

After our complexity analysis, we design an algorithm to compute bounds on the concurrency threshold using a combination of linear optimization and state-space exploration. We evaluate it on a benchmark suite of 642 sound free-choice workflow nets from an industrial source (IBM) [9]. The bounds can be computed in a total of 7 s (over all 642 nets). On the contrary, the computation of the exact value by state-space exploration techniques times out for the three largest nets, and takes 7 min for the rest. (Observe that partial-order reduction techniques cannot be used, because one may then miss the interleaving realizing the concurrency threshold.)

The paper is structured as follows. Section 2 contains preliminaries. Sections 3 and 4 study the resource and concurrency thresholds, respectively. Section 5 presents our algorithms for computing the concurrency bound, and experimental results. Finally, Sect. 6 contains conclusions.

2 Preliminaries

Petri Nets. A *Petri net* N is a tuple (P, T, F) where P is a finite set of places, T is a finite set of transitions ($P \cap T = \emptyset$), and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs. The *preset* of $x \in P \cup T$ is ${}^\bullet x \stackrel{\text{def}}{=} \{y \mid (y, x) \in F\}$ and its *postset* is $x^\bullet \stackrel{\text{def}}{=} \{y \mid (x, y) \in F\}$. We extend the definition of presets and postsets to sets of places and transitions $X \subseteq P \cup T$ by ${}^\bullet X \stackrel{\text{def}}{=} \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet \stackrel{\text{def}}{=} \bigcup_{x \in X} x^\bullet$. A net is *acyclic* if the relation F^* is a partial order, denoted by $\preceq$ and called the *causal order*. A node x of an acyclic net is *causally maximal* if no node y satisfies $x \prec y$.

A *marking* of a Petri net is a function $M \colon P \to \mathbb{N}$, representing the number of tokens in each place. For a set of places $S \subseteq P$, we define $M(S) \stackrel{\text{def}}{=} \sum_{p \in S} M(p)$. Further, for a set of places $S \subseteq P$, we define by M_S the marking with $M_S(p) = 1$ for $p \in S$ and $M_S(p) = 0$ for $p \notin S$.

A transition t is *enabled* at a marking M if for all $p \in {}^\bullet t$, we have $M(p) \geq 1$. If t is enabled at M, it may *occur*, leading to a marking M' obtained by removing one token from each place of ${}^\bullet t$ and then adding one token to each place of $t^\bullet$. We denote this by $M \xrightarrow{t} M'$. Let $\sigma = t_1 t_2 \dots t_n$ be a sequence of transitions. For a marking M_0, σ is an *occurrence sequence* if $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} M_n$ for some markings $M_1, \dots, M_n$. We say that M_n is reachable from M_0 by σ and

denote this by $M_0 \xrightarrow{\sigma} M_n$. The set of all markings reachable from M in N by some occurrence sequence σ is denoted by $\mathcal{R}^N(M)$. A *system* is a pair (N, M) of a Petri net N and a marking M. A system (N, M) is *live* if for every $M' \in \mathcal{R}^N(M)$ and every transition t some marking $M'' \in \mathcal{R}^N(M')$ enables t. The system is *1-safe* if $M'(p) \leq 1$ for every $M' \in \mathcal{R}^N(M)$ and every place $p \in P$.

Convention: Throughout this paper we assume that systems are 1-safe, i.e., we identify "system" and "1-safe system".

Net Classes. A net $N = (P, T, F)$ is a *marked graph* if $|{}^\bullet p| \leq 1$ and $|p^\bullet| \leq 1$ for every place $p \in P$, and a *free-choice net* if for any two places $p_1, p_2 \in P$ either $p_1^\bullet \cap p_2^\bullet = \emptyset$ or $p_1^\bullet = p_2^\bullet$.

Non-sequential Processes of Petri Nets. An (A, B)-*labeled Petri net* is a tuple $N = (P, T, F, \lambda, \mu)$, where $\lambda\colon P \to A$ and $\mu\colon T \to B$ are *labeling functions* over alphabets A, B. The nonsequential processes of a 1-safe system (N, M) are acyclic, (P, T)-labeled marked graphs. Say that a set P'' of places of a (P, T)-labeled acyclic net *enables* $t \in T$ if all the places of P'' are causally maximal, carry pairwise distinct labels, and $\lambda(P'') = {}^\bullet t$.

Definition 1. *Let $N = (P, T, F)$ be a Petri net and let M be a marking of N. The set $\mathcal{NP}(N, M)$ of* nonsequential processes *of (N, M) (*processes *for short) is the set of (P, T)-labeled Petri nets defined inductively as follows:*

- *The (P, T)-labeled Petri net containing for each place $p \in P$ marked at M one place $\widehat{p}$ labeled by p, no other places, and no transitions, belongs to $\mathcal{NP}(N, M)$.*
- *If $\Pi = (P', T', F', \lambda, \mu) \in \mathcal{NP}(N, M)$ and $P'' \subseteq P'$ enables some transition t of N, then the (P, T)-labeled net $\Pi_t = (P' \uplus \widehat{P}, T' \uplus \{\widehat{t}\}, F' \uplus \widehat{F}, \lambda \uplus \widehat{\lambda}, \mu \uplus \widehat{\mu})$, where*
  - *$\widehat{P} = \{\widehat{p} \mid p \in t^\bullet\}$, with $\widehat{\lambda}(\widehat{p}) = p$, and $\widehat{\mu}(\widehat{t}) = t$;*
  - *$\widehat{F} = \{(p'', \widehat{t}) \mid p'' \in P''\} \cup \{(\widehat{t}, \widehat{p}) \mid \widehat{p} \in \widehat{P}\}$;*

  also belongs to $\mathcal{NP}(N, M)$. We say that Π_t extends *Π.*

We denote the minimal and maximal places of a process Π w.r.t. the causal order by $\min(\Pi)$ *and* $\max(\Pi)$*, respectively.*

As usual, we say that two processes are *isomorphic* if they are the same up to renaming of the places and transitions (notice that we rename only the names of the places and transitions, not their labels).

Figure 2 shows two processes of the workflow net in Fig. 1(a). (The figure does not show the names of places and transitions, only their labels.) The net containing the white and grey nodes only is already a process, and the grey places are causally maximal places that enable t_6. Therefore, according to the definition we can extend the process with the green nodes to produce another process. On the right we extend the same process in a different way, with the transition t_4.

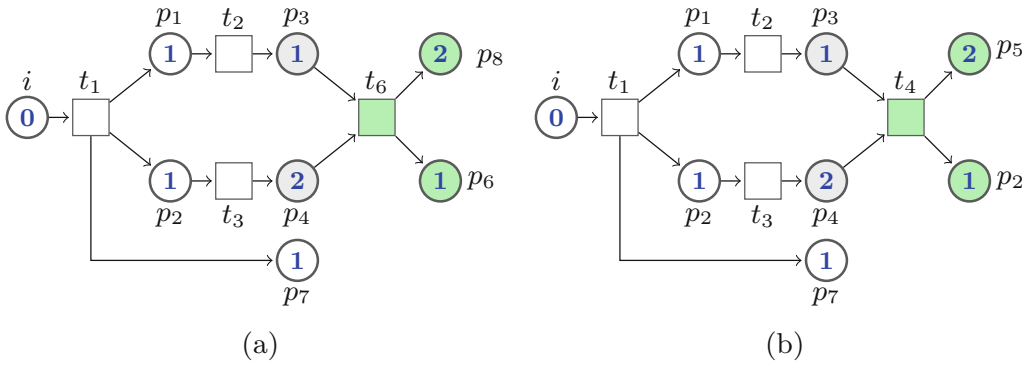


Fig. 2. Nonsequential processes of the net of Fig. 1(a) (Color figure online)

The following is well known. Let $(P', T', F', \lambda, \mu)$ be a process of (N, M):

- For every linearization $\sigma = t'_1 \dots t'_n$ of T' respecting the causal order $\preceq$, the sequence $\mu(\sigma) = \mu(t'_1) \dots \mu(t'_n)$ is a firing sequence of (N, M). Further, all these firing sequences lead to the same marking. We call it the *final marking* of Π, and say that Π leads from M to its final marking.

  For example, in Fig. 2 the sequences of the right process labeled by $t_1t_2t_3t_4$ and $t_1t_3t_2t_4$ are firing sequences leading to the marking $M = \{p_2, p_5, p_7\}$.
- For every firing sequence $t_1 \cdots t_n$ of (N, M) there is a process $(P', T', F', \lambda, \mu)$ such that $T' = \{t'_1, \dots, t'_n\}$, $\mu(t'_i) = t_i$ for every $1 \leq i \leq n$, and $\mu(t'_i) \preceq \mu(t'_j)$ implies $i \leq j$.

Workflow Nets. We slightly generalize the definition of workflow net as presented in e.g. [1] by allowing multiple initial and final places. A *workflow* net is a Petri net with two distinguished sets I and O of *input places* and *output places* such that (a) ${}^\bullet I = \emptyset = O^\bullet$ and (b) for all $x \in P \cup T$, there exists a path from some $i \in I$ to some $o \in O$ passing through x. The markings M_I and M_O are called initial and final markings of N. A workflow net N is *sound* if

- $\forall M \in \mathcal{R}^N(M_I) : M_O \in \mathcal{R}^N(M)$,
- $\forall M \in \mathcal{R}^N(M_I) : (M(O) \geq |O|) \Rightarrow (M = M_O)$, and
- $\forall t \in T : \exists M \in \mathcal{R}^N(M_I) : t$ is enabled at M.

It is well-known that every sound free-choice workflow net is a 1-safe system with the initial marking M_I [2,7]. Given a workflow net according to this definition one can construct another one with one single input place i and output place o and two transitions t_i, t_o with ${}^\bullet t_i = \{i\}, t_i^\bullet = I$ and ${}^\bullet t_o = O, t_o^\bullet = \{o\}$. For all purposes of this paper these two workflow nets are equivalent.

Given a workflow net N, we say that a process Π of (N, M_I) is a *run* if it leads to M_O. For example, the net in Fig. 1(b) is a run of the net in Fig. 1(a).

Petri Nets with Task Durations. We consider Petri nets in which, intuitively, when a token arrives in a place p it has to execute a task taking $\tau(p)$ time units before the token can be used to fire any transition. Formally, we consider tuples $N = (P, T, F, \tau)$ where (P, T, F) is a net and $\tau \colon P \to \mathbb{N}$.

Definition 2. *Given a nonsequential process $\Pi = (P', T', F', \lambda, \mu)$ of (N, M), a time bound t, and a number of resources k, we say that Π* is executable within time t with k resources *if there is a function $f\colon P' \to \mathbb{N}$ such that*

(1) for every $p'_1, p'_2 \in P'$: if $p'_1 \prec p'_2$ then $f(p'_1) + \tau(\lambda(p'_1)) \leq f(p'_2)$;
(2) for every $p' \in P'$: $f(p') + \tau(\lambda(p')) \leq t$; and
(3) for every $0 \leq u < t$ there are at most k places $p' \in P'$ such that $f(p') \leq u < f(p') + \tau(p')$.

We call a function f satisfying (1) a schedule*, a function satisfying (1) and (2) a t -*schedule*, and a function satsifying (1)–(3) a (k, t)-*schedule *of Π.*

Intuitively, $f(p')$ describes the starting time of the task executed at p'. Condition (1) states that if $p'_1 \preceq p'_2$, then the task associated to p'_2 can only start after the task for p'_1 has ended; condition (2) states that all tasks are done by time t, and condition (3) that at any moment in time at most k tasks are being executed. As an example, the process in Fig. 1(b) can be executed with two resources in time 6 with the schedule $i, p_1, p_2 \mapsto 0$; $p_3, p_4 \mapsto 1$; $p_7, p_6 \mapsto 3$, and $p_8, p_9 \mapsto 4$.

Given a process $\Pi = (P', T', F', \lambda, \mu)$ of (N, M) we define the schedule $f_{\min}$ as follows: if $p' \in \min(\Pi)$ then $f_{\min}(p') = 0$, otherwise define $f_{\min}(p') = \max\{f_{\min}(p'') + \tau(\lambda(p'')) \mid p'' \preceq p'\}$. Further, we define the *minimal execution time* $t_{\min}(\Pi) = \max\{f(p') + \tau(\lambda(p'')) \mid p' \in \max(\Pi)\}$. In the process in Fig. 1(b), the schedule $f_{\min}$ is the function that assigns $i, p_1, p_2, p_7 \mapsto 0$, $p_3, p_4 \mapsto 1$, $p_6, p_8 \mapsto 3$, $p_9 \mapsto 4$, and $o \mapsto 6$, and so $t_{\min}(\Pi) = 6$. We have:

Lemma 1. *A process $\Pi = (P', T', F', \lambda, \mu)$ of (N, M) can be executed within time $t_{\min}(\Pi)$ with $|P'|$ resources, and cannot be executed faster with any number of resources.*

Proof. For $k \geq |P'|$ resources condition (3) of Definition 2 holds vacuously. Π is executable within time t iff conditions (1) and (2) hold. Since $f_{\min}$ satisfies (1) and (2) for $t = t_{\min}(\Pi)$, Π can be executed within time $t_{\min}(\Pi)$. Further, $t_{\min}(\Pi)$ is the smallest time for which (1) and (2) can hold, and so Π cannot be executed faster with any number of resources.

3 Resource Threshold

We define the resource threshold of a run of a workflow net, and of the net itself. Intuitively, the resource threshold of a run is the minimal number of resources that allows one to execute it as fast as with unlimited resources, and the resource threshold of a workflow net is the minimal number of resources that allows one to execute *every run* as fast as with unlimited resources.

Definition 3. *Let N be a workflow net, and let Π be a run of N. The* resource threshold *of Π, denoted by $RT(\Pi)$ is the smallest number k such that Π can be executed in time $t_{\min}(\Pi)$ with k resources. A schedule of Π* realizes *the resource threshold if it is a $(RT(\Pi), t_{\min}(\Pi))$-schedule.*

The resource threshold *of N, denoted by $RT(N)$, is defined by $RT(N) = \max\{RT(\Pi) \mid \Pi \text{ is a run of } (N, M_I)\}$. A* schedule of*$N$ is a function that assigns to every process $\Pi \in \mathcal{NP}(N, M)$ a schedule of Π. A schedule of N is a (k,t)-schedule if it assigns to every run Π a (k,t)-schedule of Π. A schedule of N* realizes *the resource threshold if it assigns to every run Π a $(RT(N), t_{\min}(\Pi))$-schedule.*

Example 1. We have seen in the previous section that for the process in Fig. 1(b) we have $t_{\min}(\Pi) = 6$, and a schedule with two resources already achieves this time. So the resource bound is 2. The workflow net of Fig. 1 has infinitely many runs, in which loosely speaking, the net executes t_4 arbitrarily many times, until it "exits the loop" by choosing t_6, followed by t_7 and t_8. It can be shown that all processes have resource threshold 2, and so that is also the resource threshold of the net.

In the rest of the section we obtain two negative results about the result threshold. First, it is difficult to compute: Determining if the resource threshold exceeds a given threshold is NP-complete even for acyclic marked graphs, a very simple class of workflows. Second, we show that even for acyclic free-choice workflow nets the resource threshold may not be realized by any online scheduler.

3.1 Resource Threshold Is NP-complete for Acyclic Marked Graphs

We prove that deciding if the resource threshold exceeds a given bound is NP-complete even for acyclic sound marked graphs. The proof proceeds by reduction from the following classical scheduling problem, proved NP-complete in [18]:

> **Given**: a finite, partially ordered set of jobs with non-negative integer durations, and non-negative integers t and k.
> **Decide**: Can all jobs can be executed with k machines within t time units in a way that respects the given partial order, i.e., a job is started only after all its predecessors have been finished?

More formally, the problem is defined as follows: Given jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, where J_i has duration $\tau(J_i)$ for every $1 \leq i \leq n$, and a partial order $\preceq$ on $\mathcal{J}$, does there exist a function $f \colon \mathcal{J} \to \mathbb{N}$ such that

(1) for every $1 \leq i, j \leq n$: if $J_i \prec J_j$ then $f(J_i) + \tau(J_i) \leq f(J_j)$;
(2) for every $1 \leq i \leq n$: $f(J_i) + \tau(J_i) \leq t$; and
(3) for every $0 \leq u < t$ there are at most k indices i such that $f(J_i) \leq u < f(J_i) + \tau(J_i)$.

These conditions are almost identical to the ones we used to define if a nonsequential process can be executed within time t with k resources. We exploit this to construct an acyclic workflow marked graph that "simulates" the scheduling problem. For the detailed proof, we refer to the full version of this paper [15].

Theorem 1. *The following problem is NP-complete:*

> **Given:** *An acyclic, sound workflow marked graph N, and a number k.*
> **Decide:** *Does $RT(N) \leq k$ hold?*

3.2 Acyclic Free-Choice Workflow Nets May Have no Optimal Online Schedulers

A resource threshold of k guarantees that every run *can* be executed without penalty with k resources. In other words, *there exists* a schedule that achieves optimal runtime. However, in many applications the schedule must be determined at runtime, that is, the resources must be allocated without knowing how choices will be resolved in the future. In order to formalize this idea we define the notion of an *online schedule* of a workflow net N.

Definition 4. *Let N be a Petri net, and let Π and Π' be two processes of (N, M). We say that Π is a* prefix *of Π', denoted by $\Pi \lhd \Pi'$, if there is a sequence $\Pi_1, \ldots, \Pi_n$ of processes such that $\Pi_1 = \Pi$, $\Pi_n = \Pi'$, and Π_{i+1} extends Π_i by one transition for every $1 \leq i \leq n-1$.*

Let f be a schedule of (N, M), i.e., a function assigning a schedule to each process. We say that f is an online schedule *if for every two runs Π_1, Π_2, and for every two prefixes $\Pi_1' \lhd \Pi_1$ and $\Pi_2' \lhd \Pi_2$: If Π_1' and Π_2' are isomorphic, then $f(\Pi_1') = f(\Pi_2')$.*

Intuitively, if Π_1' and Π_2' are isomorphic then they are the same process Π, which in the future can be extended to either Π_1 or Π_2, depending on which transitions occur. In an online schedule, Π is scheduled in the same way, independently of whether it will become Π_1 or Π_2 in the future. We show that even for acyclic free-choice workflow nets there may be no online schedule that realizes the resource threshold. That is, even though for every run it is possible to schedule the tasks with $RT(N)$ resources to achieve optimal runtime, this requires knowing how it will evolve before the execution of the workflow.

Proposition 1. *There is an acyclic, sound free-choice workflow net for which no online schedule realizes the resource threshold.*

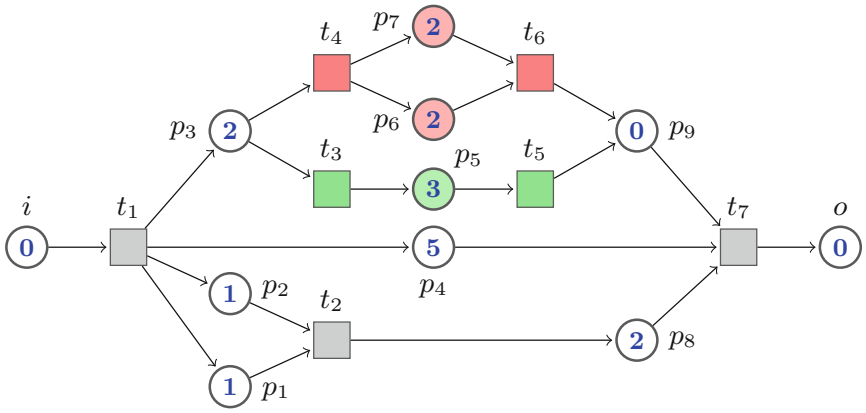


Fig. 3. A workflow net with two runs. No online scheduler for three resources achieves the minimal runtime in both runs. (Color figure online)

Proof. Consider the sound free-choice workflow net (N, M_I) of Fig. 3. It has two runs: Π_g, which executes the grey and green transitions, and Π_r, which executes the grey and red transitions. Their resource thresholds are $RT(\Pi_g) = RT(\Pi_r) = 3$, realized by the schedules f_g and f_r in Fig. 4:

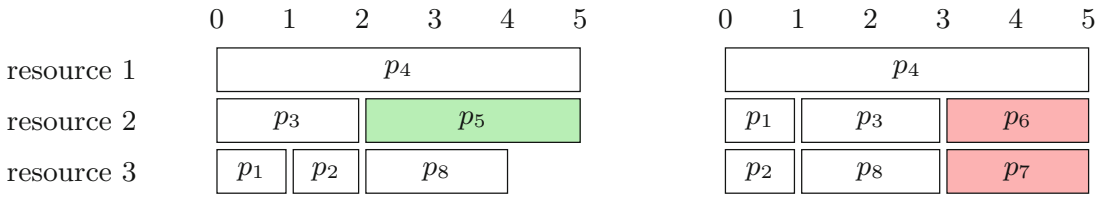


Fig. 4. Schedules f_g and f_r for the two runs Π_g and Π_r of the net of Fig. 3.

Indeed, observe that f_g and f_r execute Π_g and Π_r within time 5, and even with unlimited resources no schedule can be faster because of the task p_4, while two or fewer resources are insufficient to execute either run within time 5.

The schedule of (N, M_I) that assigns f_g and f_r to Π_g and Π_r is not an online schedule. Indeed, the process containing one single transition labeled by t_1 and places labeled by i, p_1, p_2, p_3 is isomorphic to prefixes of Π_g and Π_r. However, we have $f_g(p_3) = 0 \neq 1 = f_r(p_3)$. We now claim:

(a) Every schedule f_g of Π_g that realizes the resource threshold (time 5 with 3 resources) satisfies $f_g(p_3) = 0$.
Indeed, if $f_g(p_3) \geq 1$, then $f_g(p_5) \geq 3$, $f_g(p_9) \geq 6$, and finally $f_g(o) \geq 6$, so f_g does not meet the time bound.

(b) Every schedule f_r of Π_r that realizes the resource threshold (time 5 with 3 resources) satisfies $f_r(p_3) > 0$.
Observe first that we necessarily have $f_r(p_4) = 0$, and so a resource, say R_1, is bound to p_4 during the complete execution of the workflow, leaving two resources left. Assume $f_r(p_3) = 0$, i.e., a second resource, say R_2, is bound to p_3 at time 0, leaving one resource left, say R_3. Since both p_1 and p_2 must be executed before p_8, and only R_3 is free until time 2, we get $f_r(p_8) \geq 2$. So at time 2 we still have to execute p_6, p_7, p_8 with resources R_2, R_3. Therefore, two out of p_6, p_7, p_8 must be executed sequentially by the same resource. Since p_6, p_7, p_8 take 2 time units each, one of the two resources needs time 4, and we get $f_r(o) \geq 6$.

By this claim, at time 0, an online schedule has to decide whether to allocate a resource to p_3 or not, without knowing which of t_3 or t_4 will be executed in the future. If it schedules $f(p_3) = 0$ and later t_4 occurs, then Π_r is executed and the deadline of 5 time units is not met. The same occurs if it schedules $f(p_3) > 0$, and later t_3 occurs.

4 Concurrency Threshold

Due to the two negative results presented in the previous section, we study a different parameter, introduced in [4], called the concurrency threshold. During execution of a business process, information on the resolution of future choices is often not available, and further no information on the possible duration of a task (or only weak bounds) are known. Therefore, the scheduling is performed in practice by assigning a resource to a task at the moment some resource becomes available. The question is: What is the minimal number of resources needed to guarantee the optimal execution time achievable with an unlimited number of resources?

The answer is simple: since there is no information about the duration of tasks, every reachable marking of the workflow net without durations may be also reached for some assignment of durations. Let M be a reachable marking with a maximal number of tokens, say k, in places with positive duration, and let $d_1 \leq d_2 \leq \cdots \leq d_k$ be the durations of their associated tasks. If less than k resources are available, and we do not assign a resource to the task with duration d_k, we introduce a delay with respect to the case of an unlimited number of resources. On the contrary, if the number of available resources is k, then the scheduler for k resources can always simulate the behaviour of the scheduler for an unlimited number of resources.

Definition 5. *Let $N = (P, T, F, I, O, \tau)$ be a workflow Petri net. For every marking M of N, define the* concurrency *of M as* $\mathrm{conc}(M) \stackrel{def}{=} \sum_{p \in D} M(p)$, *where $D \subseteq P$ is the set of places $p \in P$ such that $\tau(p) > 0$. The* concurrency threshold of*N is defined by*

$$CT(N) \stackrel{def}{=} \max\left\{\mathrm{conc}(M) \mid M \in \mathcal{R}^N(M)\right\}.$$

The following lemma follows easily from the definitions.

Lemma 2. *For every workflow net N: $RT(N) \leq CT(N)$.*

Proof. Follows immediately from the fact that for every schedule f of a run of N, there is a schedule g with $CT(N)$ machines such that $g(p) \leq f(p)$ for every place p.

In the rest of the paper we study the complexity of computing the concurrency threshold. In [4], it was shown that the threshold can be computed in polynomial time for regular workflows, a class with a very specific structure, and the problem for the general free-choice case was left open. In Sect. 4.1 we prove that the concurrency threshold of marked graphs can be computed in polynomial time by reduction to a linear programming problem over the rational numbers. In Sect. 4.2 we study the free-choice case. We show that deciding if the threshold exceeds a given value is NP-complete for acyclic, sound free-choice workflow nets. Further, it can be computed by solving the same linear programming problem as in the case of marked graphs, but over the integers. Finally, we show that in the cyclic case the problem remains NP-complete, but the integer linear programming problem does not necessarily yield the correct solution.

4.1 Concurrency Threshold of Marked Graphs

The concurrency threshold of marked graphs can be computed using a standard technique based on the *marking equation* [16]. Given a net $N = (P, T, F)$, define the *incidence matrix* of N as the $|P| \times |T|$ matrix $\boldsymbol{N}$ given by:

$$\boldsymbol{N}(p,t) = \begin{cases} 1 & \text{if } p \in t^\bullet \setminus {}^\bullet t \\ -1 & \text{if } p \in {}^\bullet t \setminus t^\bullet \\ 0 & \text{otherwise} \end{cases}$$

In the following, we denote by $\boldsymbol{M}$ the representation of a marking M as a vector of dimension $|P|$. Let N be a Petri net, and let M_1, M_2 be markings of N. The following results are well known from the literature (see e.g. [16]):

- If M_2 is reachable from M_1 in N, then $\boldsymbol{M_2} = \boldsymbol{M_1} + \boldsymbol{N} \cdot \boldsymbol{X}$ for some integer vector $\boldsymbol{X} \geq 0$.
- If N is a marked graph and $\boldsymbol{M_2} = \boldsymbol{M_1} + \boldsymbol{N} \cdot \boldsymbol{X}$ for some *rational* vector $\boldsymbol{X} \geq 0$, then M_2 is reachable from M_1 in N.
- If N is acyclic and $\boldsymbol{M_2} = \boldsymbol{M_1} + \boldsymbol{N} \cdot \boldsymbol{X}$ for some *integer* vector $\boldsymbol{X} \geq 0$, then M_2 is reachable from M_1 in N.

Given a workflow net $N = (P, T, F, I, O, \tau)$, let $\boldsymbol{D} \colon P \mapsto \mathbb{N}$ be the vector defined by $\boldsymbol{D}(p) = 1$ if $p \in D$ and $\boldsymbol{D}(p) = 0$ if $p \notin D$, where D is the set of places with positive duration. We define the linear optimization problem

$$\ell^N = \max \{ \boldsymbol{D} \cdot \boldsymbol{M} \mid \boldsymbol{M} = \boldsymbol{M_I} + \boldsymbol{N} \cdot \boldsymbol{X}, \boldsymbol{M} \geq 0, \boldsymbol{X} \geq 0 \} \tag{1}$$

Since the solutions of $\boldsymbol{M} = \boldsymbol{M_I} + \boldsymbol{N} \cdot \boldsymbol{X}$ contain all the reachable markings of (N, M_I), we have $\ell^N \geq CT(N)$. Further, using these results above, we obtain:

Theorem 2. *Let N be a workflow net, and let $\ell^N_{\mathbb{Q}}$ and $\ell^N_{\mathbb{Z}}$ be the solution of the linear optimization problem (1) over the rationals and over the integers, respectively. We have:*

- $\ell^N_{\mathbb{Q}} \geq \ell^N_{\mathbb{Z}} \geq CT(N)$;
- *If N is a marked graph, then $\ell_{\mathbb{Q}} = \ell_{\mathbb{Z}} = CT(N)$.*
- *If N is acyclic, then $\ell_{\mathbb{Q}} \geq \ell_{\mathbb{Z}} = CT(N)$.*

In particular, it follows that $CT(N)$ can be computed in polynomial time for marked graphs, acyclic or not. (The result about acyclic nets is used in the next section.)

4.2 Concurrency Threshold of Free-Choice Nets

We study the complexity of computing the concurrency threshold of free-choice workflow nets. We first show that, contrary to numerous other properties for which there are polynomial algorithms, deciding if the concurrency threshold exceeds a given value is NP-complete.

Theorem 3. *The following problem is NP-complete:*

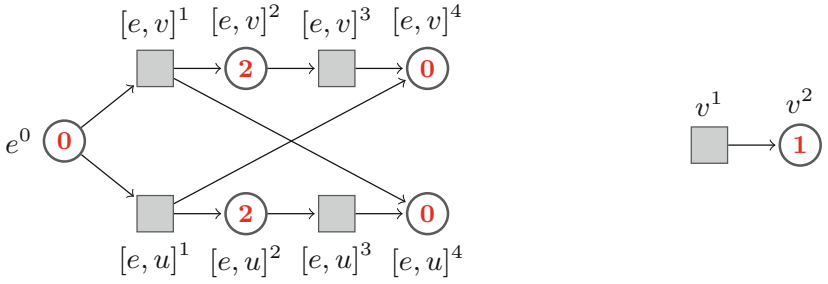
> **Given:** *A sound, free-choice workflow net $N = (P, T, F, I, O)$, and a number $k \leq |T|$.*
> **Decide:** *Is the concurrency threshold of N at least k?*

Proof. A detailed proof can be found in the full version of this paper [15], here we only sketch the argument. Membership in NP is nontrivial, and follows from results of [1,7]. We prove NP-hardness by means of a reduction from Maximum Independent Set (MIS):

> **Given**: An undirected graph $G = (V, E)$, and a number $k \leq |V|$.
> **Decide**: Is there a set $In \subseteq V$ such that $|In| \geq k$ and $\{v, u\} \notin E$ for every $u, v \in In$?

Given a graph $G = (V, E)$, we construct a sound free-choice workflow net N_G in polynomial time as follows:

- For each $e = \{v, u\} \in E$ we add to N_G the "gadget net" N_e shown in Fig. 5(a), and for every node v we add the gadget net N_v shown in Fig. 5(b).
- For every $e = \{v, u\} \in E$, we add an arc from the place $[e, v]^4$ of N_e to the transition v^1 of N_v, and from $[e, u]^4$ to the transition u^1 of N_u.
- The set I of initial places contains the place e^0 of N_e for every edge e; the set O of output places contains the places v^2 of the nets N_v.



(a) Net N_e (b) Net N_v

Fig. 5. Gadgets for the proof of Theorem 3.

It is easy to see that N_G is free-choice and sound, and in [15] we show the result of applying the reduction to a small graph and prove that G has an independent set of size at least k iff the concurrency threshold of (N_G, M_I) is at least $2|E| + k$. The intuition is that for each edge $e \in E$, we fire the transition $[e, u]^1$ where $u \notin In$, and for each $v \in In$, we fire the transition v^1, thus marking one of $[e, u]^2$ or $[e, v]^2$ for each edge $e \in E$ and the place v^2 for each $v \in In$.

4.3 Approximating the Concurrency Threshold

Recall that the solution of problem (1) over the rationals or the integers is always an upper bound on the concurrency threshold for any Petri net (Theorem 2). The question is whether any stronger result holds when the workflows are sound and free-choice. Since computing the concurrency threshold is NP-complete, we cannot expect the solution over the rationals, which is computable in polynomial time, to provide the exact value. However, it could still be the case that the solution over the integers is always exact. Unfortunately, this is not true, and we can prove the following results:

Theorem 4. *Given a Petri net N, let $\ell^N_{\mathbb{Q}}$ and $\ell^N_{\mathbb{Z}}$ be as in Theorem 2.*

(a) There is an acyclic sound free-choice workflow net N such that $CT(N) < \ell^N_{\mathbb{Q}}$.
(b) There is a sound free-choice workflow net N such that and let $CT(N) < \ell^N_{\mathbb{Z}}$.

Proof. For (a), we can take the net obtained by adding to the gadget in Fig. 5(a) a new transition with input places $[e, v]^4$ and $[e, u]^4$, and an output place o with weight 2. We take e^0 as input place. The concurrency threshold is clearly 2, reached, for example, after firing $[e, v]^1$. However, we have $\ell^N_{\mathbb{Q}} = 3$, reached by the rational solution $\boldsymbol{X} = (1/2, 1/2, \ldots, 1/2)$. Indeed, the marking equation then yields the marking M satisfying $M([e, v]^2) = M([e, u]^2) = M(o) = 1/2$.

For (b), we can take the workflow net of Fig. 6. It is easy to see that the concurrency threshold is equal to 1. The marking $\boldsymbol{M}$ that puts one token in each of the two places with weight 1, and no token in the rest of the places, is not reachable from M_I. However, it is a solution of the marking equation, even when solved over the integers. Indeed, we have $\boldsymbol{M} = \boldsymbol{M_I} + \boldsymbol{N} \cdot \boldsymbol{X}$ for $\boldsymbol{X}$=(1,0,1,1,0,0,1). Therefore, the upper bound derived from the marking equation is 2.

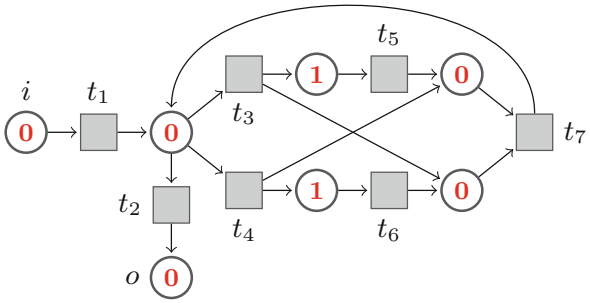


Fig. 6. A sound free-choice workflow net for which the linear programming problem derived from the marking equation does not yield the exact value of the concurrency bound, even when solved over the integers.

5 Concurrency Threshold: A Practical Approach

We have implemented a tool[1] to compute an upper bound on the concurrency threshold by constructing a linear program and solving it by calling the mixed-integer linear programming solver Cbc from the COIN-OR project [14]. Additionally, fixing a number k, we used the state-of-the art Petri net model checker LoLA [19] to both establish a lower bound, by querying LoLA for existence of a reachable marking M with $\mathrm{conc}(M) \geq k$; and to establish an upper bound, by querying LoLA if all reachable markings M' satisfy $\mathrm{conc}(M') \leq k$.

We evaluated the tool on a set of 1386 workflow nets extracted from a collection of five libraries of industrial business processes modeled in the IBM WebSphere Business Modeler [9]. For the concurrency threshold, we set $D = P \setminus O$. These nets also have multiple output places, however with a slightly different semantics for soundness allowing unmarked output places in the final marking. We applied the transformation described in [12] to ensure all output places will be marked in the final marking. This transformation preserves soundness and the concurrency threshold.

All of the 1386 nets in the benchmark libraries are free-choice nets. We selected the sound nets among them, which are 642. Out of those 642 nets, 409 are marked graphs. Out of the remaining 233 nets, 193 are acyclic and 40 cyclic. We determined the exact concurrency threshold of all sound nets with LoLA using state-space exploration. Figure 7 shows the distribution of the threshold.

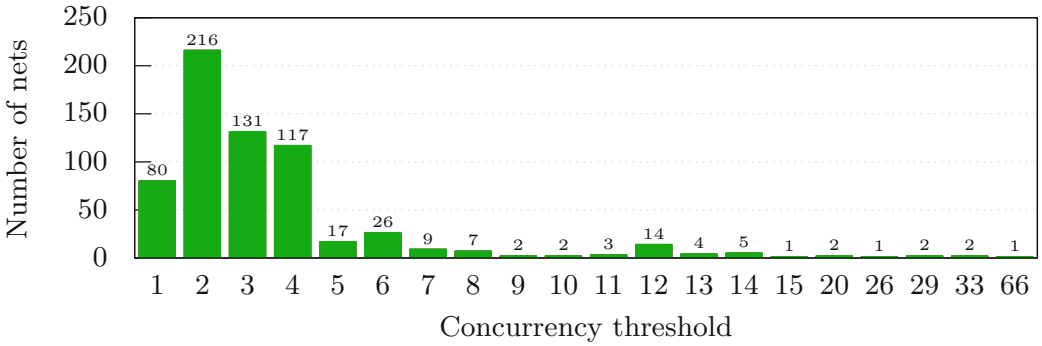


Fig. 7. Distribution of the concurrency threshold of the 642 nets analyzed.

On all 642 sound nets, we computed an upper bound on the concurrency threshold using our tool, both using rational and integer variables. We computed lower and upper bounds using LoLA with the value $k = CT(N)$ of the concurrency threshold. We report the results for computing the lower and upper bound separately.

All experiments were performed on the same machine equipped with an Intel Core i7-6700K CPU and 32 GB of RAM. The results are shown in Table 1.

[1] The tool is available from https://gitlab.lrz.de/i7/macaw.

Using the linear program, we were able to compute an upper bound for all nets in total in less than 7 s, taking at most 30 ms for any single net. LoLA could compute the lower bound for all nets in 6 s LoLA fails to compute the upper bound in three cases due to reaching the memory limit of 32 GB. For the remaining 639 nets, LoLA could compute the upper bound within 7 min in total.

We give a detailed analysis for the 9 nets with a state space of over one million. For three nets with state space of sizes 10^9, 10^{10} and 10^{17}, LoLa reaches the memory limit. For four nets with state spaces between 10^6 and 10^8 and concurrency threshold above 25, LoLA takes 2, 10, 48 and 308 s each. For two nets with a state space of 10^8 and a concurrency threshold of just 11, LoLA can establish the upper bound in at most 20 ms. The solution of the linear program can be computed in all 9 cases in less than 30 ms.

Table 1. Statistics on the size and analyis time for the 642 nets analyzed. The times marked with * exclude the 3 nets where LoLA reaches the memory limit.

	Net size				Analysis time (sec)			
	$\|P\|$	$\|T\|$	$\|\mathcal{R}^N\|$	$CT(N)$	$\ell^N_{\mathbb{Q}}$	$\ell^N_{\mathbb{Z}}$	$CT(N) \geq k$	$CT(N) \leq k$
Median	21	14	16	3	0.01	0.01	0.01	0.01
Mean	28.4	18.6	$3 \cdot 10^{14}$	3.7	0.01	0.01	0.01	0.58*
Max	262	284	$2 \cdot 10^{17}$	66	0.03	0.03	1.18	307.76*

Comparing the values of the upper bound, first we observed that we obtained the same value using either rational or integer variables. The time difference between both was however negligible. Second, quite surprisingly, we noticed that the upper bound obtained from the linear program is exact in all of our cases, even for the cyclic ones. Further, it can be computed much faster in several cases than the upper bound obtained by LoLA and it gives a bound in all cases, even when the state-space exploration reaches its limit. By combining linear programming for the upper bound and state-space exploration for the lower bound, an exact bound can always be computed within a few seconds.

6 Conclusion

Planning sufficient execution resources for a business or production process is a crucial part of process engineering [3,13,20]. We considered a simple version of this problem in which resources are uniform and tasks are not interruptible. We studied the complexity of computing the resource threshold, i.e., the minimal number of resources allowing an optimal makespan. We showed that deciding if the resource threshold exceeds a given bound is NP-hard even for acyclic marked graphs. For this reason, we investigated the complexity of computing the concurrency threshold, an upper bound of the resource threshold introduced in [4]. Solving a problem left open in [4], we showed that deciding if

the concurrency threshold exceeds a given bound is NP-hard for general sound free-choice workflow nets. We then presented a polynomial-time approximation algorithm, and showed experimentally that it computes the *exact* value of the concurrency threshold for all benchmarks of a standard suite of free-choice workflow nets.

References

1. van der Aalst, W.M.P.: Verification of workflow nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63139-9_48
2. van der Aalst, W.M.P.: Workflow verification: finding control-flow errors using Petri-net-based techniques. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_11
3. Bessai, K., Youcef, S., Oulamara, A., Godart, C., Nurcan, S.: Resources allocation and scheduling approaches for business process applications in cloud contexts. In: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, CloudCom 2012, Taipei, Taiwan, 3–6 December 2012, pp. 496–503 (2012)
4. Botezatu, M., Völzer, H., Thiele, L.: The complexity of deadline analysis for workflow graphs with multiple resources. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 252–268. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_15
5. Chuzhoy, J., Codenotti, P.: Resource minimization job scheduling. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX/RANDOM -2009. LNCS, vol. 5687, pp. 70–83. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03685-9_6
6. Chuzhoy, J., Guha, S., Khanna, S., Naor, J.: Machine minimization for scheduling jobs with interval constraints. In: 45th Symposium on Foundations of Computer Science (FOCS 2004), 17–19 October 2004, Rome, Italy, Proceedings, pp. 81–90 (2004)
7. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press, Cambridge (1995)
8. Esparza, J., Hoffmann, P., Saha, R.: Polynomial analysis algorithms for free choice probabilistic workflow nets. In: Agha, G., Van Houdt, B. (eds.) QEST 2016. LNCS, vol. 9826, pp. 89–104. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43425-4_6
9. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous soundness checking of industrial business process models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 278–293. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_19
10. Favre, C., Fahland, D., Völzer, H.: The relationship between workflow graphs and free-choice workflow nets. Inf. Syst. **47**, 197–219 (2015)
11. Hall, N.G., Sriskandarajah, C.: A survey of machine scheduling problems with blocking and no-wait in process. Oper. Res. **44**(3), 510–525 (1996)
12. Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Fundamentals of control flow in workflows. Acta Inf. **39**(3), 143–209 (2003)

13. Liu, L., Zhang, M., Lin, Y., Qin, L.: A survey on workflow management and scheduling in cloud computing. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, 26–29 May 2014, pp. 837–846 (2014)
14. Lougee-Heimer, R.: The common optimization interface for operations research: promoting open-source software in the operations research community. IBM J. Res. Dev. **47**(1), 57–66 (2003)
15. Meyer, P.J., Esparza, J., Völzer, H.: Computing the concurrency threshold of sound free-choice workflow nets. arXiv:1802.08064 [cs.LO] (2018)
16. Murata, T.: Petri nets: properties, analysis, and applications. Proc. IEEE **77**(4), 541–576 (1989)
17. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. Springer, New York (2016). https://doi.org/10.1007/978-1-4614-2361-4
18. Ullman, J.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (1975)
19. Wolf, K.: Generating Petri net state spaces. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 29–42. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73094-1_5
20. Xu, J., Liu, C., Zhao, X.: Resource allocation vs. business process improvement: how they impact on each other. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 228–243. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_18

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Paper C

Computing the Expected Execution Time of Probabilistic Workflow Nets. TACAS, 2019.

This section has been published as a **peer-reviewed conference paper**. The thesis author is **first author** of the paper.

© Philipp J. Meyer, Javier Esparza and Philip Offtermatt.

P. J. Meyer, J. Esparza, and P. Offtermatt. "Computing the Expected Execution Time of Probabilistic Workflow Nets". In: *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2019*. Ed. by T. Vojnar and L. Zhang. Vol. 11428. Lecture Notes in Computer Science. Springer, 2019, pp. 154–171. DOI: 10.1007/978-3-030-17465-1_9

Summary

We consider workflow nets annotated with task durations and probabilities to resolve choices and give them semantics based on Markov decision processes. We then analyze the problem of computing the expected execution time of a given net. We show that for confusion-free workflow nets, the expected execution time is independent of the scheduler, and use a certain *earliest-first* scheduler to give an algorithm to compute the expected execution time of a confusion-free workflow net in exponential time. The algorithm constructs a special Markov chain for this scheduler from the net and computes the expected time by solving a system of linear equations. We further show that computing the expected execution time is #P-hard, even if all task durations are 0 or 1 and all probabilities are 0.5 or 1. We evaluate our algorithm on 642 free-choice workflow nets and a scalable case study. We can compute the expected execution time of the 642 nets within milliseconds, and within minutes for the case study up until reaching more than thousand transitions.

Contributions of thesis author

Composition and revision of the manuscript. Joint discussion and development of the theoretical results presented in the paper, with the following notable individual contributions: deriving the lower bounds in Section 5 and writing the proofs; proposal of the *earliest-first scheduler* to show decidability of the problem of computing the expected time; preparation of the benchmarks and case study for the experimental evaluation.

Computing the Expected Execution Time of Probabilistic Workflow Nets

Philipp J. Meyer(✉), Javier Esparza, and Philip Offtermatt

Technical University of Munich, Munich, Germany
{meyerphi,esparza,offtermp}@in.tum.de

Abstract. Free-Choice Workflow Petri nets, also known as Workflow Graphs, are a popular model in Business Process Modeling.

In this paper we introduce Timed Probabilistic Workflow Nets (TPWNs), and give them a Markov Decision Process (MDP) semantics. Since the time needed to execute two parallel tasks is the maximum of the times, and not their sum, the expected time cannot be directly computed using the theory of MDPs with rewards. In our first contribution, we overcome this obstacle with the help of "earliest-first" schedulers, and give a single exponential-time algorithm for computing the expected time.

In our second contribution, we show that computing the expected time is #P-hard, and so polynomial algorithms are very unlikely to exist. Further, #P-hardness holds even for workflows with a very simple structure in which all transitions times are 1 or 0, and all probabilities are 1 or 0.5.

Our third and final contribution is an experimental investigation of the runtime of our algorithm on a set of industrial benchmarks. Despite the negative theoretical results, the results are very encouraging. In particular, the expected time of every workflow in a popular benchmark suite with 642 workflow nets can be computed in milliseconds. Data or code related to this paper is available at: [24].

1 Introduction

Workflow Petri Nets are a popular model for the representation and analysis of business processes [1,3,7]. They are used as back-end for different notations like BPMN (Business Process Modeling Notation), EPC (Event-driven Process Chain), and UML Activity Diagrams.

The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No. 787367 (PaVeS). Further it is partially supported by the DFG Project No. 273811150 (Negotiations: A Model for Tractable Concurrency).

© The Author(s) 2019
T. Vojnar and L. Zhang (Eds.): TACAS 2019, Part II, LNCS 11428, pp. 154–171, 2019.
https://doi.org/10.1007/978-3-030-17465-1_9

There is recent interest in extending these notations with quantitative information, like probabilities, costs, and time. The final goal is the development of tool support for computing performance metrics, like the average cost or the average runtime of a business process.

In a former paper we introduced Probabilistic Workflow Nets (PWN), a foundation for the extension of Petri nets with probabilities and rewards [11]. We presented a polynomial time algorithm for the computation of the expected cost of free-choice workflow nets, a subclass of PWN of particular interest for the workflow process community (see e.g. [1,10,13,14]). For example, 1386 of the 1958 nets in the most popular benchmark suite in the literature are free-choice Workflow Nets [12].

In this paper we introduce Timed PWNs (TPWNs), an extension of PWNs with time. Following [11], we define a semantics in terms of Markov Decision Processes (MDPs), where, loosely speaking, the nondeterminism of the MDP models absence of information about the order in which concurrent transitions are executed. For every scheduler, the semantics assigns to the TPWN an expected time to termination. Using results of [11], we prove that this expected time is actually independent of the scheduler, and so that the notion "expected time of a TPWN" is well defined.

We then proceed to study the problem of computing the expected time of a sound TPWN (loosely speaking, of a TPWN that terminates successfully with probability 1). The expected cost and the expected time have a different interplay with concurrency. The cost of executing two tasks in parallel is the sum of the costs (cost models e.g. salaries of power consumption), while the execution time of two parallel tasks is the maximum of their individual execution times. For this reason, standard reward-based algorithms for MDPs, which assume additivity of the reward along a path, cannot be applied.

Our solution to this problem uses the fact that the expected time of a TPWN is independent of the scheduler. We define an "earliest-first" scheduler which, loosely speaking, resolves the nondeterminism of the MDP by picking transitions with earliest possible firing time. Since at first sight the scheduler needs infinite memory, its corresponding Markov chain is infinite-state, and so of no help. However, we show how to construct another finite-state Markov chain with additive rewards, whose expected reward is equal to the expected time of the infinite-state chain. This finite-state Markov chain can be exponentially larger than the TPWN, and so our algorithm has exponential complexity. We prove that computing the expected time is #P-hard, even for free-choice TPWNs in which all transitions times are either 1 or 0, and all probabilities are 1 or $1/2$. So, in particular, the existence of a polynomial algorithm implies $\mathsf{P} = \mathsf{NP}$.

In the rest of the paper we show that, despite these negative results, our algorithm behaves well in practice. For all 642 sound free-choice nets of the benchmark suite of [12], computing the expected time never takes longer than a few milliseconds. Looking for a more complicated set of examples, we study a TPWN computed from a set of logs by process mining. We observe that the computation of the expected time is sensitive to the distribution of the execution

time of a task. Still, our experiments show that even for complicated distributions leading to TPWNs with hundreds of transitions and times spanning two orders of magnitude the expected time can be computed in minutes.

All missing proofs can be found in the Appendix of the full version [19].

2 Preliminaries

We introduce some preliminary definitions. The full version [19] gives more details.

Workflow Nets. A *workflow net* is a tuple $\mathbf{N} = (P, T, F, i, o)$ where P and T are disjoint finite sets of *places* and *transitions*; $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs*; $i, o \in P$ are distinguished *initial* and *final* places such that i has no incoming arcs, o has no outgoing arcs, and the graph $(P \cup T, F \cup \{(o, i)\})$ is strongly connected. For $x \in P \cup T$, we write ${}^\bullet x$ for the set $\{y \mid (y, x) \in F\}$ and $x^\bullet$ for $\{y \mid (x, y) \in F\}$. We call ${}^\bullet x$ (resp. $x^\bullet$) the *preset* (resp. *postset*) of x. We extend this notion to sets $X \subseteq P \cup T$ by ${}^\bullet X \stackrel{\text{def}}{=} \cup_{x \in X} {}^\bullet x$ resp. $X^\bullet \stackrel{\text{def}}{=} \cup_{x \in X} x^\bullet$. The notions of marking, enabled transitions, transition firing, firing sequence, and reachable marking are defined as usual. The *initial marking* (resp. *final marking*) of a workflow net, denoted by $\boldsymbol{i}$ (resp. $\boldsymbol{o}$), has one token on place i (resp. o), and no tokens elsewhere. A firing sequence σ is a *run* if $\boldsymbol{i} \xrightarrow{\sigma} \boldsymbol{o}$, i.e. if it leads to the final marking. $Run_{\mathbf{N}}$ denotes the set of all runs of $\mathbf{N}$.

Soundness and 1-safeness. Well designed workflows should be free of deadlocks and livelocks. This idea is captured by the notion of soundness [1,2]: A workflow net is *sound* if the final marking is reachable from any reachable marking.[1] Further, in this paper we restrict ourselves to 1-safe workflows: A marking M of a workflow net $\mathcal{W}$ is *1-safe* if $M(p) \leq 1$ for every place p, and $\mathcal{W}$ itself is *1-safe* if every reachable marking is 1-safe. We identify 1-safe markings M with the set $\{p \in P \mid M(p) = 1\}$.

Independence, concurrency, conflict [22]**.** Two transitions t_1, t_2 of a workflow net are *independent* if ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$, and *dependent* otherwise. Given a 1-safe marking M, two transitions are *concurrent at* M if M enables both of them, and they are independent, and *in conflict at* M if M enables both of them, and they are dependent. Finally, we recall the definition of Mazurkiewicz equivalence. Let $\mathbf{N} = (P, T, F, i, o)$ be a 1-safe workflow net. The relation $\equiv_1 \subseteq T^* \times T^*$ is defined as follows: $\sigma \equiv_1 \tau$ if there are independent transitions t_1, t_2 and sequences $\sigma', \sigma'' \in T^*$ such that $\sigma = \sigma' t_1 t_2 \sigma''$ and $\tau = \sigma' t_2 t_1 \sigma''$. Two sequences $\sigma, \tau \in T^*$ are *Mazurkiewicz equivalent* if $\sigma \equiv \tau$, where $\equiv$ is the reflexive and transitive closure of $\equiv_1$. Observe that $\sigma \in T^*$ is a firing sequence iff every sequence $\tau \equiv \sigma$ is a firing sequence.

Confusion-freeness, free-choice workflows. Let t be a transition of a workflow net, and let M be a 1-safe marking that enables t. The *conflict set of* t

[1] In [2], which examines many different notions of soundness, this is called *easy* soundness.

at M, denoted $C(t, M)$, is the set of transitions in conflict with t at M. A set U of transitions is a *conflict set* of M if there is a transition t such that $U = C(t, M)$. The conflict sets of M are given by $\mathcal{C}(M) \stackrel{\text{def}}{=} \cup_{t \in T} C(t, M)$. A 1-safe workflow net is *confusion-free* if for every reachable marking M and every transition t enabled at M, every transition u concurrent with t at M satisfies $C(u, M) = C(u, M \setminus {}^\bullet t) = C(u, (M \setminus {}^\bullet t) \cup t^\bullet)$. The following result follows easily from the definitions (see also [11]):

Lemma 1 [11]. *Let* $\mathbf{N}$ *be a 1-safe workflow net. If* $\mathbf{N}$ *is confusion-free then for every reachable marking* M *the conflict sets* $\mathcal{C}(M)$ *are a partition of the set of transitions enabled at* M*.*

A workflow net is *free-choice* if for every two places p_1, p_2, if $p_1^\bullet \cap p_2^\bullet \neq \emptyset$, then $p_1^\bullet = p_2^\bullet$. Any free-choice net is confusion-free, and the conflict set of a transition t enabled at a marking M is given by $C(t, M) = ({}^\bullet t)^\bullet$ (see e.g. [11]).

3 Timed Probabilistic Workflow Nets

In [11] we introduced a probabilistic semantics for confusion-free workflow nets. Intuitively, at every reachable marking a choice between two concurrent transitions is resolved nondeterministically by a scheduler, while a choice between two transitions in conflict is resolved probabilistically; the probability of choosing each transition is proportional to its *weight*. For example, in the net in Fig. 1a, at the marking $\{p_1, p_3\}$, the scheduler can choose between the conflict sets $\{t_2, t_3\}$ and $\{t_4\}$, and if $\{t_2, t_3\}$ is chosen, then t_2 is chosen with probability $1/5$ and t_3 with probability $4/5$. We extend Probabilistic Workflow Nets by assigning to each transition t a natural number $\tau(t)$ modeling the time it takes for the transition to fire, once it has been selected.[2]

Definition 1 (Timed Probabilistic Workflow Nets). *A* Timed Probabilistic Workflow Net *(TPWN) is a tuple* $\mathcal{W} = (\mathbf{N}, w, \tau)$ *where* $\mathbf{N} = (P, T, F, i, o)$ *is a 1-safe confusion-free workflow net,* $w \colon T \to \mathbb{Q}_{>0}$ *is a* weight function*, and* $\tau \colon T \to \mathbb{N}$ *is a* time function *that assigns to every transition a duration.*

Timed sequences. We assign to each transition sequence σ of $\mathcal{W}$ and each place p a *timestamp* $\mu(\sigma)_p$ through a *timestamp function* $\mu : T^* \to \mathbb{N}_\perp^P$. The set $\mathbb{N}_\perp$ is defined by $\mathbb{N}_\perp \stackrel{\text{def}}{=} \{\perp\} \cup \mathbb{N}$ with $\perp \leq x$ and $\perp + x = \perp$ for all $x \in \mathbb{N}_\perp$. Intuitively, if a place p is marked after σ, then $\mu(\sigma)_p$ records the "arrival time" of the token in p, and if p is unmarked, then $\mu(\sigma)_p = \perp$. When a transition occurs, it removes all tokens in its preset, and $\tau(t)$ time units later, puts tokens into its postset.

[2] The semantics of the model can be defined in the same way for both discrete and continuous time, but, since our results only concern discrete time, we only consider this case.

Formally, we define $\mu(\epsilon)_i \stackrel{\text{def}}{=} 0$, $\mu(\epsilon)_p \stackrel{\text{def}}{=} \bot$ for $p \neq i$, and $\mu(\sigma t) \stackrel{\text{def}}{=} upd(\mu(\sigma), t)$, where the update function $upd : \mathbb{N}_\bot^P \times T \to \mathbb{N}_\bot^P$ is given by:

$$upd(\boldsymbol{x}, t)_p \stackrel{\text{def}}{=} \begin{cases} \max_{q \in {}^\bullet t} \boldsymbol{x}_q + \tau(t) & \text{if } p \in t^\bullet \\ \bot & \text{if } p \in {}^\bullet t \setminus t^\bullet \\ \boldsymbol{x}_p & \text{if } p \notin {}^\bullet t \cup t^\bullet \end{cases}$$

We then define $tm(\sigma) \stackrel{\text{def}}{=} \max_{p \in P} \mu(\sigma)_p$ as the time needed to fire σ. Further $[\![\boldsymbol{x}]\!] \stackrel{\text{def}}{=} \{p \in P \mid \boldsymbol{x}_p \neq \bot\}$ is the marking represented by a timestamp $\boldsymbol{x} \in \mathbb{N}_\bot^P$.

Example 1. The net in Fig. 1a is a TPWN. Weights are shown in red next to transitions, and times are written in blue into the transitions. For the sequence $\sigma_1 = t_1 t_3 t_4 t_5$, we have $tm(\sigma_1) = 9$, and for $\sigma_2 = t_1 t_2 t_3 t_4 t_5$, we have $tm(\sigma_2) = 10$. Observe that the time taken by the sequences is *not* equal to the sum of the durations of the transitions.

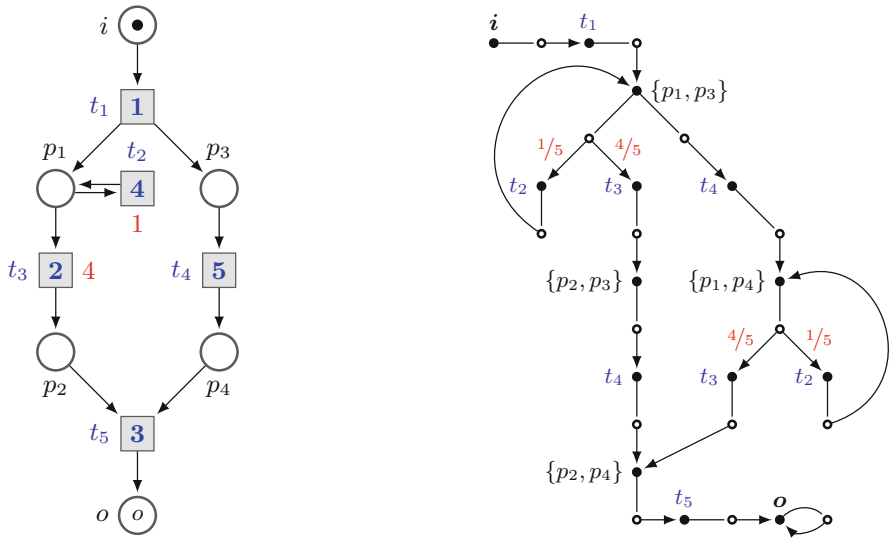
Markov Decision Process semantics. A *Markov Decision Process* (MDP) is a tuple $\mathcal{M} = (Q, q_0, Steps)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $Steps \colon Q \to 2^{dist(Q)}$ is the probability transition function. Paths of an MDP, schedulers, and the probability measure of paths compatible with a scheduler are defined as usual (see the Appendix of the full version [19]).

The semantics of a TPWN $\mathcal{W}$ is a Markov Decision Process $MDP_\mathcal{W}$. The states of $MDP_\mathcal{W}$ are either markings M or pairs (M, t), where t is a transition enabled at M. The intended meanings of M and (M, t) are "the current marking is M", and "the current marking is M, and t has been selected to fire next." Intuitively, t is chosen in two steps: first, a conflict set enabled at M is chosen nondeterministically, and then a transition of this set is chosen at random, with probability proportional to its weight.

Formally, let $\mathcal{W} = (\mathbf{N}, w, \tau)$ be a TPWN where $\mathbf{N} = (P, T, F, i, o)$, let M be a reachable marking of $\mathcal{W}$ enabling at least one transition, and let C be a conflict set of M. Let $w(C)$ be the sum of the weights of the transitions in C. The *probability distribution* $P_{M,C}$ *over* T is given by $P_{M,C}(t) = \frac{w(t)}{w(C)}$ if $t \in C$ and $P_{M,C}(t) = 0$ otherwise. Now, let $\mathcal{M}$ be the set of 1-safe markings of $\mathcal{W}$, and let $\mathcal{E}$ be the set of pairs (M, t) such that $M \in \mathcal{M}$ and M enables t. We define the Markov decision process $MDP_\mathcal{W} = (Q, q_0, Steps)$, where $Q = \mathcal{M} \cup \mathcal{E}$, $q_0 = \mathbf{i}$, the initial marking of $\mathcal{W}$, and $Steps(M)$ is defined for markings of $\mathcal{M}$ and $\mathcal{E}$ as follows. For every $M \in \mathcal{M}$,

- if M enables no transitions, then $Steps(M)$ contains exactly one distribution, which assigns probability 1 to M, and 0 to all other states.
- if M enables at least one transition, then $Steps(M)$ contains a distribution λ for each conflict set C of M. The distribution is defined by: $\lambda(M, t) = P_{M,C}(t)$ for every $t \in C$, and $\lambda(s) = 0$ for every other state s.

For every $(M, t) \in \mathcal{E}$, $Steps(M, t)$ contains one single distribution that assigns probability 1 to the marking M' such that $M \xrightarrow{t} M'$, and probability 0 to every other state.



(a) TPWN $\mathcal{W}$ (b) $MDP_{\mathcal{W}}$

Fig. 1. A TPWN and its associated MDP. (Color figure online)

Example 2. Figure 1b shows a graphical representation of the MDP of the TPWN in Fig. 1a. Black nodes represent states, white nodes probability distributions. A black node q has a white successor for each probability distribution in $Steps(q)$. A white node λ has a black successor for each node q such that $\lambda(q) > 0$; the arrow leading to this black successor is labeled with $\lambda(q)$, unless $\lambda(q) = 1$, in which case there is no label. States (M, t) are abbreviated to t.

Schedulers. Given a TPWN $\mathcal{W}$, a scheduler of MDP_W is a function $\gamma : T^* \to 2^T$ assigning to each firing sequence $\boldsymbol{i} \xrightarrow{\sigma} M$ with $\mathcal{C}(M) \neq \emptyset$ a conflict set $\gamma(\sigma) \in \mathcal{C}(M)$. A firing sequence $\boldsymbol{i} \xrightarrow{\sigma} M$ is *compatible* with a scheduler γ if for all partitions $\sigma = \sigma_1 t \sigma_2$ for some transition t, we have $t \in \gamma(\sigma_1)$.

Example 3. In the TPWN of Fig. 1a, after firing t_1 two conflict sets become concurrently enabled: $\{t_2, t_3\}$ and $\{t_4\}$. A scheduler picks one of the two. If the scheduler picks $\{t_2, t_3\}$ then t_2 may occur, and in this case, since firing t_2 does not change the marking, the scheduler chooses again one of $\{t_2, t_3\}$ and $\{t_4\}$. So there are infinitely many possible schedulers, differing only in how many times they pick $\{t_2, t_3\}$ before picking t_4.

Definition 2 ((Expected) Time until a state is reached). *Let π be an infinite path of $MDP_{\mathcal{W}}$, and let M be a reachable marking of $\mathcal{W}$. Observe that M is a state of $MDP_{\mathcal{W}}$. The* time needed to reach M along π, *denoted $tm(M, \pi)$, is defined as follows: If π does not visit M, then $tm(M, \pi) \stackrel{def}{=} \infty$; otherwise, $tm(M, \pi) \stackrel{def}{=} tm(\Sigma(\pi'))$, where $\Sigma(\pi')$ is the transition sequence corresponding to the shortest prefix π' of π ending at M. Given a scheduler S, the expected time until reaching M is defined as*

$$ET^S_{\mathcal{W}}(M) \stackrel{def}{=} \sum_{\pi \in Paths^S} tm(M,\pi) \cdot Prob^S(\pi).$$

and the expected time $ET^S_{\mathcal{W}}$ *is defined as* $ET^S_{\mathcal{W}} \stackrel{def}{=} ET^S_{\mathcal{W}}(\boldsymbol{o})$*, i.e. the expected time until reaching the final marking.*

In [11] we proved a result for Probabilistic Workflow Nets (PWNs) with rewards, showing that the expected reward of a PWN is independent of the scheduler (intuitively, this is the case because in a confusion-free Petri net the scheduler only determines the logical order in which transitions occur, but not which transitions occur). Despite the fact that, contrary to rewards, the execution time of a firing sequence is not the sum of the execution times of its transitions, the proof carries over to the expected time with only minor modifications.

Theorem 1. *Let* $\mathcal{W}$ *be a TPWN.*

(1) There exists a value $ET_{\mathcal{W}}$ *such that for every scheduler* S *of* $\mathcal{W}$*, the expected time* $ET^S_{\mathcal{W}}$ *of* $\mathcal{W}$ *under* S *is equal to* $ET_{\mathcal{W}}$*.*
(2) $ET_{\mathcal{W}}$ *is finite iff* $\mathcal{W}$ *is sound.*

By this theorem, the expected time $ET_{\mathcal{W}}$ can be computed by choosing a suitable scheduler S, and computing $ET^S_{\mathcal{W}}$.

4 Computation of the Expected Time

We show how to compute the expected time of a TPWN. We fix an appropriate scheduler, show that it induces a finite-state Markov chain, define an appropriate reward function for the chain, and prove that the expected time is equal to the expected reward.

4.1 Earliest-First Scheduler

Consider a firing sequence $\boldsymbol{i} \xrightarrow{\sigma} M$. We define the *starting time* of a conflict set $C \in \mathcal{C}(M)$ as the earliest time at which the transitions of C become enabled. This occurs after *all* tokens of ${}^\bullet C$ arrive[3], and so the starting time of C is the maximum of $\mu(\sigma)_p$ for $p \in {}^\bullet C$ (recall that $\mu(\sigma)_p$ is the latest time at which a token arrives at p while firing σ).

Intuitively, the "earliest-first" scheduler always chooses the conflict set with the earliest starting time (if there are multiple such conflict sets, the scheduler chooses any one of them). Formally, recall that a scheduler is a mapping $\gamma\colon T^* \to 2^T$ such that for every firing sequence $\boldsymbol{i} \xrightarrow{\sigma} M$, the set $\gamma(\sigma)$ is a conflict set of M. We define the *earliest-first scheduler* γ by:

$$\gamma(\sigma) \stackrel{\text{def}}{=} \arg\min_{C \in \mathcal{C}(M)} \max_{p \in {}^\bullet C} \mu(\sigma)_p \qquad \text{where } M \text{ is given by } \boldsymbol{i} \xrightarrow{\sigma} M.$$

[3] This is proved in Lemma 7 in the Appendix of the full version [19].

Example 4. Figure 2a shows the Markov chain induced by the "earliest-first" scheduler defined above in the MDP of Fig. 1b. Initially we have a token at $\boldsymbol{i}$ with arrival time 0. After firing t_1, which takes time 1, we obtain tokens in p_1 and p_3 with arrival time 1. In particular, the conflict sets $\{t_2, t_3\}$ and $\{t_4\}$ become enabled at time 1. The scheduler can choose any of them, because they have the same starting time. Assume it chooses $\{t_2, t_3\}$. The Markov chain now branches into two transitions, corresponding to firing t_2 and t_3 with probabilities $1/5$ and $4/5$, respectively. Consider the branch in which t_2 fires. Since t_2 starts at time 1 and takes 4 time units, it removes the token from p_1 at time 1, and adds a new token to p_1 with arrival time 5; the token at p_3 is not affected, and it keeps its arrival time of 1. So we have $\mu(t_1t_2) = \left\{ {}^{p_1}_{5}, {}^{p_3}_{1} \right\}$ (meaning $\mu(t_1t_2)_{p_1} = 5$, $\mu(t_1t_2)_{p_3} = 1$, and $\mu(t_1t_2)_p = \bot$ otherwise). Now the conflict sets $\{t_2, t_3\}$ and $\{t_4\}$ are enabled again, but with a difference: while $\{t_4\}$ has been enabled since time 1, the set $\{t_2, t_3\}$ is now enabled since time $\mu(t_1t_2)_{p_1} = 5$. The scheduler must now choose $\{t_4\}$, leading to the marking that puts tokens on p_1 and p_4 with arrival times $\mu(t_1t_2t_4)_{p_1} = 5$ and $\mu(t_1t_2t_4)_{p_4} = 6$. In the next steps the scheduler always chooses $\{t_2, t_3\}$ until t_5 becomes enabled. The final marking $\boldsymbol{o}$ can be reached after time 9, through $t_1t_3t_4t_5$ with probability $4/5$, or with times $10 + 4k$ for $k \in \mathbb{N}$, through $t_1t_2t_4t_2^kt_3t_5$ with probability $(1/5)^{k+1} \cdot 4/5$ (the times at which the final marking can be reached are written in blue inside the final states).

Theorem 2 below shows that the earliest-first scheduler only needs finite memory, which is not clear from the definition. The construction is similar to those of [6,15,16]. However, our proof crucially depends on TPWNs being confusion-free.

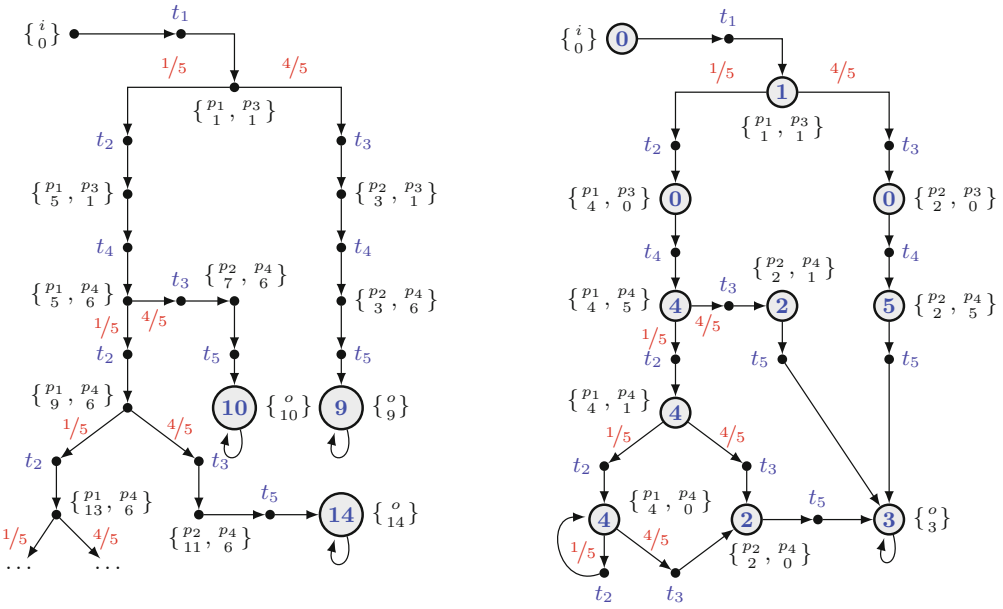
Theorem 2. *Let* $H \stackrel{def}{=} \max_{t \in T} \tau(t)$ *be the maximum duration of the transitions of* T*, and let* $[H]_\bot \stackrel{def}{=} \{\bot, 0, 1, \ldots, H\} \subseteq \mathbb{N}_\bot$*. There are functions* $\nu \colon T^* \to [H]_\bot^P$ *(compare with* $\mu \colon T^* \to \mathbb{N}_\bot^P$*),* $f \colon [H]_\bot^P \times T \to [H]_\bot^P$ *and* $r \colon [H]_\bot^P \to \mathbb{N}$ *such that for every* $\sigma = t_1 \ldots t_n \in T^*$ *compatible with* γ *and for every* $t \in T$ *enabled by* σ*:*

$$\gamma(\sigma) = \operatorname*{arg\,min}_{C \in \mathcal{C}(\llbracket \nu(\sigma) \rrbracket)} \max_{p \in {}^\bullet C} \nu(\sigma)_p \tag{1}$$

$$\nu(\sigma t) = f(\nu(\sigma), t) \tag{2}$$

$$tm(\sigma) = \max_{p \in P} \nu(\sigma)_p + \sum_{k=0}^{n-1} r(\nu(t_1 \ldots t_k)) \tag{3}$$

Observe that, unlike μ, the range of ν is finite. We call it the *finite abstraction* of μ. Equation 1 states that γ can be computed directly from the finite abstraction ν. Equation 2 shows that $\nu(\sigma t)$ can be computed from $\nu(\sigma)$ and t. So γ only needs to remember an element of $[H]_\bot^P$, which implies that it only requires finite memory. Finally, observe that the function r of Eq. 3 has a finite domain, and so it allows us to use ν to compute the time needed by σ.



(a) Infinite MC for scheduler using $\mu(\sigma)$, with final states labeled by $tm(\sigma)$.

(b) Finite MC for scheduler using $\nu(\sigma)$, with states labeled by rewards $r(\nu(\sigma))$.

Fig. 2. Two Markov chains for the "earliest-first" scheduler. (Color figure online)

The formal definition of the functions ν, f, and r is given below, together with the definition of the auxiliary operator $\ominus\colon \mathbb{N}_\perp^P \times \mathbb{N} \to \mathbb{N}_\perp^P$:

$$(\boldsymbol{x} \ominus n)_p \stackrel{\text{def}}{=} \begin{cases} \max(\boldsymbol{x}_p - n, 0) & \text{if } \boldsymbol{x}_p \neq \perp \\ \perp & \text{if } \boldsymbol{x}_p = \perp \end{cases} \qquad f(\boldsymbol{x}, t) \stackrel{\text{def}}{=} upd(\boldsymbol{x}, t) \ominus \max_{p \in {}^\bullet t} \boldsymbol{x}_p$$

$$\nu(\epsilon) \stackrel{\text{def}}{=} \mu(\epsilon) \text{ and } \nu(\sigma t) \stackrel{\text{def}}{=} \mu(\sigma t) \ominus \max_{p \in {}^\bullet t} \mu(\sigma)_p \qquad r(\boldsymbol{x}) \stackrel{\text{def}}{=} \min_{C \in \mathcal{C}(\llbracket \boldsymbol{x} \rrbracket)} \max_{p \in {}^\bullet C} \boldsymbol{x}_p$$

Example 5. Figure 2b shows the finite-state Markov chain induced by the "earliest-first" scheduler computed using the abstraction ν. Consider the firing sequence $t_1 t_3$. We have $\mu(t_1 t_3) = \left\{ \begin{smallmatrix} p_2 \\ 3 \end{smallmatrix}, \begin{smallmatrix} p_3 \\ 1 \end{smallmatrix} \right\}$, i.e. the tokens in p_2 and p_3 arrive at times 3 and 1, respectively. Now we compute $\nu(t_1 t_3)$, which corresponds to the *local* arrival times of the tokens, i.e. the time elapsed *since the last transition starts to fire until the token arrives*. Transition t_3 starts to fire at time 1, and so the local arrival times of the tokens in p_2 and p_3 are 2 and 0, respectively, i.e. we have $\nu(t_1 t_3) = \left\{ \begin{smallmatrix} p_2 \\ 2 \end{smallmatrix}, \begin{smallmatrix} p_3 \\ 0 \end{smallmatrix} \right\}$. Using these local times we compute the local starting time of the conflict sets enabled at $\{p_2, p_3\}$. The scheduler always chooses the conflict set with earliest local starting time. In Fig. 2b the earliest local starting time of the state reached by firing σ, which is denoted $r(\nu(\sigma))$, is written in blue inside the state. The theorem above shows that this scheduler always chooses the same conflict sets as the one which uses the function μ, and that the time of a sequence can be obtained by adding the local starting times. This allows us to consider the earliest local starting time of a state as a *reward*

associated to the state; then, the time taken by a sequence is equal to the sum of the rewards along the corresponding path of the chain. For example, we have $tm(t_1t_2t_4t_3t_5) = 0 + 1 + 0 + 4 + 2 + 3 = 10$.

Finally, let us see how $\nu(\sigma t)$ is computed from $\nu(\sigma)$ for $\sigma = t_1t_2t_4$ and $t = t_2$. We have $\nu(\sigma) = \left\{ \begin{smallmatrix} p_1 \\ 4 \end{smallmatrix}, \begin{smallmatrix} p_4 \\ 5 \end{smallmatrix} \right\}$, i.e. the local arrival times for the tokens in p_1 and p_4 are 4 and 5, respectively. Now $\{t_2, t_3\}$ is scheduled next, with local starting time $r(\nu(\sigma)) = \nu(\sigma)_{p_1} = 4$. If t_2 fires, then, since $\tau(t_2) = 4$, we first add 4 to the time of p_1, obtaining $\left\{ \begin{smallmatrix} p_1 \\ 8 \end{smallmatrix}, \begin{smallmatrix} p_4 \\ 5 \end{smallmatrix} \right\}$. Second, we subtract 4 from *all* times, to obtain the time elapsed since t_2 started to fire (for local times the origin of time changes every time a transition fires), yielding the final result $\nu(\sigma t_2) = \left\{ \begin{smallmatrix} p_1 \\ 4 \end{smallmatrix}, \begin{smallmatrix} p_4 \\ 1 \end{smallmatrix} \right\}$.

4.2 Computation in the Probabilistic Case

Given a TPWN and its corresponding MDP, in the previous section we have defined a finite-state earliest-first scheduler and a reward function of its induced Markov chain. The reward function has the following property: the execution time of a firing sequence compatible with the scheduler is equal to the sum of the rewards of the states visited along it. From the theory of Markov chains with rewards, it follows that the expected accumulated reward until reaching a certain state, provided that this state is reached with probability 1, can be computed by solving a linear equation system. We use this result to compute the expected time ET_W.

Let $\mathcal{W}$ be a sound TPWN. For every firing sequence σ compatible with the earliest-first scheduler γ, the finite-state Markov chain induced by γ contains a state $\boldsymbol{x} = \nu(\sigma) \in [H]^P_\perp$. Let $C_{\boldsymbol{x}}$ be the conflict set scheduled by γ at $\boldsymbol{x}$. We define a system of linear equations with variables $X_{\boldsymbol{x}}$, one for each state $\boldsymbol{x}$:

$$\begin{aligned} X_{\boldsymbol{x}} &= r(\boldsymbol{x}) + \sum_{t \in C_{\boldsymbol{x}}} \frac{w(t)}{w(C_{\boldsymbol{x}})} \cdot X_{f(\boldsymbol{x},t)} && \text{if } [\![\boldsymbol{x}]\!] \neq \boldsymbol{o} \\ X_{\boldsymbol{x}} &= \max_{p \in P} \boldsymbol{x}_p && \text{if } [\![\boldsymbol{x}]\!] = \boldsymbol{o} \end{aligned} \tag{4}$$

The solution of the system is the expected reward of a path leading from $\boldsymbol{i}$ to $\boldsymbol{o}$. By the theory of Markov chains with rewards/costs ([4], Chap. 10.5), we have:

Lemma 2. *Let $\mathcal{W}$ be a sound TPWN. Then the system of linear equations (4) has a unique solution $\boldsymbol{X}$, and $ET_{\mathcal{W}} = \boldsymbol{X}_{\nu(\epsilon)}$.*

Theorem 3. *Let $\mathcal{W}$ be a TPWN. Then $ET_{\mathcal{W}}$ is either ∞ or a rational number and can be computed in single exponential time.*

Proof. We assume that the input has size n and all times and weights are given in binary notation. Testing whether $\mathcal{W}$ is sound can be done by exploration of the state space of reachable markings in time $\mathcal{O}(2^n)$. If $\mathcal{W}$ is unsound, we have $ET_{\mathcal{W}} = \infty$.

Now assume that $\mathcal{W}$ is sound. By Lemma 2, $ET_{\mathcal{W}}$ is the solution to the linear equation system (4), which is finite and has rational coefficients, so it is a

rational number. The number of variables $|\boldsymbol{X}|$ of (4) is bounded by the size of $[H]_{\perp}^{P}$, and as $H = \max_{t \in T} \tau(t)$ we have $|\boldsymbol{X}| \leq (1+H)^{|P|} \leq (1+2^n)^n \leq 2^{n^2+n}$. The linear equation system can be solved in time $\mathcal{O}\left(n^2 \cdot |\boldsymbol{X}|^3\right)$ and therefore in time $\mathcal{O}(2^{p(n)})$ for some polynomial p.

5 Lower Bounds for the Expected Time

We analyze the complexity of computing the expected time of a TPWN. Botezano *et al.* show in [5] that deciding if the expected time exceeds a given bound is NP-hard. However, their reduction produces TPWNs with weights and times of arbitrary size. An open question is if the expected time can be computed in polynomial time when the times (and weights) must be taken from a finite set. We prove that this is not the case unless P = NP, even if all times are 0 or 1, all weights are 1, the workflow net is sound, acyclic and free-choice, and the size of each conflict set is at most 2 (resulting only in probabilities 1 or $1/2$). Further, we show that even computing an ϵ-approximation is equally hard. These two results above are a consequence of the main theorem of this section: computing the expected time is #P-hard [23]. For example, counting the number of satisfying assignments for a boolean formula (#SAT) is a #P-complete problem. Therefore a polynomial-time algorithm for a #P-hard problem would imply P = NP.

The problem used for the reduction is defined on PERT networks [9], in the specialized form of *two-state stochastic PERT networks* [17], described below.

Definition 3. *A* two-state stochastic PERT network *is a tuple* $\mathbf{PN} = (G, s, t, \boldsymbol{p})$*, where* $G = (V, E)$ *is a directed acyclic graph with vertices* V*, representing events, and edges* E*, representing tasks, with a single source vertex* s *and sink vertex* t*, and where the vector* $\boldsymbol{p} \in \mathbb{Q}^E$ *assigns to each edge* $e \in E$ *a rational probability* $p_e \in [0,1]$*. We assume that all* p_e *are written in binary.*

Each edge $e \in E$ *of* **PN** *defines a random variable* X_e *with distribution* $\Pr(X_e = 1) = p_e$ *and* $\Pr(X_e = 0) = 1 - p_e$*. All* X_e *are assumed to be independent. The* project duration PD *of* **PN** *is the length of the longest path in the network*

$$PD(\mathbf{PN}) \stackrel{def}{=} \max_{\pi \in \Pi} \sum_{e \in \pi} X_e$$

where Π *is the set of paths from vertex* s *to vertex* t*. As this defines a random variable, the* expected project duration *of* **PN** *is then given by* $\mathbb{E}(PD(\mathbf{PN}))$*.*

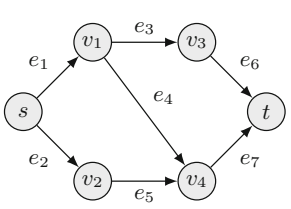
Example 6. Figure 3a shows a small PERT network (without $\boldsymbol{p}$), where the project duration depends on the paths $\Pi = \{e_1e_3e_6, e_1e_4e_7, e_2e_5e_7\}$.

The following problem is #P-hard (from [17], using the results from [20]):

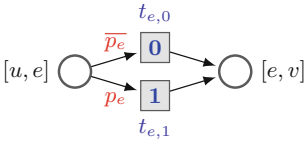
Given: A two-state stochastic PERT network **PN**.
Compute: The expected project duration $\mathbb{E}(PD(\mathbf{PN}))$.

First reduction: 0/1 times, arbitrary weights. We reduce the problem above to computing the expected time of an acyclic TPWN with 0/1 times but arbitrary weights. Given a two-state stochastic PERT network **PN**, we construct a timed probabilistic workflow net $\mathcal{W}_{\mathbf{PN}}$ as follows:

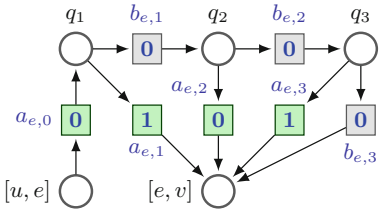
- For each edge $e = (u, v) \in E$, add the "gadget net" shown in Fig. 3b. Assign $w(t_{e,0}) = 1 - p_e$, $w(t_{e,1}) = p_e$, $\tau(t_{e,0}) = 0$, and $\tau(t_{e,1}) = 1$.
- For each vertex $v \in V$, add a transition t_v with arcs from each $[e, v]$ such that $e = (u, v) \in E$ for some u and arcs to each $[v, e]$ such that $e = (v, w) \in E$ for some w. Assign $w(t_v) = 1$ and $\tau(t_v) = 0$.
- Add the place i with an arc to t_s and the place o with an arc from t_t.



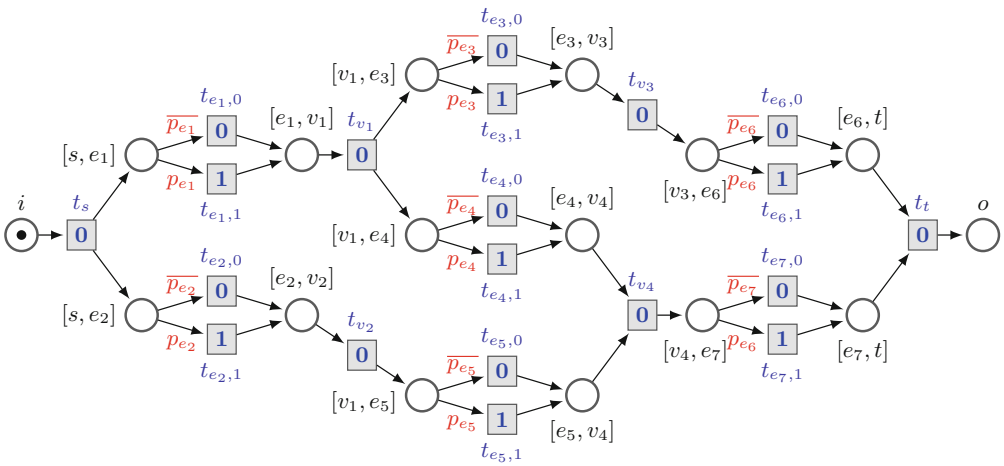
(a) PERT network **PN**.



(b) Gadget for $e = (u, v)$ with rational weights $p_e, \overline{p_e}$.



(c) Equivalent gadget for e with weights 1 for $p_e = 5/8 = (0.101)_2$.



(d) Timed probabilistic workflow net $\mathcal{W}_{\mathbf{PN}}$.

Fig. 3. A PERT network and its corresponding timed probabilistic workflow net. The weight $\overline{p}$ is short for $1 - p$. Transitions without annotations have weight 1.

The result of applying this construction to the PERT network from Fig. 3a is shown in Fig. 3d. It is easy to see that this workflow net is sound, as from any reachable marking, we can fire enabled transitions corresponding to the edges and vertices of the PERT network in the topological order of the graph, eventually firing t_t and reaching $\boldsymbol{o}$. The net is also acyclic and free-choice.

Lemma 3. *Let* **PN** *be a two-state stochastic PERT network and let* $\mathcal{W}_{\mathbf{PN}}$ *be its corresponding TPWN by the construction above. Then* $ET_{\mathcal{W}_{\mathbf{PN}}} = \mathbb{E}(PD(\mathbf{PN}))$.

Second reduction: 0/1 times, 0/1 weights. The network constructed this way already uses times 0 and 1, however the weights still use arbitrary rational numbers. We now replace the gadget nets from Fig. 3b by equivalent nets where all transitions have weight 1. The idea is to use the binary encoding of the probabilities p_e, deciding if the time is 0 or 1 by a sequence of coin flips. We assume that $p_e = \sum_{i=0}^{k} 2^{-i} p_i$ for some $k \in \mathbb{N}$ and $p_i \in \{0, 1\}$ for $0 \leq i \leq k$. The replacement is shown in Fig. 3c for $p_e = 5/8 = (0.101)_2$.

Approximating the expected time is #P-hard. We show that computing an ϵ-approximation for $ET_{\mathcal{W}}$ is #P-hard [17,20].

Theorem 4. *The following problem is* #P*-hard:*

> **Given:** *A sound, acyclic and free-choice TPWN* $\mathcal{W}$ *where all transitions* t *satisfy* $w(t) = 1$, $\tau(t) \in \{0, 1\}$ *and* $|({}^\bullet t)^\bullet| \leq 2$, *and an* $\epsilon > 0$.
> **Compute:** *A rational* r *such that* $r - \epsilon < ET_W < r + \epsilon$.

6 Experimental Evaluation

We have implemented our algorithm to compute the expected time of a TPWN as a package of the tool `ProM`[4]. It is available via the package manager of the latest nightly build under the package name `WorkflowNetAnalyzer`.

We evaluated the algorithm on two different benchmarks. All experiments in this section were run on the same machine equipped with an Intel Core i7-6700K CPU and 32 GB of RAM. We measure the actual runtime of the algorithm, split into construction of the Markov chain and solving the linear equation system, and exclude the time overhead due to starting `ProM` and loading the plugin.

6.1 IBM Benchmark

We evaluated the tool on a set of 1386 workflow nets extracted from a collection of five libraries of industrial business processes modeled in the IBM WebSphere Business Modeler [12]. All of the 1386 nets in the benchmark libraries are free-choice and therefore confusion-free. We selected the sound and 1-safe nets among them, which are 642 nets. Out of these, 409 are marked graphs, i.e. the size of any conflict set is 1. Out of the remaining 233 nets, 193 are acyclic and 40 cyclic.

As these nets do not come with probabilistic or time information, we annotated transitions with integer weights and times chosen uniformly from different intervals: (1) $w(t) = \tau(t) = 1$, (2) $w(t), \tau(t) \in [1, 10^3]$ and (3) $w(t), \tau(t) \in [1, 10^6]$. For each interval, we annotated the transitions of each net with random weights and times, and computed the expected time of all 642 nets.

For all intervals, we computed the expected time for any net in less than 50 ms. The analysis time did not differ much for different intervals. The solving time for the linear equation system is on average 5% of the total analysis time,

[4] http://www.promtools.org/.

and at most 68%. The results for the nets with the longest analysis times are given in Table 1. They show that even for nets with a huge state space, thanks to the earliest-first scheduler, only a small number of reachable markings is explored.

Table 1. Analysis times and size of the state space $|\boldsymbol{X}|$ for the 4 nets with the highest analysis times, given for each of the three intervals $[1], [10^3], [10^6]$ of possible times. Here, $|\mathcal{R}^N|$ denotes the number of reachable markings of the net.

Net	Net info & size				Analysis time (ms)			$\lvert\boldsymbol{X}\rvert$		
	cyclic	$\lvert P\rvert$	$\lvert T\rvert$	$\lvert\mathcal{R}^N\rvert$	$[1]$	$[10^3]$	$[10^6]$	$[1]$	$[10^3]$	$[10^6]$
m1.s30_s703	no	264	286	6117	40.3	44.6	43.8	304	347	347
m1.s30_s596	yes	214	230	623	21.6	24.4	23.6	208	232	234
b3.s371_s1986	no	235	101	$2 \cdot 10^{17}$	16.8	16.4	16.5	101	102	102
b2.s275_s2417	no	103	68	237626	14.2	17.8	15.9	355	460	431

6.2 Process Mining Case Study

As a second benchmark, we evaluated the algorithm on a model of a loan application process. We used the data from the BPI Challenge 2017 [8], an event log containing 31509 cases provided by a financial institute, and took as a model of the process the final net from the report of the winner of the academic category [21], a simple model with high fitness and precision w.r.t. the event log.

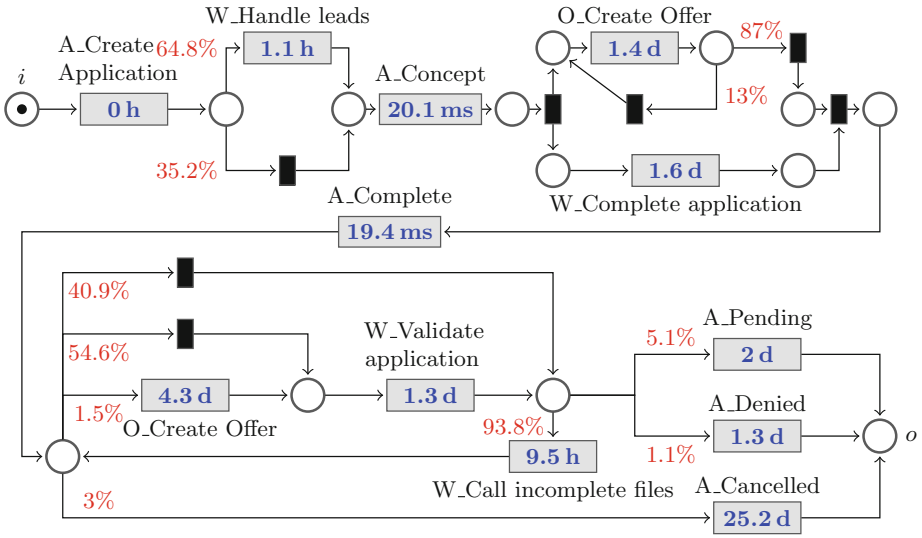


Fig. 4. Net from [21] of process for personal loan applications in a financial institute, annotated with mean waiting times and local trace weights. Black transitions are invisible transitions not appearing in the event log with time 0.

Table 2. Expected time, analysis time and state space size for the net in Fig. 4 for various distributions, where `memout` denotes reaching the memory limit.

Distribution	$\lvert T\rvert$	ET_W	$\lvert \boldsymbol{X}\rvert$	Analysis time		
				Total	Construction	Solving
Deterministic	19	24 d 1 h	33	40 ms	18 ms	22 ms
Histogram/12 h	141	24 d 18 h	4054	244 ms	232 ms	12 ms
Histogram/6 h	261	24 d 21 h	15522	2.1 s	1.8 s	0.3 s
Histogram/4 h	375	24 d 22 h	34063	10 s	6 s	4 s
Histogram/2 h	666	24 d 23 h	122785	346 s	52 s	294 s
Histogram/1 h	1117	—	422614	—	12.7 min	`memout`

Using the `ProM` plugin "Multi-perspective Process Explorer" [18] we annotated each transition with waiting times and each transition in a conflict set with a local percentage of traces choosing this transition when this conflict set is enabled. The net with mean times and weights as percentages is displayed in Fig. 4.

For a first analysis, we simply set the execution time of each transition deterministically to its mean waiting time. However, note that the two transitions "O_Create Offer" and "W_Complete application" are executed in parallel, and therefore the distribution of their execution times influences the total expected time. Therefore we also annotated these two transitions with a histogram of possible execution times from each case. Then we split them up into multiple transitions by grouping the times into buckets of a given interval size, where each bucket creates a transition with an execution time equal to the beginning of the interval, and a weight equal to the number of cases with a waiting time contained in the interval. The times for these transitions range from 6 ms to 31 days. As bucket sizes we chose $12, 6, 4, 2$ and 1 hour(s). The net always has 14 places and 15 reachable markings, but a varying number of transitions depending on the chosen bucket size. For the net with the mean as the deterministic time and for the nets with histograms for each bucket size, we then analyzed the expected execution time using our algorithm.

The results are given in Table 2. They show that using the complete distribution of times instead of only the mean can lead to much more precise results. When the linear equation system becomes very large, the solver time dominates the construction time of the system. This may be because we chose to use an exact solver for sparse linear equation systems. In the future, this could possibly be improved by using an approximative iterative solver.

7 Conclusion

We have shown that computing the expected time to termination of a probabilistic workflow net in which transition firings have deterministic durations is

#P-hard. This is the case even if the net is free-choice, and both probabilities and times can be written down with a constant number of bits. So, surprisingly, computing the expected time is much harder than computing the expected cost, for which there is a polynomial algorithm [11].

We have also presented an exponential algorithm for computing the expected time based on earliest-first schedulers. Its performance depends crucially on the maximal size of conflict sets that can be concurrently enabled. In the most popular suite of industrial benchmarks this number turns out to be small. So, very satisfactorily, the expected time of any of these benchmarks, some of which have hundreds of transitions, can still be computed in milliseconds.

Acknowledgements. We thank Hagen Völzer for input on the implementation and choice of benchmarks.

References

1. van der Aalst, W.M.P.: The application of Petri nets to workflow management. J. Circ. Syst. Comput. **8**(1), 21–66 (1998). https://doi.org/10.1142/S0218126698000043
2. van der Aalst, W.M.P., et al.: Soundness of workflow nets: classification, decidability, and analysis. Formal Asp. Comput. **23**(3), 333–363 (2011). https://doi.org/10.1007/s00165-010-0161-4
3. van der Aalst, W., van Hee, K.M.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2004)
4. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
5. Botezatu, M., Völzer, H., Thiele, L.: The complexity of deadline analysis for workflow graphs with multiple resources. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 252–268. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_15
6. Carlier, J., Chrétienne, P.: Timed Petri net schedules. In: Rozenberg, G. (ed.) APN 1987. LNCS, vol. 340, pp. 62–84. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-50580-6_24
7. Desel, J., Erwin, T.: Modeling, simulation and analysis of business processes. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 129–141. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_9
8. van Dongen, B.F.: BPI Challenge 2017 (2017). https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
9. Elmaghraby, S.E.: Activity Networks: Project Planning and Control by Network Models. Wiley, Hoboken (1977)
10. Esparza, J., Hoffmann, P.: Reduction rules for colored workflow nets. In: Stevens, P., Wąsowski, A. (eds.) FASE 2016. LNCS, vol. 9633, pp. 342–358. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49665-7_20
11. Esparza, J., Hoffmann, P., Saha, R.: Polynomial analysis algorithms for free choice probabilistic workflow nets. Perform. Eval. **117**, 104–129 (2017). https://doi.org/10.1016/j.peva.2017.09.006

12. Fahland, D., et al.: Instantaneous soundness checking of industrial business process models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 278–293. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_19
13. Favre, C., Fahland, D., Völzer, H.: The relationship between workflow graphs and free-choice workflow nets. Inf. Syst. **47**, 197–219 (2015). https://doi.org/10.1016/j.is.2013.12.004
14. Favre, C., Völzer, H., Müller, P.: Diagnostic information for control-flow analysis of workflow graphs (a.k.a. free-choice workflow nets). In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 463–479. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_27
15. Gaubert, S., Mairesse, J.: Asymptotic analysis of heaps of pieces and application to timed Petri nets. In: Proceedings of the 8th International Workshop on Petri Nets and Performance Models, PNPM 1999, Zaragoza, Spain, 8–10 September 1999, pp. 158–169 (1999). https://doi.org/10.1109/PNPM.1999.796562
16. Gaubert, S., Mairesse, J.: Modeling and analysis of timed Petri nets using heaps of pieces. IEEE Trans. Autom. Control **44**(4), 683–697 (1999). https://doi.org/10.1109/9.754807
17. Hagstrom, J.N.: Computational complexity of PERT problems. Networks **18**(2), 139–147 (1988). https://doi.org/10.1002/net.3230180206
18. Mannhardt, F., de Leoni, M., Reijers, H.A.: The multi-perspective process explorer. In: Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management, BPM 2015, Innsbruck, Austria, 2 September 2015, pp. 130–134 (2015)
19. Meyer, P.J., Esparza, J., Offtermatt, P.: Computing the expected execution time of probabilistic workflow nets. arXiv:1811.06961 [cs.LO] (2018)
20. Provan, J.S., Ball, M.O.: The complexity of counting cuts and of computing the probability that a graph is connected. SIAM J. Comput. **12**(4), 777–788 (1983). https://doi.org/10.1137/0212053
21. Rodrigues, A., et al.: Stairway to value: mining a loan application process (2017). https://www.win.tue.nl/bpi/lib/exe/fetch.php?media=2017:bpi2017_winner_academic.pdf
22. Rozenberg, G., Thiagarajan, P.S.: Petri nets: basic notions, structure, behaviour. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) Current Trends in Concurrency. LNCS, vol. 224, pp. 585–668. Springer, Heidelberg (1986). https://doi.org/10.1007/BFb0027048
23. Valiant, L.G.: The complexity of computing the permanent. Theoret. Comput. Sci. **8**, 189–201 (1979). https://doi.org/10.1016/0304-3975(79)90044-6
24. Meyer, P.J., Esparza, J., Offtermatt, P.: Artifact and instructions to generate experimental results for TACAS 2019 paper: Computing the Expected Execution Time of Probabilistic Workflow Nets (artifact). Figshare (2019). https://doi.org/10.6084/m9.figshare.7831781.v1

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Appendix II

Non-first Author Publications

Paper D

An SMT-based Approach to Coverability Analysis. CAV, 2014.

This section has been published as a **peer-reviewed conference paper**. The thesis author is **not** first author of the paper.

© Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp Meyer and Filip Niksic.

J. Esparza, R. Ledesma-Garza, R. Majumdar, P. Meyer, and F. Niksic. "An SMT-Based Approach to Coverability Analysis". In: *Proceedings of the 26th International Conference on Computer Aided Verification, CAV 2014*. Ed. by A. Biere and R. Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 603–619. DOI: 10.1007/978-3-319-08867-9_40

Summary

We revisit the constraint approach of [EM00] for verifying safety properties of Petri nets using the marking equation and an iterative refinement with traps. We give a procedure employing an SMT solver implementing this constraint approach. Further, we show how we can generate an inductive invariant as a proof of safety from the set of constructed constraints. We experimentally evaluate our approach on a large set of benchmarks and compare it to existing tools. The results show that using our approach, one can prove safety of most of the safe instances within a few seconds. Especially for large Petri nets we are faster than other tools, and the invariants generated by our tool are often orders of magnitude smaller than those constructed by other tools.

Contributions of thesis author

Composition and revision of the manuscript. Joint discussion and development of the theoretical results presented in the paper, with the following notable individual contributions: development of the constraints to construct and minimize invariants; implementation of the verification method; preliminary experimental evaluation.

An SMT-Based Approach to Coverability Analysis

Javier Esparza[1], Ruslán Ledesma-Garza[1], Rupak Majumdar[2], Philipp Meyer[1], and Filip Niksic[2]

[1] Technische Universität München
[2] Max Planck Institute for Software Systems (MPI-SWS)

Abstract. Model checkers based on Petri net coverability have been used successfully in recent years to verify safety properties of concurrent shared-memory or asynchronous message-passing software. We revisit a constraint approach to coverability based on classical Petri net analysis techniques. We show how to utilize an SMT solver to implement the constraint approach, and additionally, to generate an inductive invariant from a safety proof. We empirically evaluate our procedure on a large set of existing Petri net benchmarks. Even though our technique is incomplete, it can quickly discharge most of the safe instances. Additionally, the inductive invariants computed are usually orders of magnitude smaller than those produced by existing solvers.

1 Introduction

In recent years many papers have proposed and developed techniques for the verification of concurrent software [10,6,1,11,4]. In particular, model checkers based on Petri net coverability have been successfully applied. Petri nets are a simple and natural automata-like model for concurrent systems, and can model certain programs with an unbounded number of threads or thread creation. In a nutshell, the places of the net correspond to program locations, and the number of tokens in a place models the number of threads that are currently at that location. This point was first observed in [9], and later revisited in [3] and, more implicitly, in [10,6].

The problem whether at least one thread can reach a given program location (modelling some kind of error), naturally reduces to the *coverability problem* for Petri nets: given a net N and a marking M, decide whether some reachable marking of N *covers* M, i.e., puts at least as many tokens as M on each place. While the decidability and EXPSPACE-completeness of the coverability problem were settled long ago [12,17], new algorithmic ideas have been developed in recent years [8,7,21,11,13]. The techniques are based on forward or backward state-space exploration, which is accelerated in a number of ways in order to cope with the possibly infinite number of states.

In this paper we revisit an approach to the coverability problem based on classical Petri net analysis techniques: the marking equation and traps [16,18]. The marking equation is a system of linear constraints that can be easily derived

A. Biere and R. Bloem (Eds.): CAV 2014, LNCS 8559, pp. 603–619, 2014.
© Springer International Publishing Switzerland 2014

from the net, and whose set of solutions overapproximates the set of reachable markings. This system can be supplemented with linear constraints specifying a set of unsafe markings, and solved using standard linear or integer programming. If the constraints are infeasible, then all reachable markings are safe. If not, then one can try different aproaches. In [5] a solution of the constraints is used to derive an additional constraint in the shape of a *trap*: a set of places that, loosely speaking, once marked cannot be "emptied"; the process can be iterated. More recently, in [22], Wimmel and Wolf propose to use the solution to guide a state space exploration searching for an unsafe marking; if the search fails, then information gathered during it is used to construct an additional constraint.

Constraint-based techniques, while known for a while, have always suffered from the absence of efficient decision procedures for linear arithmetic together with Boolean satisfiability. Profiting from recent advances in SMT-solving technology, we reimplement the technique of [5] on top of the Z3 SMT solver [2], and apply it to a large collection of benchmarks.

The technique is theoretically incomplete, i.e., the set of linear constraints derived from the marking equation and traps may be feasible even if all reachable markings are safe. Our first and surprising finding is that, despite this fact, the technique is powerful enough to prove safety of 96 out of a total of 115 safe benchmarks gathered from current research papers in concurrent software verification. In contrast, three different state-of-the-art tools for coverability proved only 61, 51, or 33 of these 115 cases! Moreover, and possibly due to the characteristics of the application domain, even the simplest version of the technique —based on the marking equation— is successful in 84 cases.

As a second contribution, and inspired by work on interpolation, we show that a dual version of the classical set of constraints, equivalent in expressive power, can be used not only to check safety, but to produce an inductive invariant. While some existing solvers based on state-space exploration can also produce such invariants, we show that inductive invariants obtained through our technique are usually orders of magnitude smaller. Additionally, while we can use the SMT solver iteratively to minimize the invariant, the tool almost always provides a minimal one at the first attempt.

Related Work. Our starting point was the work of Esparza and Melzer on extending the marking equation with trap conditions to gain a stronger method for proving safety of Petri nets [5]. We combined the constraint-based approach there with modern SMT solvers. Their focus on (integer) linear programming tools of the time enforced some limitations. First, while traps are naturally encoded using Boolean variables, [5] encoded traps and the marking equation together into a set of linear constraints. This encoding came at a practical cost: the encoding required (roughly) $n \times m$ constraints for a Petri net with n places and m transitions, whereas the natural Boolean encoding requires m constraints. Moreover, (I)LP solvers were not effective in searching large Boolean state spaces; our use of modern SAT techniques alleviates this problem. Second, (I)LP solvers used by [5] did not handle strict inequalities. Hence, the authors used additional tricks, such as posing the problem that includes a strict inequality as a minimization

```
     procedure Process 1                    procedure Process 2
     begin                                  begin
       bit1 := false                          bit2 := false
       while true do                          while true do
p1:      bit1 := true                   q1:      bit2 := true
p2:      while bit2 do skip od          q2:      if bit1 then
p3:      (* critical section *)          q3:         bit2 := false
         bit1 := false                  q4:         while bit1 do skip od
       od                                          goto q1
     end                                       fi
                                        q5:      (* critical section *)
                                                 bit2 := false
                                               od
                                             end
```

Fig. 1. Lamport's 1-bit algorithm for mutual exclusion [14]

problem, with the goal of minimizing the involved expression, and testing if the minimal value equaled zero. Unfortunately, this trick led to numerical instabilities. All of these concerns vanish by using an SMT solver.

The marking equation is also the starting point of [22], but the strategies of this approach and ours are orthogonal: while we use the solutions of the marking equation to derive new constraints, [22] uses them to guide state space explorations that search for unsafe markings; new constraints are generated only if the searches fail.

In contrast to other recent techniques for coverability [7,11,13], our technique and the one of [22] are incomplete. However, in [22] Wimmel and Wolf obtain very good results for business process benchmarks, and in this paper we empirically demonstrate that our technique is effective for safe software verification benchmarks, often beating well-optimized state exploration approaches.

Our technique theoretically applies not only to coverability but also to *reachability*. It will be interesting to see whether the techniques can effectively verify reachability questions, e.g., arising from liveness verification [6].

2 Preliminaries

A *Petri net* is a tuple (P, T, F, m_0), where P is a set of *places*, T is a (disjoint) set of *transitions*, $F : (P \times T) \cup (T \times P) \to \{0, 1\}$ is the *flow function*, and $m_0 : P \to \mathbb{N}$ is the initial marking. For $x \in P \cup T$, the *pre-set* is ${}^\bullet x = \{y \in P \cup T \mid F(y, x) = 1\}$ and the *post-set* is $x^\bullet = \{y \in P \cup T \mid F(x, y) = 1\}$. We extend the pre- and post-set to a subset of $P \cup T$ as the union of the pre- and post-sets of its elements.

A *marking* of a Petri net is a function $m \colon P \to \mathbb{N}$, which describes the number of tokens $m(p)$ in each place $p \in P$. Assuming an enumeration $p_1, \ldots, p_n$ of P, we often identify m and the vector $(m(p_1), \ldots, m(p_n))$. For a subset $P' \subseteq P$ of places, we write $m(P') = \sum_{p \in P'} m(p)$.

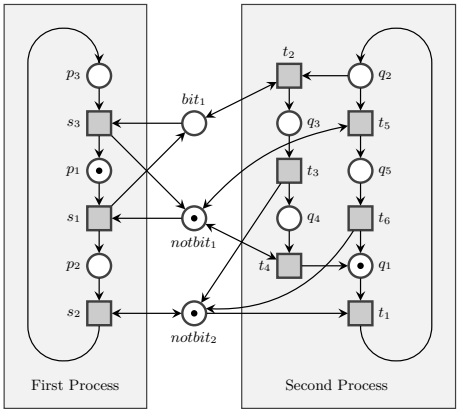


Fig. 2. Petri net for Lamport's 1-bit algorithm

A transition $t \in T$ is *enabled at* m iff for all $p \in {}^\bullet t$, we have $m(p) \geq F(p,t)$. A transition t enabled at m may *fire*, yielding a new marking m' (denoted $m \xrightarrow{t} m'$), where $m'(p) = m(p) + F(t,p) - F(p,t)$. A sequence of transitions, $\sigma = t_1 t_2 \dots t_r$ is an *occurrence sequence* of N iff there exist markings $m_1, \dots, m_r$ such that $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \dots \xrightarrow{t_r} m_r$. The marking m_r is said to be *reachable* from m_0 by the occurrence of σ (denoted $m_0 \xrightarrow{\sigma} m_r$).

A *property* φ is a linear arithmetic constraint over the free variables P. The property φ holds on a marking m iff $m \models \varphi$. A Petri net N satisfies a property φ (denoted by $N \models \varphi$) iff for all reachable markings $m_0 \xrightarrow{\sigma} m$, we have $m \models \varphi$. A property φ is an *invariant* of N if it holds for every reachable marking. A property is inductive if whenever $m \models \varphi$ and $m \xrightarrow{t} m'$ for some $t \in T$ and marking m', we have $m' \models \varphi$.

Petri nets are represented graphically as follows: places and transitions are represented as circles and boxes, respectively. For $x, y \in P \cup T$, there is an arc leading from x to y iff $F(x,y) = 1$. As an example, consider Lamport's 1-bit algorithm for mutual exclusion [14], shown in Fig. 1. Fig. 2 shows a Petri net model for the code. The two grey blocks model the control flow of the two processes. For instance, the token in place p_1 models the current position of process 1 at program location p_1. The three places in the middle of the diagram model the current values of the variables. For instance, a token in place $notbit_1$ indicates that the variable bit_1 is currently set to *false*. The mutual exclusion property, which states that the two processes cannot be in the critical section at the same time, corresponds to the property that places p_3 and q_5 cannot both have a token at the same time.

3 Marking Equation

We now recall a well-known method, which we call SAFETY, that provides a sufficient condition for a given Petri net N to satisfy a property φ by reducing

the problem to checking satisfiability of a linear arithmetic formula. We illustrate the method on Lamport's 1-bit algorithm for mutual exclusion.

Before going into details, we state several conventions. For a Petri net $N = (P, T, F, m_0)$, we introduce a vector of $|P|$ variables M, and a vector of $|T|$ variables X. The vectors M and X will be used to represent the current marking and the number of occurrences of transitions in the occurrence sequence leading to the current marking, respectively. If a place or a transition is given a specific name, we use the same name for its associated variable. Given a place p, the intended meaning of a constraint like $p \geq 3$ is "at the current marking place p must have at least 3 tokens." Given a transition t, the intended meaning of a constraint like $t \leq 2$ is "in the occurrence sequence leading to the current marking, transition t must fire at most twice."

The key idea of the SAFETY method lies in the *marking equation*:

$$M = m_0 + CX\,,$$

where the incidence matrix C is a $|P| \times |T|$ matrix given by

$$C(p,t) = F(t,p) - F(p,t)\,.$$

For each place p, the marking equation contains a constraint that formulates a simple token conservation law: the number of tokens in p at the current marking is equal to the initial number of tokens $m_0(p)$, plus the number of tokens added by the input transitions of p, minus the number of tokens removed by the output transitions. So, for instance, in Lamport's algorithm the constraint for place $notbit_1$ is:

$$notbit_1 = 1 + s_3 + t_5 + t_4 - s_1 - t_4 - t_5 = 1 + s_3 - s_1\,.$$

We equip the marking equation with the non-negativity conditions, modeling that the number of tokens in a place, or the number of occurrences of a transition in an occurrence sequence cannot become negative. All together, we get the following set of *marking constraints*:

$$\mathcal{C}(P,T,F,m_0) :: \begin{cases} M = m_0 + CX & \text{marking equation} \\ M \geq 0 & \text{non-negativity conditions for places} \\ X \geq 0 & \text{non-negativity conditions for transitions} \end{cases}$$

Method SAFETY for checking that a property φ is invariant for a Petri net $N = (P, T, F, m_0)$ consists of checking for satisfiability of the constraints

$$\mathcal{C}(P,T,F,m_0) \wedge \neg\varphi(M)\,. \tag{1}$$

If the constraints are unsatisfiable, then no reachable marking violates φ. To see that this is true, consider the converse: If there exists an occurrence sequence $m_0 \xrightarrow{\sigma} m$ leading to a marking m that violates the property, then we can construct a valuation of the variables that assigns $m(p)$ to $M(p)$ for each place

p, and the number of occurrences of t in σ to $X(t)$ for each transition t. This valuation then satisfies the constraints.

The method does not work in the other direction: If the constraints (1) are satisfiable, we cannot conclude that the property φ is violated.

As an example, consider the Lamport's algorithm. SAFETY successfully proves the property "if process 1 is at location p_3, then $bit_1 = true$" by showing that $\mathcal{C}(P,T,F,m_0) \wedge p_3 \geq 1 \wedge bit_1 \neq 1$ is unsatisfiable. However, if we apply it to the mutual exclusion property, i.e., check for satisfiability of $\mathcal{C}(P,T,F,m_0) \wedge p_3 \geq 1 \wedge q_5 \geq 1$, we obtain a solution, but we cannot conclude that the mutual exclusion property does not hold.

Note that the marking constraints (1) are interpreted over integer variables. As usual in program analysis, one can solve the constraints over rationals to get an approximation of the method. Solving the constraints over rationals will become useful in Section 5.

4 Refining Marking Equations with Traps

Esparza and Melzer [5] strengthened SAFETY with additional *trap constraints*. A *trap* of a Petri net $N = (P,T,F,m_0)$ is a subset of places $Q \subseteq P$ satisfying the following condition for every transition $t \in T$: if t is an output transition of at least one place of Q, then it is also an input transition of at least one place of Q. Equivalently, Q is a trap if its set of output transitions is included in its set of input transitions, i.e., if $Q^\bullet \subseteq {}^\bullet Q$. Here we present a variant of Esparza's and Melzer's method that encodes traps using Boolean constraints. We call the new method SAFETYBYREFINEMENT.

The method SAFETYBYREFINEMENT is based on the following observation about traps. If Q is a trap and a marking m marks Q, i.e., $m(p) > 0$ for some $p \in Q$, then for each occurrence sequence σ and marking m' such that $m \xrightarrow{\sigma} m'$, we also have $m'(p') > 0$ for some $p' \in Q$. Indeed, by the trap property any transition removing tokens from places of Q also adds at least one token to some place of Q. So, while $m'(Q)$ can be smaller than $m(Q)$, it can never become 0. In particular, if a trap Q satisfies $m_0(Q) > 0$, then every reachable marking m satisfies $m(Q) > 0$ as well.

Since the above property must hold for any trap, we can restrict the constraints from method SAFETY as follows. First, we add an additional vector B of $|P|$ Boolean variables. These variables are used to encode traps: for $p \in P$, $B(p)$ is true if and only if place p is part of the trap. The following constraint specifies that B encodes a trap:

$$trap(B) ::= \bigwedge_{t \in T} \left[\bigvee_{p \in {}^\bullet t} B(p) \implies \bigvee_{p \in t^\bullet} B(p) \right] .$$

Next, we define a predicate $mark(m,B)$ that specifies marking m marks a trap:

$$mark(m,B) ::= \bigvee_{p \in P} B(p) \wedge (m(p) > 0) .$$

Finally, we conjoin the following constraint to the constraints (1):

$$\forall B : \mathit{trap}(B) \wedge \mathit{mark}(m_0, B) \implies \mathit{mark}(M, B)\,. \tag{2}$$

This constraint conceptually enumerates over all subsets of places, and ensures that if the subset forms a trap, and this trap is marked by the initial marking, then it is also marked by the current marking. Thus, markings violating the trap constraint are eliminated.

While the above constraint provides a refinement of the Safety method, it requires the SMT solver to reason with universally quantified variables. Instead of directly using universal quantifiers, we use a counterexample-guided heuristic [5,20] of adding trap constraints one-at-a-time in the following way.

If the set of constraints constructed so far (for instance, the set given by the method Safety) is feasible, the SMT solver delivers a model that assigns values to each place, corresponding to a potentially reachable marking m. We search for a trap P_m that violates the trap condition (2) for this specific model m. If we find such a trap, then we know that m is unreachable, and we can add the constraint $\sum_{p \in P_m} M(p) \geq 1$ to exclude all markings that violate this specific trap condition.

The search for P_m is a pure Boolean satisfiability question. We ask for an assignment to

$$\mathit{trap}(B) \wedge \mathit{mark}(m_0, B) \wedge \neg \mathit{mark}(m, B) \tag{3}$$

Notice that for a fixed marking m, the predicate $\mathit{mark}(m, B)$ simplifies to a Boolean predicate. Given a satisfying assignment b for this formula, we add the constraint

$$\sum_{\substack{p \in P \\ b(p) = \mathit{true}}} M(p) \geq 1 \tag{4}$$

to the current set of constraints to rule out solutions that do not satisfy this trap constraint. We iteratively add such constraints until either the constraints are unsatisfiable or the Boolean constraints (3) are unsatisfiable (i.e., no traps are found to invalidate the current solution).

This yields the method SafetyByRefinement. It is still not complete [5]: one can find nets and unreachable markings that mark all traps of the net.

Let us apply the algorithm SafetyByRefinement to Lamport's algorithm and the mutual exclusion property. Recall that the markings violating the property are those satisfying $p_3 \geq 1$ and $q_5 \geq 1$. Safety yields a satisfying assignment with $p_3 = \mathit{bit}_1 = q_5 = 1$, and $p = 0$ for all other places p, which corresponds to a potentially reachable marking m. We search for a trap marked at m_0 but not at m. To simplify the notation, we simply write p instead of $B(p)$. The constraints derived from the trap property are:

$$\begin{array}{lcl} p_1 \vee notbit_1 & \Longrightarrow & p_2 \vee bit_1 \\ p_2 \vee notbit_2 & \Longrightarrow & p_3 \vee notbit_2 \\ p_3 \vee bit_1 & \Longrightarrow & p_1 \vee notbit_1 \end{array} \qquad \begin{array}{lcl} q_1 \vee notbit_2 & \Longrightarrow & q_2 \\ q_2 \vee bit_1 & \Longrightarrow & q_3 \vee bit_1 \\ q_3 & \Longrightarrow & q_4 \vee notbit_2 \\ q_4 \vee notbit_1 & \Longrightarrow & q_1 \vee notbit_1 \\ q_2 \vee notbit_1 & \Longrightarrow & q_5 \vee notbit_1 \\ q_5 & \Longrightarrow & q_1 \vee notbit_2 \end{array}$$

and the following constraints model that at least one of the places initially marked belongs to the trap, but none of the places marked at the satisfying assigment do:

$$p_1 \vee q_1 \vee notbit_1 \vee notbit_2 \qquad \neg p_3 \wedge \neg q_5 \wedge \neg bit_1$$

For this set of constraints we find the satisfying assignment that sets p_2, $notbit_1$, $notbit_2$, q_2, q_3 to *true* and all other variables to *false*. So this set of places is an initially marked trap, and so every reachable marking should put at least one token in it. Hence we can add the refinement constraint to marking constraints (1):

$$p_2 + q_2 + q_3 + notbit_1 + notbit_2 \geq 1\,.$$

On running the SMT solver again, we find the constraints are unsatisfiable, proving that the mutual exclusion property holds.

5 Constructing Invariants from Constraints

We now show that one can compute inductive invariants from the method SafetyByRefinement. That is, given a Petri net $N = (P, T, F, m_0)$ and a property φ, if SafetyByRefinement (over the rationals) can prove N satisfies φ, then in fact we can construct a linear inductive invariant that contains m_0 and does not intersect $\neg\varphi$. We call the new method InvariantByRefinement.

The key observation is to use a constraint system dual to the constraint system for SafetyByRefinement. We assume φ is a co-linear property, i.e., the negation $\neg\varphi$ is represented as the constraints:

$$\neg\varphi \;::\; AM \geq b$$

where A is a $k \times |P|$ matrix, and b is a $k \times 1$ vector, for some $k \geq 1$. Furthermore, we assume that there are $l \geq 0$ trap constraints (4), which are collected in matrix form $DM \geq 1$, for an $l \times |P|$ matrix D, and an $l \times 1$ vector of ones, denoted simply by 1. Consider the following primal system $\mathcal{S}$:

$\mathcal{C}(P, T, F, m_0)$	marking constraints
$AM \geq b$	negation of property φ
$DM \geq 1$	trap constraints

By transforming $\mathcal{S}$ into a suitable form and applying Farkas' Lemma [19], we get the following theorem.

Theorem 1. *The primal system $\mathcal{S}$ is unsatisfiable over the rational numbers if and only if the following dual system $\mathcal{S}'$ is satisfiable over the rational numbers.*

$$\begin{aligned} \lambda C &\leq 0 && \textit{inductivity constraint} \\ \lambda m_0 &< Y_1 b + Y_2 1 && \textit{safety constraint} \\ \lambda &\geq Y_1 A + Y_2 D && \textit{property constraint} \\ Y_1, Y_2 &\geq 0 && \textit{non-negativity constraint} \end{aligned}$$

Here λ, Y_1 and Y_2 are vectors of variables of size $1 \times |P|$, $1 \times k$ and $1 \times l$, respectively.

If the primal system $\mathcal{S}$ is unsatisfiable, we can take λ from a solution to $\mathcal{S}'$ and construct an inductive invariant:

$$I(M) \ ::= \ DM \geq 1 \wedge \lambda M \leq \lambda m_0 \, .$$

In order to show that $I(M)$ is an invariant, recall that for every reachable marking m there is a solution to $m = m_0 + CX$, with $X \geq 0$. Multiplying by λ and taking into account that λ is a solution to $\mathcal{S}'$, we get

$$\lambda m = \lambda m_0 + \lambda C X \leq \lambda m_0 \, .$$

Furthermore, every reachable marking satisfies the trap constraints $DM \geq 1$. On the other hand, a marking m that violates the property φ does not satisfy $I(M)$, for it either does not satisfy $DM \geq 1$, or both $Am \geq b$ and $Dm \geq 1$ hold. But in the latter case we have

$$\lambda m \geq (Y_1 A + Y_2 D)m = Y_1 Am + Y_2 Dm \geq Y_1 b + Y_2 1 > \lambda m_0 \, .$$

In order to show that $I(M)$ is inductive, we have to show that if $I(m)$ holds for some marking m (reachable or not), and $m \xrightarrow{t} m'$ for some transition t, then $I(m')$ holds as well. Indeed, in this case we have $m' = m + Ce_t$, where e_t is the unit vector with 1 in the t-th component and 0 elsewhere. Hence

$$\lambda m' = \lambda(m + Ce_t) = \lambda m + \lambda C e_t \leq \lambda m \leq \lambda m_0 \, ,$$

and furthermore, as m satisfies the trap constraints, m' also satisfies them.

So far, we have assumed that property φ is a co-linear property. However, it is easy to extend the method to the case when $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_r$, and each φ_i is a co-linear property. In that case, for each φ_i we invoke InvariantByRefinement to obtain an inductive invariant I_i. One can easily verify that $I_1 \wedge \ldots \wedge I_r$ is an inductive invariant with respect to φ.

Minimizing invariants. Note that the system $\mathcal{S}'$ from Theorem 1 may in general have many solutions, and each solution yields an inductive invariant. Solutions where λ has fewer non-zero components yield shorter inductive invariants $I(M)$, assuming terms in $I(M)$ with coefficient zero are left out. We can force the

Inductivity constraints

$$\begin{array}{rrrrl} -p_1 + p_2 & & + bit_1 - notbit_1 & \leq 0 \\ & -p_2 + p_3 & & \leq 0 \\ p_1 & & -p_3 - bit_1 + notbit_1 & \leq 0 \end{array} \qquad \begin{array}{rrrrrrl} -q_1 + q_2 & & & & & - notbit_2 & \leq 0 \\ & -q_2 + q_3 & & & & & \leq 0 \\ & & -q_3 + q_4 & & & + notbit_2 & \leq 0 \\ q_1 & & & -q_4 & & & \leq 0 \\ & -q_2 & & & + q_5 & & \leq 0 \\ q_1 & & & & - q_5 & + notbit_2 & \leq 0 \end{array}$$

Safety constraint

$$p_1 + q_1 + notbit_1 + notbit_2 < target_1 + target_2 + trap_1$$

Property constraints

$$\begin{array}{llll} p_1 \geq 0 & q_1 \geq 0 & q_4 \geq 0 & bit_1 \geq 0 \\ p_2 \geq trap_1 & q_2 \geq trap_1 & q_5 \geq target_2 & notbit_1 \geq trap_1 \\ p_3 \geq target_1 & q_3 \geq trap_1 & & notbit_2 \geq trap_1 \end{array}$$

Non-negativity constraints

$$target_1, target_2, trap_1 \geq 0$$

Fig. 3. System of constraints $\mathcal{S}'$ for Lamport's algorithm and the mutual exclusion property. Here, $\lambda = (p_1\ p_2\ p_3\ q_1\ q_2\ q_3\ q_4\ q_5\ bit_1\ notbit_1\ notbit_2)$, $Y_1 = (target_1\ target_2)$ and $Y_2 = (trap_1)$.

number of non-zero components to be at most K by introducing a vector of $|P|$ variables Z, adding for each $p \in P$ constraints

$$\begin{aligned} \lambda(p) > 0 &\implies Z(p) = 1 \\ \lambda(p) = 0 &\implies Z(p) = 0 \end{aligned}$$

and adding a constraint $\sum_{p \in P} Z(p) \leq K$. By varying K, we can find a solution with the smallest number of non-zero components in λ.

Example. Consider again Lamport's algorithm and the mutual exclusion property. Recall that the negation of the property for this example is $p_3 \geq 1 \wedge q_5 \geq 1$, and the trap constraint is $p_2 + q_2 + q_3 + notbit_1 + notbit_2 \geq 1$. Fig. 3 shows the system of constraints $\mathcal{S}'$ for this example. A possible satisfying assignment sets q_1, q_4, and bit_1 to 0, p_2, p_3, and $target_1$ to 2, and all other variables to 1. The corresponding inductive invariant is:

$$\begin{aligned} I(M) \ ::= &\ (p_2 + q_2 + q_3 + notbit_1 + notbit_2 \geq 1) \wedge \\ &\ (p_1 + 2p_2 + 2p_3 + notbit_1 + notbit_2 + q_2 + q_3 + q_5 \leq 3)\,. \end{aligned}$$

If we add constraints that bound the number of non-zero components in λ to 7, the SMT solver finds a new solution, setting p_2, p_3, $notbit_1$, $notbit_2$, q_2, q_3,

$target_1$, $target_2$, and $trap_1$ to 1, and all other variables to 0. The corresponding inductive invariant for this solution is

$$I'(M) \ ::= (p_2 + q_2 + notbit_1 + notbit_2 + q_3 \geq 1) \wedge \\ (p_2 + p_3 + notbit_1 + notbit_2 + q_2 + q_3 + q_5 \leq 2)\,.$$

6 Experimental Evaluation

We implemented our algorithms in a tool called *Petrinizer*. Petrinizer is implemented as a script on top of the Z3 SMT solver [2]. It takes as input coverability problem instances encoded in the MIST input format[1], and it runs one of the selected methods. We implemented all possible combinations of methods: with and without trap refinement, with rational and integer arithmetic, with and without invariant construction, with and without invariant minimization.

Our evaluation had two main goals. First, as the underlying methods are incomplete, we wanted to measure their success rate on standard benchmark sets. As a subgoal, we wanted to investigate the usefulness and necessity of traps, the benefit of using integer arithmetic over rational arithmetic, and the sizes of the constructed invariants. The second goal was to measure Petrinizer's performance and to compare it with state-of-the-art tools: IIC [13], BFC[2] [11], and MIST[3].

Benchmarks. For the inputs used in the experiments, we collected coverability problem instances originating from various sources. The collection contains 178 examples, out of which 115 are safe, and is organized into five example suites. The first suite is a collection of Petri net examples from the MIST toolkit. This suite contains a mixture of 29 examples, both safe and unsafe. It contains both real-world and artificially created examples. The second suite consists of 46 Petri nets that were used in the evaluation of BFC [11]. They originate from the analysis of concurrent C programs, and they are mostly unsafe. The third and the fourth suites come from the provenance analysis of messages in a medical system and a bug-tracking system [15]. The medical suite contains 12 safe examples, and the bug-tracking suite contains 41 examples, all safe except for one. The fifth suite contains 50 examples that come from the analysis of Erlang programs [4]. We generated them ourselves using an Erlang verification tool called Soter [4], from the example programs found on Soter's website[4]. Out of 50 examples in this suite, 38 are safe. This suite also contains the largest example in the collection, with 66,950 places and 213,635 transitions. For our evaluation, only the 115 safe instances are interesting.

[1] `https://github.com/pierreganty/mist`

[2] The most recent version of BFC at the time of writing the paper was 2.0. However, we noticed it sometimes reports inconsistent results, so we used version 1.0 instead. The tool can be obtained at `http://www.cprover.org/bfc/`.

[3] MIST consists of several methods, most of them based on EEC [8]. We used the abstraction refinement method that tries to minimize the number of places in the Petri net [7].

[4] `http://mjolnir.cs.ox.ac.uk/soter/`

Table 1. Safe examples that were successfully proved safe. Symbols $\mathbb{Q}$ and $\mathbb{Z}$ denote rational and integer numbers.

Suite	Safety/$\mathbb{Q}$	Safety/$\mathbb{Z}$	Ref./$\mathbb{Q}$	Ref./$\mathbb{Z}$	IIC	BFC	MIST	Total
MIST	14	14	20	20	**23**	21	19	23
BFC	**2**	**2**	**2**	**2**	**2**	**2**	**2**	2
Medical	4	4	4	4	9	**12**	10	12
Bug-tracking	**32**	**32**	**32**	**32**	0	0	0	40
Erlang	32	32	36	**38**	17	26	2	38
Total	84	84	94	**96**	51	61	33	115

Rate of success on safe examples. As shown in Table 1, even with the weakest of the methods —safety based on marking equation over rationals— Petrinizer is able to prove safety for 84 out of 115 examples. Switching to integer arithmetic does not help: the number of examples proved safe remains 84. Using refinement via traps, Petrinizer proves safety for 94 examples. Switching to integer arithmetic in this case helps: Another two examples are proved safe, totaling 96 out of 115 examples. In contrast to these numbers, the most successful existing tool turned out to be BFC, proving safety for only 61 examples. Even though the methods these tools implement are theoretically complete, the tools themselves are limited by the time and space they can use.

Looking at the results accross different suites, we see that Petrinizer performed poorest on the medical suite, proving safety for only 4 out of 12 examples. On the other hand, on the bug-tracking suite, which was completely intractable for other tools, it proved safety for 32 out of 40 examples. Furthermore, using traps and integer arithmetic, Petrinizer successfuly proved safety for all safe Erlang examples. We find this result particularly surprising, as the original verification problems for these examples seem non-trivial.

Invariant sizes. We measure the size of inductive invariants produced by Petrinizer without minimization. We took the number of atomic (non-zero) terms appearing in an invariant's linear expressions as a measure of its size. When we relate sizes of invariants to number of places in the corresponding Petri net (top left graph in Fig. 4), we see that invariants are usually very succinct. As an example, the largest invariant had 814 atomic terms, and the corresponding Petri net, coming from the Erlang suite, had 4,763 places. For the largest Petri net, with 66,950 places, the constructed invariant had 339 atomic terms.

The added benefit of minimization is negligible: there are only four examples where the invariant was reduced, and the reduction was about 2-3%. Thus, invariant minimization does not pay off for these examples.

We also compared sizes of constructed invariants with sizes of invariants produced by IIC [13]. IIC's invariants are expressed as CNF formulas over atoms of the form $x < a$, for a variable x and a constant a. As a measure of size for these formulas, we took the number of atoms they contain. As the bottom left graph in Fig. 4 shows, when compared to IIC's invariants, Petrinizer's invariants are never larger, and are often orders of magnitude smaller.

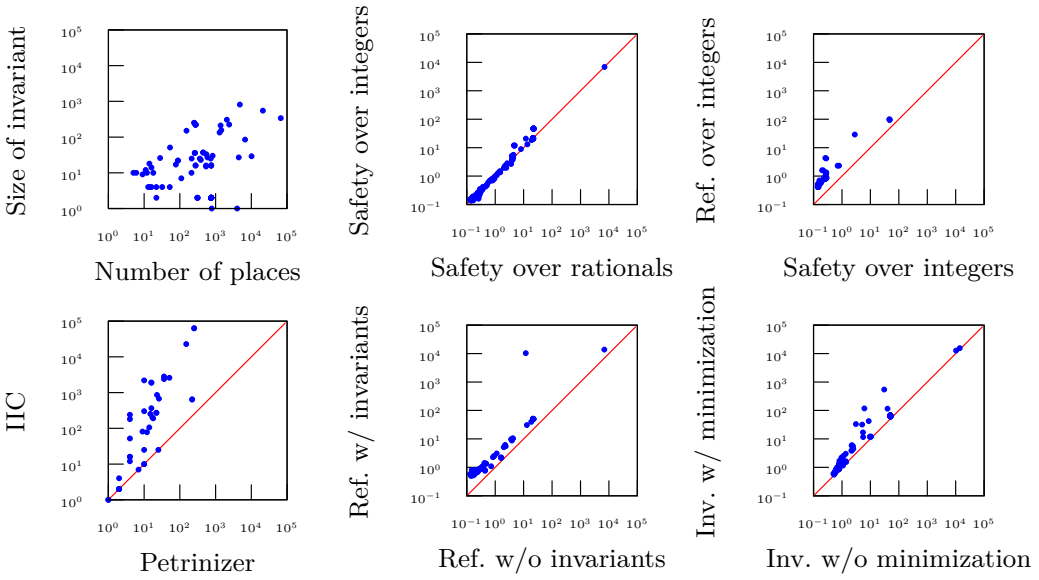


Fig. 4. Graph on the top left shows a relation of sizes of constructed invariants to the number of places in the corresponding Petri nets. Graph on the bottom left shows comparison in size of invariants produced by Petrinizer and IIC. Axes represent size on a logarithmic scale. Each dot represents one example. The four graphs in the center and on the right show time overhead of integer arithmetic, trap refinement, invariant construction and invariant minimization. Axes represent time in seconds on a logarithmic scale. Each dot represents execution time on one example. The graph on the top right only shows examples for which at least one trap appeared in the refinement. Similarly, the bottom center and bottom right graphs only show safe examples.

Performance. To ensure accuracy and fairness, all experiments were performed on identical machines, equipped with Intel Xeon 2.66 GHz CPUs and 48 GB of memory, running Linux 3.2.48.1 in 64-bit mode. Execution time was limited to 100,000 seconds (27 hours, 46 minutes and 40 seconds), and memory to 2 GB.

Due to dissimilarities between the compared tools, selecting a fair measure of time was non-trivial. On the one hand, as Petrinizer communicates with Z3 via temporary files, it spends a considerable amount of time doing I/O operations. On the other hand, as BFC performs both a forward and a backward search, it naturally splits the work into two threads, and runs them in parallel on two CPU cores. In both cases, the actual elapsed time does not quite correspond to the amount of computational effort we wanted to measure. Therefore, for the measure of time we selected the *user time*, as reported by the *time* utility on Linux. User time measures the total CPU time spent executing the process and its children. In the case of Petrinizer, it excludes the I/O overhead, and in the case of BFC, it includes total CPU time spent on both CPU cores.

We report mean and median times measured for each tool in Table 2.

Table 2. Mean and median times in seconds for each tool. We report times for safe examples, as well as for all examples. Memory-out cases were set to the timeout value of 100,000 s. Symbols $\mathbb{Q}$ and $\mathbb{Z}$ denote rational and integer numbers.

Method/tool	Safety/$\mathbb{Q}$	Safety/$\mathbb{Z}$	Ref./$\mathbb{Q}$	Ref./$\mathbb{Z}$	Safety+inv.	Safety+inv.min.
Mean (safe)	69.26	70.20	69.36	72.20	168.46	203.05
Median (safe)	2.45	2.23	2.35	3.81	3.70	4.03
Mean (all)	45.17	46.04	45.52	47.70	109.23	131.58
Median (all)	0.44	0.43	0.90	0.93	0.66	1.00

Method/tool	Ref.+inv.	Ref.+inv.min.	IIC	BFC	MIST
Mean (safe)	228.88	275.12	56954.09	47126.12	69196.77
Median (safe)	5.96	6.30	100000.00	1642.43	100000.00
Mean (all)	148.57	178.45	44089.93	31017.80	61586.56
Median (all)	1.37	1.94	138.00	0.77	100000.00

Time overhead of Petrinizer's methods. Before comparing Petrinizer with other tools, we analyze time overhead of integer arithmetic, trap refinement, invariant construction, and invariant minimization. The four graphs in the center and on the right in Fig. 4 summarize the results. The top central graph shows that the difference in performance between integer and rational arithmetic is negligible.

The top right graph in Fig. 4 shows that traps incur a significant overhead. This is not too surprising as, each time a trap is found, the main system has to be updated with a new trap constraint and solved again. Thus the actual overhead depends on the number of traps that appear during the refinement. In the experiments, there were 32 examples for refinement with integer arithmetic where traps appeared at least once. The maximal number of traps in a single example was 9. In the examples where traps appear once, we see a slowdown of 2-3×. In the extreme cases with 9 traps we see slowdowns of 10-16×.

In the case of invariant construction, as shown on the bottom central graph in Fig. 4, the overhead is more uniform and predictable. The reason is that constructing the invariant involves solving the dual form of the main system as many times as there are disjuncts in the property violation constraint. In most cases, the property violation constraint has one disjunct. A single example with many disjuncts, having 8989 of them, appears on the graph as an outlier.

In the case of invariant minimization, as the bottom right graph in Fig. 4 shows, time overhead is quite severe. The underlying data contains examples of slowdowns of up to 30×.

Comparison with other tools. The six graphs in Fig. 5 show the comparison of execution times for Petrinizer vs. IIC, BFC, and MIST. In the comparison, we used the refinement methods, both with and without invariant construction. In general, we observe that other tools outperform Petrinizer on small examples, an effect that can be explained by the overhead of starting script interpreters and Z3. However, on large examples Petrinizer consistently outperforms other tools. Not only does it finish in all cases within the given time and memory constraints, it even finishes in under 100 seconds in all but two cases. The two cases are the large example from the Erlang suite, with 66,950 places and 213,635 transitions

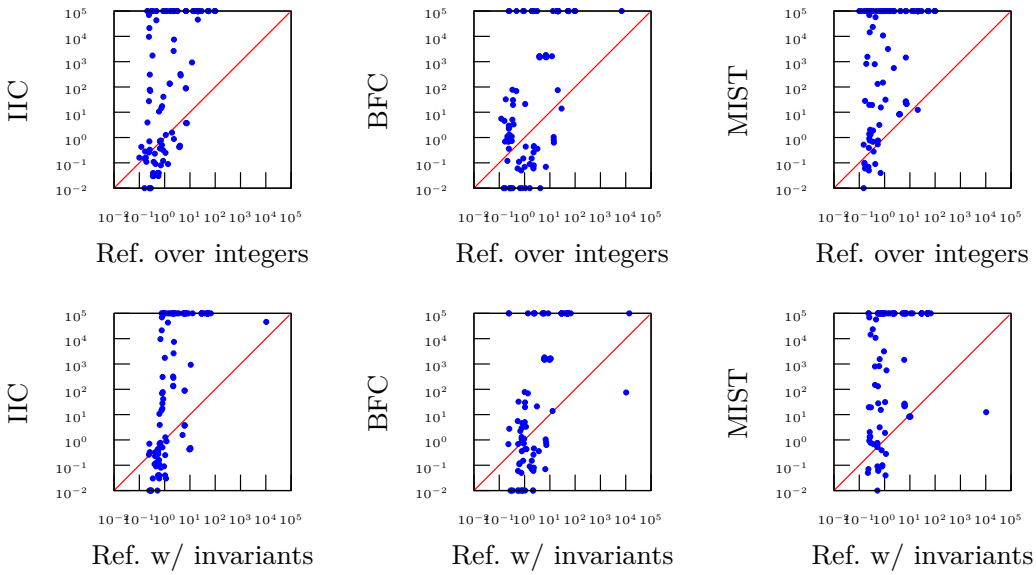


Fig. 5. Comparison of execution time for Petrinizer vs. IIC, BFC and MIST. Graphs in the top row show comparison in the case without invariant construction, and graphs in the bottom row show comparison in the case with invariant construction. Axes represent time in seconds on a logarithmic scale. Each dot represents execution time on one example.

and, in the case of invariant construction, the example from the MIST suite, with 8989 disjuncts in the property violation constraint.

Conclusions. Marking equations and traps are classical techniques in Petri net theory, but have fallen out of favor in recent times in comparison with state-space traversal techniques in combination with abstractions or symbolic representations. Our experiments demonstrate that, when combined with the power of a modern SMT solver, these techniques can be surprisingly effective in finding proofs of correctness (inductive invariants) of common benchmark examples arising out of software verification.

Our results also suggest incorporating these techniques into existing tools as a cheap preprocessing step. A finer integration with these tools is conceivable, where a satisfying assignment to a system of constraints is used to guide the more sophisticated search, similar to [22].

Acknowledgements. We thank Emanuele D'Osualdo for help with the Soter tool. Ledesma-Garza was supported by the Collaborative Research Center 1480 "Program and Model Analysis" funded by the German Research Council.

References

1. Bouajjani, A., Emmi, M.: Bounded phase analysis of message-passing programs. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 451–465. Springer, Heidelberg (2012)
2. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
3. Delzanno, G., Raskin, J.-F., Van Begin, L.: Towards the automated verification of multithreaded java programs. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 173–187. Springer, Heidelberg (2002)
4. D'Osualdo, E., Kochems, J., Ong, C.-H.L.: Automatic verification of Erlang-style concurrency. In: Logozzo, F., Fähndrich, M. (eds.) Static Analysis. LNCS, vol. 7935, pp. 454–476. Springer, Heidelberg (2013)
5. Esparza, J., Melzer, S.: Verification of safety properties using integer programming: Beyond the state equation. Formal Methods in System Design 16(2), 159–189 (2000)
6. Ganty, P., Majumdar, R.: Algorithmic verification of asynchronous programs. ACM Trans. Program. Lang. Syst. 34(1), 6 (2012)
7. Ganty, P., Raskin, J.-F., Van Begin, L.: From many places to few: Automatic abstraction refinement for Petri nets. Fundam. Inform. 88(3), 275–305 (2008)
8. Geeraerts, G., Raskin, J.-F., Begin, L.V.: Expand, enlarge and check: New algorithms for the coverability problem of WSTS. J. Comput. Syst. Sci. 72(1), 180–203 (2006)
9. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. Journal of the ACM (JACM) 39(3), 675–735 (1992)
10. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 645–659. Springer, Heidelberg (2010)
11. Kaiser, A., Kroening, D., Wahl, T.: Efficient coverability analysis by proof minimization. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 500–515. Springer, Heidelberg (2012)
12. Karp, R., Miller, R.: Parallel program schemata. J. Comput. Syst. Sci. 3(2), 147–195 (1969)
13. Kloos, J., Majumdar, R., Niksic, F., Piskac, R.: Incremental, inductive coverability. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 158–173. Springer, Heidelberg (2013)
14. Lamport, L.: The mutual exclusion problem: Part II—statement and solutions. J. ACM 33(2), 327–348 (1986)
15. Majumdar, R., Meyer, R., Wang, Z.: Static provenance verification for message passing programs. In: Logozzo, F., Fähndrich, M. (eds.) SAS 2013. LNCS, vol. 7935, pp. 366–387. Springer, Heidelberg (2013)
16. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
17. Rackoff, C.: The covering and boundedness problems for vector addition systems. Theor. Comput. Sci. 6, 223–231 (1978)
18. Reisig, W.: Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies. Springer (2013)

19. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons Ltd. (1986)
20. Solar-Lezama, A., Tancau, L., Bodík, R., Seshia, S., Saraswat, V.: Combinatorial sketching for finite programs. In: ASPLOS, pp. 404–415. ACM (2006)
21. Valmari, A., Hansen, H.: Old and new algorithms for minimal coverability sets. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 208–227. Springer, Heidelberg (2012)
22. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. Logical Methods in Computer Science 8(3) (2012)

Paper E

Towards Efficient Verification of Population Protocols. PODC, 2017.

This section has been published as a **peer-reviewed conference paper**. The thesis author is **not** first author of the paper.

© Michael Blondin, Javier Esparza, Stefan Jaax and Philipp J. Meyer.

M. Blondin, J. Esparza, S. Jaax, and P. J. Meyer. "Towards Efficient Verification of Population Protocols". In: *Proceedings of the 36th ACM Symposium on Principles of Distributed Computing, PODC 2017*. Ed. by E. M. Schiller and A. A. Schwarzmann. ACM, 2017, pp. 423–430. DOI: 10.1145/3087801.3087816

Summary

We analyze the *well-specification* and *correctness* problems for population protocols, which are known to be TOWER-hard [Esp+15]. We identify a subclass of population protocols for which these two problems have a much lower complexity, but which still has the same computational power as general population protocols. The decision procedures for this class build upon two components: deciding *strong consensus*, a safety property with complexity in co-NP, for which we extend the constraint-based methods from [Esp+14] using the marking equation with traps and siphons, and deciding *layered termination*, a liveness property with complexity in NP, for which we extend the methods of [EM15] to the special fairness requirements in population protocols. We experimentally evaluate our method on a set of population protocols from the literature. We can show well-specification and correctness for protocols with up to several hundreds of states and thousands of transitions within an hour. Our implementation is the first fully automatic verification tool for population protocols able to fully verify a large set of relevant protocols.

Contributions of thesis author

Composition and revision of the manuscript. Joint discussion and development of the theoretical results presented in the paper, with the following notable individual contributions: development of the constraints for StrongConsensus; implementation of the verification method in the tool Peregrine; conducting the experimental evaluation; writing Section 6 with the experimental results.

Towards Efficient Verification of Population Protocols

Michael Blondin
Technische Universität München
blondin@in.tum.de

Javier Esparza
Technische Universität München
esparza@in.tum.de

Stefan Jaax
Technische Universität München
jaax@in.tum.de

Philipp J. Meyer
Technische Universität München
meyerphi@in.tum.de

ABSTRACT

Population protocols are a well established model of computation by anonymous, identical finite state agents. A protocol is well-specified if from every initial configuration, all fair executions of the protocol reach a common consensus. The central verification question for population protocols is the *well-specification problem*: deciding if a given protocol is well-specified. Esparza et al. have recently shown that this problem is decidable, but with very high complexity: it is at least as hard as the Petri net reachability problem, which is EXPSPACE-hard, and for which only algorithms of non-primitive recursive complexity are currently known.

In this paper we introduce the class WS^3 of well-specified strongly-silent protocols and we prove that it is suitable for automatic verification. More precisely, we show that WS^3 has the same computational power as general well-specified protocols, and captures standard protocols from the literature. Moreover, we show that the membership problem for WS^3 reduces to solving boolean combinations of linear constraints over $\mathbb{N}$. This allowed us to develop the first software able to automatically prove well-specification for *all* of the infinitely many possible inputs.

CCS CONCEPTS

• **Networks → Protocol testing and verification**; • **Theory of computation → Logic and verification**;

KEYWORDS

population protocols; automated verification; termination

1 INTRODUCTION

Population protocols [1, 2] are a model of distributed computation by many anonymous finite-state agents. They were initially introduced to model networks of passively mobile sensors [1, 2], but are now also used to describe chemical reaction networks (see e.g. [7, 19]).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PODC '17, July 25-27, 2017, Washington, DC, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-4992-5/17/07…$15.00
http://dx.doi.org/10.1145/3087801.3087816

In each computation step of a population protocol, a fixed number of agents are chosen nondeterministically, and their states are updated according to a joint transition function. Since agents are anonymous and identical, the global state of a protocol is completely determined by the number of agents at each local state, called a configuration. A protocol computes a boolean value b for a given initial configuration C_0 if in all fair executions starting at C_0, all agents eventually agree to b — so, intuitively, population protocols compute by reaching consensus under a certain fairness condition. A protocol is *well-specified* if it computes a value for each of its infinitely many initial configurations (also called *inputs*). The predicate computed by a protocol is the function that assigns to each input the corresponding consensus value. In a famous series of papers, Angluin *et al.* [1, 2] have shown that well-specified protocols compute exactly the predicates definable in Presburger arithmetic [1–4].

In this paper we search for efficient algorithms for the *well-specification problem*: Given a population protocol, is it well-specified? This is a question about an infinite family of finite-state systems. Indeed, for every input the semantics of a protocol is a finite graph with the reachable configurations as nodes. Deciding if the protocol reaches consensus for a fixed input only requires to inspect one of these graphs, and can be done automatically using a model checker. This approach has been followed in a number of papers [6, 8, 20, 24], but it only shows well-specification for some inputs. There has also been work in formalizing well-specification proofs in interactive theorem provers [10], but this approach is not automatic: a human prover must first come up with a proof for each particular protocol.

Recently, the second author, together with other co-authors, has shown that the well-specification problem is decidable [13]. That is, there is an algorithm that decides if for all inputs the protocol stabilizes to a boolean value. The proof uses deep results of the theory of Petri nets, a model very close to population protocols. However, the same paper shows that the well-specification problem is at least as hard as the reachability problem for Petri nets, a famously difficult problem. More precisely, the problem is known to be EXPSPACE-hard, and all known algorithms for it have non-primitive recursive complexity [22]. In particular, there are no stable implementations of any of these algorithms, and they are considered impractical for nearly all applications.

For this reason, in this paper we search for a class of well-specified protocols satisfying three properties:

(a) *No loss of expressive power*: the class should compute all Presburger-definable predicates.

(b) *Natural*: the class should contain most protocols discussed in the literature.
(c) *Feasible membership problem*: membership for the class should have reasonable complexity.

The class *WS* of all well-specified protocols obviously satisfies (a) and (b), but not (c). So we introduce a new class WS^3, standing for *Well-Specified Strongly Silent* protocols. We show that WS^3 still satisfies (a) and (b), and then prove that the membership problem for WS^3 is in the complexity class DP; the class of languages L such that $L = L_1 \cap L_2$ for some languages $L_1 \in \mathsf{NP}$ and $L_2 \in \mathsf{coNP}$. This is a dramatic improvement with respect to the EXPSPACE-hardness of the membership problem for *WS*.

Our proof that the problem is in DP reduces membership for WS^3 to checking (un)satisfiability of two systems of boolean combinations of linear constraints over the natural numbers. This allowed us to implement our decision procedure on top of the constraint solver Z3 [9], yielding the first software able to automatically prove well-specification for *all* inputs. We tested our implementation on the families of protocols studied in [6, 8, 20, 24]. These papers prove well-specification for some inputs of protocols with up to 9 states and 28 transitions. Our approach proves well-specification for all inputs of protocols with up to 20 states in less than one second, and protocols with 70 states and 2500 transitions in less than one hour. In particular, we can automatically prove well-specification for *all* inputs in less time than previous tools needed to check *one single large input*.

The verification problem for population protocols naturally divides into two parts: checking that a given protocol is well specified, and checking that a given well-specified protocol computes the desired predicate. While in this paper we are concerned with well-specification, our implementation is already able to solve the second problem for all the families of protocols described above. This is achieved by adding to the second system of constraints used to check well-specification further linear constraints describing the sets of input configurations for which the protocol should return *true* or *false*. An extension of the software that, given a protocol and an arbitrary Presburger predicate, checks whether the protocol computes the predicate, requires to solve implementation problems related to Presburger arithmetic, and is left for future research.

The paper is organized as follows. Section 2 contains basic definitions. Section 3 introduces an intermediate class WS^2 of silent well-specified protocols, and shows that its membership problem is still as hard as for *WS*. In Section 4, we characterize WS^2 in terms of two properties which are then strengthened to define our new class WS^3. We then show that the properties defining WS^3 can be tested in NP and coNP, and so that membership for WS^3 is in DP. Section 5 proves that WS^3-protocols compute all Presburger predicates. Section 6 reports on our experimental results, and Section 7 presents conclusions.

2 PRELIMINARIES

Multisets. A *multiset* over a finite set E is a mapping $M : E \to \mathbb{N}$. The set of all multisets over E is denoted $\mathbb{N}^E$. For every $e \in E$, $M(e)$ denotes the number of occurrences of e in M. We sometimes denote multisets using a set-like notation, e.g. $\{\!| f, g, g |\!\}$ is the multiset M such that $M(f) = 1$, $M(g) = 2$ and $M(e) = 0$ for every $e \in E \setminus \{f, g\}$. The *support* of $M \in \mathbb{N}^E$ is $[\![M]\!] \stackrel{\text{def}}{=} \{e \in E : M(e) > 0\}$. The *size* of $M \in \mathbb{N}^E$ is $|M| \stackrel{\text{def}}{=} \sum_{e \in E} M(e)$. Addition and comparison are extended to multisets componentwise, i.e. $(M + M')(e) \stackrel{\text{def}}{=} M(e) + M'(e)$ for every $e \in E$, and $M \leq M' \stackrel{\text{def}}{\iff} M(e) \leq M(e)$ for every $e \in E$. We define multiset difference as $(M \ominus M')(e) \stackrel{\text{def}}{=} \max(M(e) - M'(e), 0)$ for every $e \in E$. The empty multiset is denoted $\mathbf{0}$, and for every $e \in E$ we write $\boldsymbol{e} \stackrel{\text{def}}{=} \{\!| e |\!\}$.

Population protocols. A *population* P over a finite set E is a multiset $P \in \mathbb{N}^E$ such that $|P| \geq 2$. The set of all populations over E is denoted by $\mathrm{Pop}(E)$. A *population protocol* is a tuple $\mathcal{P} = (Q, T, \Sigma, I, O)$ where

- Q is a non-empty finite set of *states*,
- $T \subseteq Q^2 \times Q^2$ is a set of *transitions* such that for every $(p, q) \in Q^2$ there exists at least a pair $(p', q') \in Q^2$ such that $(p, q, p', q') \in T$,
- Σ is a non-empty finite *input alphabet*,
- $I : \Sigma \to Q$ is the *input function* mapping input symbols to states,
- $O : Q \to \{0, 1\}$ is the *output function* mapping states to boolean values.

Following the convention of previous papers, we call the populations of $\mathrm{Pop}(Q)$ *configurations*. Intuitively, a configuration C describes a collection of identical finite-state *agents* with Q as set of states, containing $C(q)$ agents in state q for every $q \in Q$, and at least two agents in total.

Pairs of agents[1] interact using transitions. For every $t = (p, q, p', q') \in T$, we write $(p, q) \mapsto (p', q')$ to denote t, and we define $\mathrm{pre}(t) \stackrel{\text{def}}{=} \{\!| p, q |\!\}$ and $\mathrm{post}(t) \stackrel{\text{def}}{=} \{\!| p', q' |\!\}$. For every configuration C and transition $t \in T$, we say that t is *enabled* at C if $C \geq \mathrm{pre}(t)$. Note that by definition of T, every configuration enables at least one transition. A transition $t \in T$ enabled at C can *occur*, leading to the configuration $C \ominus \mathrm{pre}(t) + \mathrm{post}(t)$. Intuitively, a pair of agents in states $\mathrm{pre}(t)$ move to states $\mathrm{post}(t)$. We write $C \xrightarrow{t} C'$ to denote that t is enabled at C and that its occurrence leads to C'. A transition $t \in T$ is *silent* if $\mathrm{pre}(t) = \mathrm{post}(t)$, i.e., if it cannot change the current configuration.

For every sequence of transitions $w = t_1 t_2 \cdots t_k$, we write $C \xrightarrow{w} C'$ if there exists a sequence of configurations $C_0, C_1, \ldots, C_k$ such that $C = C_0 \xrightarrow{t_1} C_1 \cdots \xrightarrow{t_k} C_k = C'$. We also write $C \to C'$ if $C \xrightarrow{t} C'$ for some transition $t \in T$, and call $C \to C'$ a *step*. We write $C \xrightarrow{*} C'$ if $C \xrightarrow{w} C'$ for some $w \in T^*$. We say that C' is *reachable from* C if $C \xrightarrow{*} C'$. An *execution* is an infinite sequence of configurations $C_0 C_1 \cdots$ such that $C_i \to C_{i+1}$ for every $i \in \mathbb{N}$. An execution $C_0 C_1 \cdots$ is *fair* if for every step $C \to C'$, if $C_i = C$ for infinitely many indices $i \in \mathbb{N}$, then $C_j = C$ and $C_{j+1} = C'$ for infinitely many indices $j \in \mathbb{N}$. We say that a configuration C is

- *terminal* if $C \xrightarrow{*} C'$ implies $C = C'$, i.e., if every transition enabled at C is silent;
- a *consensus configuration* if $O(p) = O(q)$ for every $p, q \in [\![C]\!]$.

[1] While protocols only model interactions between two agents, k-way interactions for a fixed $k > 2$ can be simulated by adding additional states.

For every consensus configuration C, let $O(C)$ denote the unique output of the states in $[\![C]\!]$. An execution $C_0C_1\cdots$ *stabilizes* to $b \in \{0,1\}$ if there exists $n \in \mathbb{N}$ such that C_i is a consensus configuration and $O(C_i) = b$ for every $i \geq n$.

Predicates computable by population protocols. Every *input* $X \in \mathrm{Pop}(\Sigma)$ is mapped to the configuration $I(X) \in \mathrm{Pop}(Q)$ defined by

$$I(X)(q) \stackrel{\text{def}}{=} \sum_{\substack{\sigma\in\Sigma \\ I(\sigma)=q}} X(\sigma) \text{ for every } q \in Q.$$

A configuration C is said to be *initial* if $C = I(X)$ for some input X. A population protocol is *well-specified* if for every input X, there exists $b \in \{0,1\}$ such that every fair execution of $\mathcal{P}$ starting at $I(X)$ stabilizes to b. We say that $\mathcal{P}$ *computes* a predicate φ if for every input X, every fair execution of $\mathcal{P}$ starting at $I(X)$ stabilizes to $\varphi(X)$. It is readily seen that $\mathcal{P}$ computes a predicate if and only if it is well-specified.

Example 2.1. We consider the majority protocol of [3] as a running example. Initially, agents of the protocol can be in either state A or B. The protocol computes whether there are at least as many agents in state B as there are in state A. The states and the input alphabet are $Q = \{A, B, a, b\}$ and $\Sigma = \{A, B\}$ respectively. The input function is the identity function, and the output function is given by $O(B) = O(b) = 1$ and $O(A) = O(a) = 0$. The set of transitions T consists of:

$$t_{AB} = (A,B) \mapsto (a,b)$$
$$t_{Ab} = (A,b) \mapsto (A,a)$$
$$t_{Ba} = (B,a) \mapsto (B,b)$$
$$t_{ba} = (b,a) \mapsto (b,b)$$

and of silent transitions for the remaining pairs of states. Transition t_{AB} ensures that every fair execution eventually reaches a configuration C such that $C(A) = 0$ or $C(B) = 0$. If $C(A) = 0 = C(B)$, then there were initially equally many agents in A and B. Transition t_{ba} then acts as tie breaker, resulting in a terminal configuration populated only by b. If, say, $C(A) > 0$ and $C(B) = 0$, then there were initially more As than Bs, and t_{Ab} ensures that every fair execution eventually reaches a terminal configuration populated only by A and a.

3 WELL-SPECIFIED SILENT PROTOCOLS

Silent protocols[2] were introduced in [12]. Loosely speaking, a protocol is silent if communication between agents eventually ceases, i.e. if every fair execution eventually stays in the same configuration forever. Observe that a well-specified protocol need not be silent: fair executions may keep alternating from a configuration to another as long as they are consensus configurations with the same output.

More formally, we say that an execution $C_0C_1\cdots$ is *silent* if there exists $n \in \mathbb{N}$ and a configuration C such that $C_i = C$ for every $i \geq n$. A population protocol $\mathcal{P}$ is *silent* if every fair execution of $\mathcal{P}$ is silent, regardless of the starting configuration. We call a protocol that is well-specified and silent a *WS^2-protocol*, and denote by WS^2 the set of all WS^2-protocols.

[2] Silent protocols are also referred to as *protocols with stabilizing states* and silent transitions are called *ineffective* in [17, 18].

Example 3.1. As explained in Example 2.1, every fair execution of the majority protocol is silent. This implies that the protocol is silent. If, for example, we add a new state b' where $O(b') = 1$, and transitions $(b,b) \mapsto (b',b')$, $(b',b') \mapsto (b,b)$, then the protocol is no longer silent since the execution where two agents alternate between states b and b' is fair but not silent.

Being silent is a desirable property. While in arbitrary protocols it is difficult to determine if an execution has already stabilized, in silent protocols it is simple: one just checks if the current configuration only enables silent transitions. Even though it is was not observed explicitely, the protocols introduced in [1] to characterize the expressive power of population protocols belong to WS^2. Therefore, WS^2-protocols can compute the same predicates as general ones.

Unfortunately, a slight adaptation of [14, Theorem 10] shows that the complexity of the membership problem for WS^2-protocols is still as high as for the general case:

Proposition 3.2. *The reachability problem for Petri nets is reducible in polynomial time to the membership problem for WS^2. In particular, membership for WS^2 is* EXPSPACE*-complete.*

To circumvent this high complexity, we will show in the next section how WS^2 can be refined into a smaller class of well-specified protocols with the same expressive power, and a membership problem of much lower complexity.

4 A FINER CLASS OF SILENT WELL-SPECIFIED PROTOCOLS: WS^3

It can be shown that WS^2-protocols are exactly the protocols satisfying the two following properties:

- Termination: for every configuration C, there exists a terminal configuration C' such that $C \stackrel{*}{\to} C'$.
- Consensus: for every initial configuration C, there exists $b \in \{0,1\}$ such that every terminal configuration C' reachable from C is a consensus configuration with output b, i.e. $C \stackrel{*}{\to} C'$ implies $O(C') = b$.

We will introduce the new class WS^3 as a refinement of WS^2 obtained by strengthening Termination and Consensus into two new properties called LayeredTermination and StrongConsensus. We introduce these properties in Section 4.1 and Section 4.2, and show that their decision problems belong to NP and coNP respectively.

Before doing so, let us introduce some useful notions. Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol. For every $S \subseteq T$, $\mathcal{P}[S]$ denotes the *protocol induced by* S, i.e. $\mathcal{P}[S] \stackrel{\text{def}}{=} (Q, S \cup T', \Sigma, I, O)$ where $T' \stackrel{\text{def}}{=} \{(p,q,p,q) : p,q \in Q\}$ is added to ensure that any two states can interact. Let $\to_S$ denote the transition relation of $\mathcal{P}[S]$. An *ordered partition* of T is a tuple $(T_1, T_2, \ldots, T_n)$ of nonempty subsets of T such that $T = \bigcup_{i=1}^{n} T_i$ and $T_i \cap T_j = \emptyset$ for every $1 \leq i < j \leq n$.

4.1 Layered termination

We replace Termination by a stronger property called LayeredTermination, and show that deciding LayeredTermination belongs to NP. The definition of LayeredTermination is inspired by the typical structure of protocols found in the literature. Such

protocols are organized in layers such that transitions of higher layers cannot be enabled by executing transitions of lower layers. In particular, if the protocol reaches a configuration of the highest layer that does not enable any transition, then this configuration is terminal. For such protocols, Termination can be proven by showing that every (fair or unfair) execution of a layer is silent.

Definition 4.1. A population protocol $\mathcal{P} = (Q, T, \Sigma, I, O)$ satisfies LayeredTermination if there is an ordered partition

$$(T_1, T_2, \ldots, T_n)$$

of T such that the following properties hold for every $i \in [n]$:

(a) For every configuration C, every (fair or unfair) execution of $\mathcal{P}[T_i]$ starting at C is silent.

(b) For every configurations C and C', if $C \xrightarrow{*}_{T_i} C'$ and C is terminal in $\mathcal{P}[T_1 \cup T_2 \cup \cdots \cup T_{i-1}]$, then C' is also terminal in $\mathcal{P}[T_1 \cup T_2 \cup \cdots \cup T_{i-1}]$.

Example 4.2. The majority protocol satisfies LayeredTermination. Indeed, consider the ordered partition (T_1, T_2), where

$$T_1 = \{(A, B) \mapsto (a, b), (A, b) \mapsto (A, a)\}$$
$$T_2 = \{(B, a) \mapsto (B, b), (b, a) \mapsto (b, b)\}.$$

All executions of $\mathcal{P}[T_1]$ and $\mathcal{P}[T_2]$ are silent. For every terminal configuration C of $\mathcal{P}[T_1]$, we have $[\![C]\!] \subseteq \{A, a\}$ or $[\![C]\!] \subseteq \{B, a, b\}$. In the former case, no transition of T_2 is enabled; in the latter case, taking transitons of T_2 cannot enable T_1.

As briefly sketched above, LayeredTermination implies Termination. In the rest of this section, we prove that checking LayeredTermination is in NP. We do this by showing that conditions (a) and (b) of Definition 4.1 can be tested in polynomial time.

We recall a basic notion of Petri net theory recast in the terminology of population protocols. For every step $C \xrightarrow{t} C'$ and every state q of a population protocol, we have $C'(q) = C(q) + \mathrm{post}(t)(q) - \mathrm{pre}(t)(q)$. This observation can be extended to sequences of transitions. Let $|w|_t$ denote the number of occurrences of transition t in a sequence w. We have $C'(q) = C(q) + \sum_{t \in T} |w|_t \cdot (\mathrm{post}(t)(q) - \mathrm{pre}(t)(q))$. Thus, a necessary condition for $C \xrightarrow{w} C'$ is the existence of some $\boldsymbol{x} : T \to \mathbb{N}$ such that

$$C'(q) = C(q) + \sum_{t \in T} \boldsymbol{x}(t) \cdot (\mathrm{post}(t)(q) - \mathrm{pre}(t)(q)). \qquad (1)$$

We call (1) the *flow equation* for state q.

Proposition 4.3. *Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol. Deciding whether an ordered partition $(T_1, T_2, \ldots, T_n)$ of T satisfies condition (a) of Definition 4.1 can be done in polynomial time.*

Proof. Let $i \in [n]$ and let U_i be the set of non silent transitions of T_i. It can be shown that $\mathcal{P}[T_i]$ is non silent if and only if there exists $\boldsymbol{x} : U_i \to \mathbb{Q}$ such that $\sum_{t \in U_i} \boldsymbol{x}(t) \cdot (\mathrm{post}(t)(q) - \mathrm{pre}(t)(q)) = 0$ and $\boldsymbol{x}(q) \geq 0$ for every $q \in Q$, and $\boldsymbol{x}(q) > 0$ for some $q \in Q$. Therefore, since linear programming is in P, we can check for the (non) existence of an appropriate rational solution $\boldsymbol{x}_i$ for every $i \in [n]$. □

We show how to check condition (b) of Definition 4.1 in polynomial time. Let $U \subseteq T$ be a set of transitions. A configuration $C \in \mathrm{Pop}(Q)$ is *U-dead* if for every $t \in U$, $C \xrightarrow{t} C'$ implies $C' = C$. We say that $\mathcal{P}$ is *U-dead from* $C_0 \in \mathrm{Pop}(Q)$ if every configuration reachable from C_0 is U-dead, i.e. $C_0 \xrightarrow{*} C$ implies that C is U-dead. Finally, we say that $\mathcal{P}$ is *U-dead* if it is U-dead from every U-dead configuration $C_0 \in \mathrm{Pop}(Q)$.

Proposition 4.4. *Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol. Deciding whether an ordered partition $(T_1, \ldots, T_n)$ of T satisfies condition (b) of Definition 4.1 can be done in polynomial time.*

Proof. Let $i \in [n]$ and let $U \stackrel{\text{def}}{=} T_1 \cup T_2 \cup \cdots \cup T_{i-1}$. $\mathcal{P}[T_i]$ satisfies condition (b) if and only if $\mathcal{P}[T_i]$ is U-dead. The latter can be tested in polynomial time through the following characterization: $\mathcal{P}[T_i]$ is *not* U-dead if and only if there exist $t \in T_i$ and non silent $u \in U$ such that for every non silent $u' \in U$:

$$\mathrm{pre}(u') \not\leq \mathrm{pre}(t) + (\mathrm{pre}(u) \ominus \mathrm{post}(t)). \qquad \square$$

Propositions 4.3 and 4.4 yield an NP procedure to decide LayeredTermination. Indeed, it suffices to guess an ordered partition and to check whether it satisfies conditions (a) and (b) of Definition 4.1 in polynomial time.

Corollary 4.5. *Deciding if a protocol satisfies LayeredTermination is in* NP.

4.2 Strong consensus

To overcome the high complexity of reachability in population protocols, we strengthen Consensus by replacing the reachability relation in its definition by an *over-approximation*, i.e., a relation $\dashrightarrow$ over configurations such that $C \xrightarrow{*} C'$ implies $C \dashrightarrow C'$. Observe that the flow equations provide an over-approximation of the reachability relation. Indeed, as mentioned earlier, if $C \xrightarrow{*} C'$, then there exists $\boldsymbol{x} : T \to \mathbb{N}$ such that $(C, C', \boldsymbol{x})$ satisfies all of the flow equations. However, this over-approximation alone is too crude for the verification of protocols.

Example 4.6. For example, let us consider the configurations $C = \{\!|A, B|\!\}$ and $C' = \{\!|a, a|\!\}$ of the majority protocol. The flow equations are satisfied by the mapping $\boldsymbol{x}$ such that $\boldsymbol{x}(t_{AB}) = \boldsymbol{x}(t_{Ab}) = 1$ and $\boldsymbol{x}(t_{Ba}) = \boldsymbol{x}(t_{ba}) = 0$. Yet, $C \xrightarrow{*} C'$ does not hold.

To obtain a finer reachability over-approximation, we introduce so-called traps and siphons constraints borrowed from the theory of Petri nets [11, 15, 16] and successfully applied to a number of analysis problems (see e.g. [5, 15, 16]). Intuitively, for some subset of transitions $U \subseteq T$, a U-trap is a set of states $P \subseteq Q$ such that every transition of U that removes an agent from P also moves an agent into P. Conversely, a U-siphon is a set $P \subseteq Q$ such that every transition of U that moves an agent into P also removes an agent from P. More formally, let ${}^\bullet R \stackrel{\text{def}}{=} \{t \in T : [\![\mathrm{post}(t)]\!] \cap R \neq \emptyset\}$ and $R^\bullet \stackrel{\text{def}}{=} \{t \in T : [\![\mathrm{pre}(t)]\!] \cap R \neq \emptyset\}$. U-siphons and U-traps are defined as follows:

Definition 4.7. A subset of states $P \subseteq Q$ is a *U-trap* if $P^\bullet \cap U \subseteq {}^\bullet P$, and a *$U$-siphon* if ${}^\bullet P \cap U \subseteq P^\bullet$.

For every configuration $C \in \mathrm{Pop}(Q)$ and $P \subseteq Q$, let $C(P) \stackrel{\text{def}}{=} \sum_{q \in P} C(q)$. Consider a sequence of steps $C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} C_n$

where $t_1, \ldots, t_n \in U$. It follows from Definition 4.7 that if some transition t_i moves an agent to a U-trap P, then $C_j(P) > 0$ for every $j \geq i$. Similarly, if some transition t_i removes an agent from a U-siphon, then $C_j(P) > 0$ for every $j < i$. In particular:

Observation 4.8. *Let $U \subseteq T$, and let C and C' be configurations such that $C \xrightarrow{*}_U C'$. For every U-trap P, if $C'(P) = 0$, then ${}^\bullet P \cap U = \emptyset$. For every U-siphon P, if $C(S) = 0$, then $P^\bullet \cap U = \emptyset$.*

We obtain a necessary condition for $C \xrightarrow{*}_U C'$ to hold, which we call potential reachability:

Definition 4.9. Let C, C' be two configurations, let $\boldsymbol{x} : T \to \mathbb{N}$, and let $U = [\![\boldsymbol{x}]\!]$. We say that C' is *potentially reachable from C through $\boldsymbol{x}$*, denoted $C \xdashrightarrow{\boldsymbol{x}} C'$, if

(a) the flow equation (1) holds for every $q \in Q$,
(b) $C'(P) = 0$ implies ${}^\bullet P \cap U = \emptyset$ for every U-trap P, and
(c) $C(P) = 0$ implies $P^\bullet \cap U = \emptyset$ for every U-siphon P.

Example 4.10. Let us reconsider Example 4.6. Let $U = [\![\boldsymbol{x}]\!] = \{t_{AB}, t_{Ab}\}$ and $P = \{A, b\}$. Recall that $t_{AB} = (A, B) \mapsto (a, b)$ and $t_{Ab} = (A, b) \mapsto (A, a)$. We have $P^\bullet \cap U = U$ which implies that P is a U-trap. This means that Definition 4.9(b) is violated as $C'(P) = 0$ and ${}^\bullet P \cap U = U \neq \emptyset$. Therefore, $\wr A, B \wr \xdashrightarrow{\boldsymbol{x}} \wr a, a \wr$ does not hold.

We write $C \dashrightarrow C'$ if $C \xdashrightarrow{\boldsymbol{x}} C'$ for some $\boldsymbol{x} : T \to \mathbb{N}$. As an immediate consequence of Observation 4.8, for every configurations C and C', if $C \xrightarrow{*} C'$, then $C \dashrightarrow C'$. This allows us to strengthen Consensus by redefining it in terms of potential reachability instead of reachability:

Definition 4.11. A protocol satisfies StrongConsensus if for every initial configuration C, there exists $b \in \{0, 1\}$ such that every terminal configuration C' potentially reachable from C is a consensus configuration with output b, i.e. $C \dashrightarrow C'$ implies $O(C') = b$.

Since the number of U-traps and U-siphons of a protocol can be exponential in the number of states, checking trap and siphon constraints by enumerating them may take exponential time. Fortunately, this can be avoided. By definition, it follows that the union of two U-traps is again a U-trap, and similarly for siphons. Therefore, given a configuration C, there exists a unique maximal U-siphon $P_{\max}$ such that $C(P_{\max}) = 0$, and a unique maximal U-trap $P'_{\max}$ such that $C(P'_{\max}) = 0$. Moreover, $P_{\max}$ and $P'_{\max}$ can be computed in linear time by means of a simple greedy algorithm (see e.g. [11, Ex. 4.5]). This simplifies the task of checking traps and siphons constraints, and yields a coNP procedure for testing StrongConsensus:

Proposition 4.12. *Deciding if a protocol satisfies StrongConsensus is in* coNP.

Proof. Testing whether a protocol *does not* satisfy StrongConsensus can be done by guessing $C_0, C, C' \in \mathrm{Pop}(Q)$, $b \in \{0, 1\}$, $q, q' \in Q$ and $\boldsymbol{x}, \boldsymbol{x}' : T \to \mathbb{N}$, and testing whether

(a) C_0 is initial, C is terminal, C' is terminal, $q \in [\![C]\!]$, $q' \in [\![C']\!]$, $O(q) \neq O(q')$, and
(b) $C_0 \xdashrightarrow{\boldsymbol{x}} C$ and $C_0 \xdashrightarrow{\boldsymbol{x}'} C'$.

Since there is no *a priori* bound on the size of C_0, C, C' and $\boldsymbol{x}, \boldsymbol{x}'$, we guess them carefully. First, we guess whether $D(p) = 0$, $D(p) = 1$ or $D(p) \geq 2$ for every $D \in \{C_0, C, C'\}$ and $p \in Q$. This gives enough information to test (a). Then, we guess $[\![\boldsymbol{x}]\!]$ and $[\![\boldsymbol{x}']\!]$. This allows to test traps/siphons constraints as follows. Let $U \stackrel{\text{def}}{=} [\![\boldsymbol{x}]\!]$, let $P_{\max}$ be the maximal U-trap such that $C(P_{\max}) = 0$, and let $P'_{\max}$ be the maximal U-siphon such that $C_0(P'_{\max}) = 0$. Conditions (b) and (c) of Definition 4.9 hold if and only if ${}^\bullet(P_{\max}) \cap U = \emptyset$ and $(P'_{\max})^\bullet \cap U = \emptyset$, which can be tested in polynomial time. The same is done for $\boldsymbol{x}'$. If (a) and siphons/traps constraints hold, we build the system $\mathcal{S}$ of linear equations/inequalities obtained from the conjunction of the flow equations together with the constraints already guessed. By standard results on integer linear programming (see e.g. [23, Sect. 17]), if $\mathcal{S}$ has a solution, then it has one of polynomial size, and hence we may guess it. □

4.3 WS^3-protocols

We say that a protocol belongs to WS^3 if it satisfies LayeredTermination and StrongConsensus. Since $WS^3 \subseteq WS^2 \subseteq WS$ holds, every WS^3-protocol is well-specified. Recall that a language L belongs to the class DP [21] if there exist languages $L_1 \in$ NP and $L_2 \in$ coNP such that $L = L_1 \cap L_2$. By taking L_1 and L_2 respectively as the languages of population protocols satisfying LayeredTermination and StrongConsensus, Corollary 4.5 and Proposition 4.12 yield:

Theorem 4.13. *The membership problem for WS^3-protocols is in* DP.

5 WS^3 IS AS EXPRESSIVE AS WS

In a famous result, Angluin *et al.* [3] have shown that a predicate is computable by a population protocol if and only if it is definable in Presburger arithmetic, the first-order theory of addition [1, 3]. In particular, [1] constructs protocols for Presburger-definable predicates by means of a well-known result: *Presburger-definable* predicates are the smallest set of predicates containing all threshold and remainder predicates, and closed under boolean operations. A *threshold predicate* is a predicate of the form

$$P(x_1, \ldots, x_k) = \left(\sum_{i=1}^{k} a_i x_i < c \right),$$

where $k \geq 1$ and $a_1, \ldots, a_k, c \in \mathbb{Z}$. A *remainder predicate* is a predicate of the form

$$P(x_1, \ldots, x_k) = \left(\sum_{i=1}^{k} a_i x_i \equiv c \pmod{m} \right),$$

where $k \geq 1$, $m \geq 2$ and $a_1, \ldots, a_k, c, m \in \mathbb{Z}$. Here, we show that these predicates can be computed by WS^3-protocols, and that WS^3 is closed under negation and conjunction. As a consequence, we obtain that WS^3-protocols are as expressive as WS, the class of all well-specified protocols.

Threshold. We describe the protocol given in [1] to compute the threshold predicate $\sum_{i=1}^{k} a_i x_i < c$. Let

$$v_{\max} \stackrel{\text{def}}{=} \max(|a_1|, |a_2|, \ldots, |a_k|, |c| + 1)$$

and define

$$f(m,n) \stackrel{\text{def}}{=} \max(-v_{\max}, \min(v_{\max}, m+n))$$
$$g(m,n) \stackrel{\text{def}}{=} (m+n) - f(m,n)$$
$$b(m,n) \stackrel{\text{def}}{=} (f(m,n) < c)$$

The protocol is $\mathcal{P}_{\text{thr}} \stackrel{\text{def}}{=} (Q, T, \Sigma, I, O)$, where

$$Q \stackrel{\text{def}}{=} \{0,1\} \times [-v_{\max}, v_{\max}] \times \{0,1\}$$
$$\Sigma \stackrel{\text{def}}{=} \{x_1, x_2, \ldots, x_k\}$$
$$I(x_i) \stackrel{\text{def}}{=} (1, a_i, a_i < c) \text{ for every } i \in [k]$$
$$O(\ell, n, o) \stackrel{\text{def}}{=} o \text{ for every state } (\ell, n, o),$$

and T contains

$$(1,n,o), (l,n',o') \mapsto (1, f(n,n'), b(n,n')), (0, g(n,n'), b(n,n'))$$

for every $n, n' \in [-v_{\max}, v_{\max}]$, $\ell, o, o' \in \{0,1\}$. Intuitively, a state (ℓ, n, o) indicates that the agent has value n, opinion o, and that it is a leader if and only if $\ell = 1$. When a leader q and a state r interact, r becomes a non leader, and q increases its value as much as possible by substracting from the value of r. Moreover, a leader can change the opinion of any non leader.

Proposition 5.1. *$\mathcal{P}_{thr}$ satisfies StrongConsensus.*

Proof. Let $\text{val}(q) \stackrel{\text{def}}{=} n$ for every state $q = (\ell, n, o) \in Q$, and let $\text{val}(C) \stackrel{\text{def}}{=} \sum_{q \in Q} C(q) \cdot \text{val}(q)$ for every configuration $C \in \text{Pop}(Q)$. The following holds for every $C, C' \in \text{Pop}(Q)$:

(a) If $(C, C', \boldsymbol{x})$ is a solution to the flow equations for some $\boldsymbol{x} : T \to \mathbb{N}$, then $\text{val}(C) = \text{val}(C')$.
(b) If C, C' are terminal, C and C' contain a leader, and $\text{val}(C) = \text{val}(C')$, then $O(C) = O(C')$.

Suppose for the sake of contradiction that $\mathcal{P}$ does not satisfy StrongConsensus. There exist $C_0, C, C' \in \text{Pop}(Q)$, $q, q' \in Q$ and $\boldsymbol{x}, \boldsymbol{x}' : T \to \mathbb{N}$ such that $C_0 \xrightarrow{\boldsymbol{x}} C$, $C_0 \xrightarrow{\boldsymbol{x}'} C'$, C_0 is initial, C and C' are terminal consensus configurations, $q \in [\![C]\!]$, $q' \in [\![C']\!]$ and $O(q) \neq O(q')$. Note that $(C_0, C, \boldsymbol{x})$ and $(C_0, C', \boldsymbol{x}')$ both satisfy the flow equations. Thus, by (a), $\text{val}(C) = \text{val}(C_0) = \text{val}(C')$. Since C_0 is initial, it contains a leader. Since the set of leaders forms a U-trap for every $U \subseteq T$, and $(C_0, C, \boldsymbol{x})$ and $(C_0, C', \boldsymbol{x})$ satisfy trap constraints, C and C' contain a leader. By (b), C and C' are consensus configurations with $O(C) = O(C')$, which is a contradiction. □

Proposition 5.2. *$\mathcal{P}_{thr}$ satisfies LayeredTermination.*

Proof. Let $L_0 \stackrel{\text{def}}{=} \{(1,x,0) : c \leq x \leq v_{\max}\}$, $L_1 \stackrel{\text{def}}{=} \{(1,x,1) : -v_{\max} \leq x < c\}$, $N_0 \stackrel{\text{def}}{=} \{(0,0,0)\}$ and $N_1 \stackrel{\text{def}}{=} \{(0,0,1)\}$. It can be shown that the following ordered partitions satisfy layered termination for $c > 0$ and $c \leq 0$ respectively:

$$T_1 \stackrel{\text{def}}{=} \{t \in T : \text{pre}(t) \neq \wr q, r \wr \text{ for all } q \in L_0, r \in N_1\},$$
$$T_2 \stackrel{\text{def}}{=} T \setminus T_1, \text{ and}$$
$$S_1 \stackrel{\text{def}}{=} \{t \in T : \text{pre}(t) \neq \wr q, r \wr \text{ for all } q \in L_1, r \in N_0\},$$
$$S_2 \stackrel{\text{def}}{=} T \setminus S_1.$$ □

Remainder. We give a protocol for the remainder predicate

$$\sum_{i=1}^{k} a_i x_i \equiv c \pmod{m}.$$

The protocol is $\mathcal{P}_{\text{rmd}} = (Q, T, \Sigma, I, O)$, where

$$Q \stackrel{\text{def}}{=} [0, m) \cup \{\text{true}, \text{false}\}$$
$$\Sigma \stackrel{\text{def}}{=} \{x_1, x_2, \ldots, x_k\}$$
$$I(x_i) \stackrel{\text{def}}{=} a_i \bmod m \text{ for every } i \in [k]$$
$$O(q) \stackrel{\text{def}}{=} \begin{cases} 1 \text{ if } q \in \{c, \text{true}\} \\ 0 \text{ otherwise} \end{cases}$$

and where T contains the following transitions for every $n, n' \in [0, m)$ and $b \in \{\text{false}, \text{true}\}$:

$$(n, n') \mapsto (n + n' \bmod m, n + n' \bmod m = c) \quad \text{and}$$
$$(n, b) \mapsto (n, n = c).$$

In the the full version of this paper[3] we show that $\mathcal{P}_{\text{rmd}}$ belongs to WS^3 by adapting the proof for $\mathcal{P}_{\text{thr}}$.

Negation and conjunction. Let $\mathcal{P}_1 = (Q_1, T_1, \Sigma, I_1, O_1)$ and $\mathcal{P}_2 = (Q_2, T_2, \Sigma, I_2, O_2)$ be WS^3-protocols computing predicates φ_1 and φ_2 respectively. We may assume that $\mathcal{P}_1$ and $\mathcal{P}_2$ are defined over identical Σ, for we can always extend the input domain of threshold/remainder predicates by variables with coefficients of value zero. The predicate $\neg\varphi_i$ can be computed by replacing O_i by the new output function O_i' such that $O_i'(q) \stackrel{\text{def}}{=} \neg O_i(q)$ for every $q \in Q_i$. To compute $\varphi_1 \wedge \varphi_2$, we build an asynchronous product where steps of $\mathcal{P}_1$ and $\mathcal{P}_2$ can be executed independently.

More formally, the *conjunction* of $\mathcal{P}_1$ and $\mathcal{P}_2$ is defined as the population protocol $\mathcal{P} \stackrel{\text{def}}{=} (Q, S, I, \Sigma, O)$ such that $Q \stackrel{\text{def}}{=} Q_1 \times Q_2$, $S \stackrel{\text{def}}{=} S_1 \cup S_2$, $I(\sigma) \stackrel{\text{def}}{=} (I_1(\sigma), I_2(\sigma))$ and $O(p,q) \stackrel{\text{def}}{=} O_1(p) \wedge O_2(q)$ where

$$S_1 \stackrel{\text{def}}{=} \{(p,r), (p',r') \mapsto (q,r), (q',r') : (p,p',q,q') \in T_1, r, r' \in Q_2\},$$
$$S_2 \stackrel{\text{def}}{=} \{(r,p), (r',p') \mapsto (r,q), (r',q') : (p,p',q,q') \in T_2, r, r' \in Q_1\}.$$

In the full version of this paper. we show that $\mathcal{P}$ is in WS^3 since terminal/consensus configurations, flow equations, and traps and siphons constraints are preserved by projections from $\mathcal{P}$ onto $\mathcal{P}_1$ and $\mathcal{P}_2$.

[3] https://arxiv.org/abs/1703.04367

Threshold			
$v_{\max}$	$\lvert Q\rvert$	$\lvert T\rvert$	Time
3	28	288	8.0
4	36	478	26.5
5	44	716	97.6
6	52	1002	243.4
7	60	1336	565.0
8	68	1718	1019.7
9	76	2148	2375.9
10	84	2626	timeout

Remainder			
m	$\lvert Q\rvert$	$\lvert T\rvert$	Time
10	12	65	0.4
20	22	230	2.8
30	32	495	15.9
40	42	860	79.3
50	52	1325	440.3
60	62	1890	3055.4
70	72	2555	3176.5
80	82	3320	timeout

Majority			
	$\lvert Q\rvert$	$\lvert T\rvert$	Time
	4	4	0.1

Flock of birds [6]			
c	$\lvert Q\rvert$	$\lvert T\rvert$	Time
20	21	210	1.5
25	26	325	3.3
30	31	465	7.7
35	36	630	20.8
40	41	820	106.9
45	46	1035	295.6
50	51	1275	181.6
55	56	1540	timeout

Broadcast			
	$\lvert Q\rvert$	$\lvert T\rvert$	Time
	2	1	0.1

Flock of birds [8]			
c	$\lvert Q\rvert$	$\lvert T\rvert$	Time
50	51	99	11.8
100	101	199	44.8
150	151	299	369.1
200	201	399	778.8
250	251	499	1554.2
300	301	599	2782.5
325	326	649	3470.8
350	351	699	timeout

Table 1: Results of the experimental evaluation where $|Q|$ denotes the number of states, $|T|$ denotes the number of non silent transitions, and the time to prove membership for WS^3 is given in seconds.

6 EXPERIMENTAL RESULTS

We have developed a tool called `Peregrine`[4] to check membership in WS^3. Peregrine is implemented on top of the SMT solver Z3 [9].

`Peregrine` reads in a population protocol $\mathcal{P} = (Q, T, \Sigma, I, O)$ and constructs two sets of constraints. The first set is satisfiable if and only if LayeredTermination holds, and the second is unsatisfiable if and only if StrongConsensus holds.

For LayeredTermination, our tool `Peregrine` iteratively constructs constraints checking the existence of an ordered partition of size 1,2, ..., $|T|$ and decides if they are satisfiable. To check that the execution of a layer is silent, the constraints mentioned in the proof of Proposition 4.3 are transformed using Farkas' lemma (see e.g. [23]) into a version that is satisfiable if and only if all the executions of the layer are silent. Also, the constraints for condition (b) of Definition 4.1 are added.

For StrongConsensus, `Peregrine` initially constructs the constraints for the flow equation for three configurations C_0, C_1, C_2 and vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, with additional constraints to guarantee that C_0 is initial, C_1 and C_2 are terminal, and C_1 and C_2 are consensus of different values. If these constraints are unsatisfiable, the protocol satisfies StrongConsensus. Otherwise, `Peregrine` searches for a U-trap or U-siphon to show that either $C_0 \xrightarrow{\boldsymbol{x}_1} C_1$ or $C_0 \xrightarrow{\boldsymbol{x}_2} C_2$ does not hold. If, say, a U-siphon S is found, then `Peregrine` adds the constraint $C_0(S) > 0$ to the set of initial constraints. This process is repeated until either the constraints are unsatisfiable and StrongConsensus is shown, or all possible U-traps and U-siphons are added, in which case StrongConsensus does not hold. We use this refinement-based approach instead of the coNP approach described in Proposition 4.12, as that could require a quadratic number of variables and constraints, and we generally expect to need a small number of refinement steps.

We evaluated `Peregrine` on a set of benchmarks: the threshold and remainder protocols of [2], the majority protocol of [3], the broadcast protocol of [8] and two versions of the flock of birds[5] protocol from [6, 8]. We checked the parametrized protocols for increasing values of their primary parameter until we reached a timeout. For the threshold and remainder protocols, we set the secondary parameter c to 1 since it has no incidence on the size of the protocol, and since the variation in execution time for different values of c was negligible. Moreover, we assumed that all possible values for a_i were present in the inputs, which represents the worst case.

All experiments were performed on the same machine equipped with an Intel Core i7-4810MQ CPU and 16 GB of RAM. The time limit was set to 1 hour. The results are shown in Table 1. In all cases where we terminated within the time limit, we were able to show membership for WS^3. Generally, showing StrongConsensus took much less time than showing LayeredTermination, except for the flock of birds protocols, where we needed linearly many U-traps.

As an extension, we also tried proving correctness after proving membership in WS^3. For this, we constructed constraints for the existence of an input X and configuration C with $I(X) \xrightarrow{\boldsymbol{x}} C$ and $\varphi(X) \neq O(C)$. We were able to prove correctness for all the protocols in our set of benchmarks. The correctness check was faster than the well-specification check for broadcast, majority, threshold and both flock of birds protocols, and slower for the remainder protocol, where we reached a timeout for $m = 70$.

7 CONCLUSION AND FURTHER WORK

We have presented WS^3, the first class of well-specified population protocols with a membership problem of reasonable complexity (i.e. in DP) and with the full expressiveness of well-specified protocols. Previous work had shown that the membership problem for the general class of well-specified protocols is decidable, but at least EXPSPACE-hard with algorithms of non primitive recursive complexity.

[4] `Peregrine` and benchmarks are available from https://gitlab.lrz.de/i7/peregrine/.

[5] The variant from [8] is referred to as *threshold-n* by its authors.

We have shown that WS^3 is a natural class that contains many standard protocols from the literature, like flock-of-birds, majority, threshold and remainder protocols. We implemented the membership procedure for WS^3 on top of the SMT solver Z3, yielding the first software able to automatically prove well-specification of population protocols for *all* (of the infinitely many) inputs. Previous work could only prove partial correctness of protocols with at most 9 states and 28 transitions, by trying exhaustively a *finite* number of inputs [6, 8, 20, 24]. Our algorithm deals with all inputs and can handle larger protocols with up to 70 states and over 2500 transitions.

Future work will concentrate on three problems: improving the performance of our tool; automatically deciding if a WS^3-protocol computes the predicate described by a given Presburger formula; and the diagnosis problem: when a protocol does not belong to WS^3, delivering an explanation, e.g. a non-terminating fair execution. We think that our constraint-based approach provides an excellent basis for attacking these questions.

ACKNOWLEDGMENTS

We wish to thank David de Frutos Escrig and Pierre Ganty for pointing out some mistakes in an earlier version of this paper.

REFERENCES

[1] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. 2004. Computation in networks of passively mobile finite-state sensors. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*. 290–299. https://doi.org/10.1145/1011767.1011810

[2] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. 2006. Computation in networks of passively mobile finite-state sensors. *Distributed Computing* 18, 4 (2006), 235–253. https://doi.org/10.1007/s00446-005-0138-3

[3] Dana Angluin, James Aspnes, and David Eisenstat. 2006. Stably computable predicates are semilinear. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. 292–299. https://doi.org/10.1145/1146381.1146425

[4] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. 2007. The computational power of population protocols. *Distributed Computing* 20, 4 (2007), 279–304. https://doi.org/10.1007/s00446-007-0040-2

[5] Konstantinos Athanasiou, Peizun Liu, and Thomas Wahl. 2016. Unbounded-Thread Program Verification using Thread-State Equations. In *IJCAR (Lecture Notes in Computer Science)*, Vol. 9706. Springer, 516–531.

[6] Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. 2010. Algorithmic Verification of Population Protocols. In *Proc. 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. 221–235.

[7] Ho-Lin Chen, Rachel Cummings, David Doty, and David Soloveichik. 2014. Speed Faults in Computation by Chemical Reaction Networks. In *DISC (Lecture Notes in Computer Science)*, Vol. 8784. Springer, 16–30.

[8] Julien Clément, Carole Delporte-Gallet, Hugues Fauconnier, and Mihaela Sighireanu. 2011. Guidelines for the Verification of Population Protocols. In *ICDCS*. IEEE Computer Society, 215–224.

[9] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS (Lecture Notes in Computer Science)*, Vol. 4963. Springer, 337–340. Z3 is available at https://github.com/Z3Prover/z3.

[10] Yuxin Deng and Jean-François Monin. 2009. Verifying Self-stabilizing Population Protocols with Coq. In *Proc. 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE)*. 201–208. https://doi.org/10.1109/TASE.2009.9

[11] Jörg Desel and Javier Esparza. 1995. *Free choice Petri nets*. Cambridge University Press.

[12] Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. 1999. Memory requirements for silent stabilization. *Acta Informatica* 36, 6 (1999), 447–462.

[13] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. 2015. Verification of Population Protocols. In *Proc. 26th International Conference on Concurrency Theory (CONCUR)*. 470–482.

[14] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. 2016. Model Checking Population Protocols. In *FSTTCS (LIPIcs)*, Vol. 65. 27:1–27:14.

[15] Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp J. Meyer, and Filip Niksic. 2014. An SMT-Based Approach to Coverability Analysis. In *CAV (Lecture Notes in Computer Science)*, Vol. 8559. Springer, 603–619.

[16] Javier Esparza and Stephan Melzer. 2000. Verification of Safety Properties Using Integer Programming: Beyond the State Equation. *Formal Methods in System Design* 16, 2 (2000), 159–189.

[17] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. 2011. Mediated population protocols. *Theor. Comput. Sci.* 412, 22 (2011), 2434–2450.

[18] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. 2012. Terminating Population Protocols via Some Minimal Global Knowledge Assumptions. In *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium, SSS 2012, Toronto, Canada, October 1-4, 2012. Proceedings*. 77–89.

[19] Saket Navlakha and Ziv Bar-Joseph. 2015. Distributed information processing in biological and computational systems. *Commun. ACM* 58, 1 (2015), 94–102. https://doi.org/10.1145/2678280

[20] Jun Pang, Zhengqin Luo, and Yuxin Deng. 2008. On Automatic Verification of Self-Stabilizing Population Protocols. In *Proc. 2nd IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering (TASE)*. 185–192. https://doi.org/10.1109/TASE.2008.8

[21] Christos H. Papadimitriou. 2007. *Computational complexity*. Academic Internet Publ.

[22] Sylvain Schmitz. 2016. The complexity of reachability in vector addition systems. *SIGLOG News* 3, 1 (2016), 4–21.

[23] Alexander Schrijver. 1986. *Theory of Linear and Integer Programming*. John Wiley & Sons.

[24] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. 2009. PAT: Towards Flexible Verification under Fairness. In *Proc. 21st International Conference on Computer Aided Verification (CAV)*. 709–714. https://doi.org/10.1007/978-3-642-02658-4_59

Appendix III

Note on Copyrights

According to the Consent to Publish in Lecture Notes in Computer Science with Springer-Verlag GmbH, the author of the thesis is allowed to include Paper D in the thesis:

> Author retains, in addition to uses permitted by law, the right to communicate the content of the Contribution to other scientists, to share the Contribution with them in manuscript form, to perform or present the Contribution or to use the content for non-commercial internal and educational purposes, provided the Springer publication is mentioned as the original source of publication in any printed or electronic materials. Author retains the right to republish the Contribution in any collection consisting solely of Author's own works without charge subject to ensuring that the publication by Springer is properly credited and that the relevant copyright notice is repeated verbatim.
>
> [...]
>
> Author retains the right to use his/her Contribution for his/her further scientific career by including the final published paper in his/her dissertation or doctoral thesis provided acknowledgment is given to the original source of publication.

For more information, please see the copyright form available under the link "LNCS Consent to Publish form" accessible at: https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines

According to the FMCAD Copyright Transfer Form to publish in the Proceedings of Formal Methods in Computer-Aided Design, FMCAD 2015, with FMCAD Inc., the author of the thesis is allowed to include Paper A in the thesis:

> Authors retain the same rights as FMCAD Inc. They can distribute their contributions on their Websites or in any manner they see fit, as long as a full citation to FMCAD 2015 is given, and a link to the FMCAD 2015 Website is provided. In addition, the Authors can create derivative works in connection with the Authors' scientific and academic activities. The Authors retain all proprietary rights in any process, procedure, or article of manufacture described in the Contribution.

The FMCAD 2015 Website is accessible at: http://www.cs.utexas.edu/users/hunt/FMCAD/FMCAD15/

According to the ACM Copyright Transfer Agreement and ACM Publishing License Agreement to publish in the Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, with the Association for Computing Machinery (ACM), the author of the thesis is allowed to include Paper E in the thesis:

> (b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM pursuant to Paragraph 2 (a), Owner shall have the right to do the following:
>
> (i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

For more information, please see the copyright and license forms available under the links "ACM Copyright Transfer Agreement" and "ACM Publishing License Agreement" accessible at: https://www.acm.org/publications/policies/copyright-and-license-forms

According to the Consent to Publish in Lecture Notes in Computer Science with Springer-Verlag GmbH and the Creative Commons Attribution License, the author of the thesis is allowed to include Paper B and C in the thesis:

> The Contribution is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, duplication, adaptation, distribution, and reproduction in any medium or format, as long as appropriate credit is given to the original author(s) and the source, a link is provided to the Creative Commons license, and any changes made are indicated.

According to the information by the IEEE RightsLink Service for Thesis / Dissertation Reuse, the author of the thesis is allowed to include the quote by Leslie Lamport at the beginning of Chapter 1:

> Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:
>
> 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © [Year of original publication] IEEE.

For more information, follow the instructions in https://www.ieee.org/publications/rights/rights-link.html for the article [Lam77].