A polyhedral framework for reachability problems in Petri Nets

Nicolas Amat

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Institut National des Sciences Appliquées de Toulouse

Présentée et soutenue par

Nicolas AMAT

Le 4 décembre 2023

A Polyhedral Framework for Reachability Problems in Petri Nets

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
François VERNADAT et Didier LE BOTLAN

Jury

Mme Laure PETRUCCI, Rapporteure
M. Igor WALUKIEWICZ, Rapporteur
Mme Béatrice BÉRARD, Examinatrice
M. Fabrice KORDON, Examinateur
M. François VERNADAT, Directeur de thèse
M. Didier LE BOTLAN, Co-directeur de thèse
M. Silvano DAL ZILIO, Co-encadrant de thèse
M. Loïc HÉLOUËT, Président

To those who have accompanied me.

Three years already.

It is all yours.

Remerciements

Après trois belles années à travailler sur ce sujet de thèse, je vous livre dans ce manuscrit ma vision d'un cadre polyédrique pour les problèmes d'accessibilité dans les réseaux de Petri, et j'emporte avec moi tous les bons souvenirs.

Mes premiers remerciements vont à mes encadrants, sans qui ce travail n'aurait été mené à bien. Merci à François Vernadat. En plus d'avoir partagé un bureau, on aura partagé des idées, des débats, mais surtout des moments de vie agréables. Merci pour les conseils, la sagesse et l'amitié ; je ressors grandi de notre rencontre. Merci à Didier Le Botlan, pour son aide infaillible, sa passion, ses idées et sa virtuosité qui m'aura sorti plus d'une fois de moments périlleux. Il ne finira jamais de m'étonner. Et pour finir, je tiens à remercier Silvano Dal Zilio, mon respect est immense. Merci de m'avoir proposé ce sujet et de m'avoir enseigné le métier de chercheur. Sa Science, son humour, son humilité, ainsi que sa confiance et sa bienveillance envers moi, sont tant de choses pour lesquels je le remercie. Have fun.

Je remercie chaleureusement mes rapporteurs, Laure Petrucci et Igor Walukiewicz, qui m'ont fait l'honneur d'évaluer mes travaux. Un grand merci également aux autres membres du jury, Béatrice Bérard, Loïc Hélouët et Fabrice Kordon, d'avoir accepté de siéger à ma soutenance de thèse. Merci à vous tous, pour les échanges et les retours.

Je suis convaincu que je n'aurais pas envisagé de réaliser une thèse en informatique théorique si je n'avais pas croisé à l'ENSIMAG la route d'enseignants qui m'ont fait préférer les théorèmes aux challenges RootMe. Merci à Sophie Quinton pour ses TDs de calculabilité. Merci à Mnacho Echenim et Nicolas Peltier de m'avoir initié à la logique ; je garde un souvenir particulier de notre formalisation de la logique de séparation en Isabelle.

Je tiens à remercier Jérôme Leroux et Yann Thierry-Mieg, pour avoir toujours répondu à mes mails, pour m'avoir ouvert la porte de leurs bureaux et tout simplement pour m'avoir inspiré dans mes travaux. Merci également aux organisateurs du Model Checking Contest, en particulier à Fabrice Kordon pour sa patience et sa disponibilité ; ce fut un plaisir de prendre part à ce terrain de jeu.

Merci à Pierre Bouvier et Hubert Garavel de INRIA Grenoble pour notre fructueuse collaboration sur le problème des places concurrentes. Ce sujet m'a amené à introduire les Token Flow

Graphs qui, par la suite, ont été essentiels à l'élimination de quantificateurs. Certaines choses ne tiennent parfois qu'à une rencontre.

En parlant de rencontres, merci à l'équipe VERTICS du LAAS-CNRS, ma terre d'accueil durant ces trois ans. Merci à Bernard Berthomieu d'avoir été un mentor au Model Checking Contest. Certains lui sont reconnaissants pour les graphes de classes, d'autres pour le développement de TINA, moi je lui suis pour son idée de réduction polyédrique, sans laquelle mon sujet n'aurait pas existé. Mes remerciements vont également à Tomasz Kloda pour avoir égayé ma dernière année au bureau et pour sa compagnie parfois tard le soir. Je n'oublie pas Thomas Hujsa, de passage dans l'équipe, avec qui j'ai gardé une belle amitié.

Un grand merci à mes anciens collègues du LAAS. Merci aux (post-)doctorants que j'ai pu côtoyer : Alexandre, Amaury, Éric, Ibis, Léonie et Rafael, merci pour les moments de détente. Une mention spéciale pour Camille, qui m'a accompagné dans la rédaction de thèse. J'ai hâte que tu me présentes tes nombreuses nouvelles idées, bonne route à toi l'ami. Merci également à ceux qui m'ont rendu le travail plus facile au quotidien, en particulier à David Gauchard et Thibault Hueber pour leur gentillesse et leur disponibilité. Et merci à tous ceux qui m'ont fait confiance et avec qui j'ai pu enseigner dans la bonne humeur : Philippe Esteban, Emmanuel Hebrard, Didier Le Botlan, Euriell Le Corronc et Pauline Ribot.

Comme chaque départ implique un lieu d'arrivé, je tiens à remercier Pierre Ganty et Alessio Mansutti pour leur accueil à l'IMDEA Software Institute à Madrid.

Un grand merci à mes amis, avec qui j'ai partagé des moments de joie. Merci au crew TMC31 pour les semaines de surf, les week-ends de snowboard et les soirées au skatepark. Plus de dix ans, et même si le temps nous éloigne, chaque fois, c'est comme si rien n'avait jamais changé. Une pensée forte à Arno, tu nous manques. Merci à mes amis de l'IMAG en exil à Toulouse. Baptiste, pour avoir rendu la thèse moins solitaire, pour son soutien et pour les sorties sportives. Théo, pour m'avoir supporté toutes ces années sur les bancs de l'école. Et Philémon, pour tous ces souvenirs à coups de chaîne à vélo.

Pour terminer merci à ma famille, de m'avoir permis de vivre mes rêves. À ma grande sœur, qui m'a toujours rendu fier et qui a ouvert la voie de l'informatique. Merci à mes parents, de m'avoir enseigné de courir après le bonheur et d'avoir tout mis en œuvre pour m'y aider. Le petit garçon qui pianotait dans sa chambre sur son clavier d'ordinateur réalise aujourd'hui la chance qu'il avait. Merci à Eva de partager ma vie. Merci de m'encourager à continuer de rêver et de me m'accompagner dans cette nouvelle aventure à Madrid ; je suis certain que demain sera aussi beau qu'hier ...

Abstract

We propose and study a method to accelerate the verification of reachability problems in Petri nets based on structural reductions. This approach, that we call polyhedral reduction, relies on a state space abstraction that combines structural reductions and linear arithmetic constraints on the marking of places.

The correctness of this method is based on a new notion of equivalence between nets. Combined with an SMT-based model checker, one can transform a reachability problem about some Petri net, into the verification of an equivalent reachability property on a reduced version of this net. We also propose an automated procedure to prove that such an abstraction is correct, exploiting a connection with a class of Petri nets with Presburger-definable reachability sets.

In addition, we present a data structure, called Token Flow Graph (TFG), that captures the particular structure of constraints stemming from structural reductions. We leverage TFGs to efficiently solve two problems. First, to eliminate quantifiers that appear during our transformation, in the updated formula to be checked on the reduced net. Second, to compute the concurrency relation of a net, that is all pairs of places that can be marked simultaneously in some reachable marking.

We apply our approach to several symbolic model checking procedures and introduce a new semi-decision procedure for checking reachability properties in Petri nets based on the Property Directed Reachability (PDR) method. A distinctive feature of this PDR method is its ability to generate verdict certificates that can be verified using an external SMT solver.

Our approach and algorithms are implemented in four open-source tools: SMPT for checking reachability properties; Kong for accelerating the computation of concurrent places; Octant for eliminating quantifiers; and Reductron for automatically proving the correctness of polyhedral reductions. We give experimental results about their effectiveness, both for bounded and unbounded nets, using a large benchmark provided by the Model Checking Contest. We focus on the reproducibility of our results and provide an accompanying artifact that covers all our experiments.

Keywords: **Model checking** **Reachability problems** **Petri nets** **Structural reductions** **Abstraction techniques** **SMT solving**

Résumé

Nous proposons une méthode, appelée réduction polyédrique, pour accélérer la vérification de problèmes d'accessibilité sur les réseaux de Petri basée sur des réductions structurelles. Notre approche repose sur une abstraction de l'espace d'états qui combine réductions structurelles et contraintes arithmétiques sur le marquage des places.

La correction de cette méthode est basée sur une nouvelle notion d'équivalence entre réseaux. Combinée avec un model checker basé sur des méthodes SMT, nous montrons comment transformer un problème d'accessibilité sur un réseau de Petri, en la vérification d'une propriété équivalente sur une version réduite de ce réseau. Nous proposons également une procédure automatique pour prouver qu'une telle abstraction est correcte, en exploitant une connexion avec une classe de réseaux de Petri qui ont un ensemble d'accessibilité définissable par l'arithmétique de Presburger.

De plus, nous présentons une nouvelle structure de données, appelée Token Flow Graph (TFG), qui capture la structure particulière des contraintes résultant des réductions structurelles. Nous exploitons les TFGs pour résoudre efficacement deux problèmes. Premièrement, pour éliminer les quantificateurs, qui apparaissent lors de notre transformation, dans la formule mise à jour à vérifier sur le réseau réduit. Deuxièmement, pour le calcul de la relation de concurrence d'un réseau, c'est-à-dire énumérer toutes les paires de places qui peuvent être marquées simultanément dans un marquage accessible.

Nous appliquons notre approche à plusieurs procédures de vérification symboliques et nous introduisons une nouvelle procédure de semi-décision pour la vérification des propriétés d'accessibilité sur les réseaux de Petri, basée sur la méthode Property Directed Reachability (PDR). La particularité de cette méthode PDR réside dans sa capacité à générer des certificats de verdict qui peuvent être vérifiés à l'aide d'un solveur SMT externe.

Notre approche et nos algorithmes sont implémentés dans quatre outils open source : SMPT pour vérifier des propriétés d'accessibilité ; Kong pour accélérer le calcul de places concurrentes ; Octant pour l'élimination de quantificateurs ; et enfin Reductron pour prouver automatiquement la correction de réductions polyédriques. Nous donnons des résultats expérimentaux sur leur efficacité, à la fois pour des réseaux bornés et non bornés, en utilisant les modèles et formules fournis par le Model Checking Contest. Nous mettons l'accent sur la reproductibilité de nos résultats et fournissons un artefact couvrant l'ensemble de nos expérimentations.

Mots clés : Model checking **Problèmes d'accessibilité** **Réseaux de Petri**
Réductions structurelles **Techniques d'abstraction** **Résolution SMT**

Table of Contents

List of Figures

List of Tables

List of Algorithms

Introduction

> Computer Science is no more about computers than astronomy is about telescopes.
>
> Edsger Wybe Dijkstra

In this thesis, we address the problem of checking *reachability properties*, that is to find whether a given state—or a class of states—can be reached by the model of a system. Reachability is a fundamental and difficult problem, with many practical applications. It obviously plays an important role in the formal verification of concurrent systems, for instance, for testing the absence of "bad states", such as deadlocks, or at the opposite, for testing that an invariant is preserved. It can be used to study diverse kinds of systems, such as software systems [GS92; BE12; KKW14], distributed systems [Cos+10], biological systems [Bal+10], business workflows [Aal15], etc.

The question we study is a subproblem of *model checking*, a formal verification technique introduced concurrently in the eighties by Emerson and Clarke [EC80], in the US, and Queille and Sifakis [QS82], in France. Model checking defines a set of algorithmic methods for determining whether an abstract model satisfies a formal specification, generally expressed as a temporal logic formula [CGP99; BK08]. Model checking also often entails that, when a property is not satisfied, then the method should exhibit a counter-example, that is, a witness execution that shows the source of the problem.

In this context, a *model* defines a behavior abstraction of the system and a *specification* describes its expected properties (in other words what the system should do and the characteristics it should have). In our work, we will focus on models expressed using Petri nets and on specifications reduced to *reachability properties*; such properties are also called *state formulas* since their truth value only depends on the state of its evaluation.

Choice of Petri Nets as Formal Model. Petri nets (see, for example, [Rei12]) is one of the earliest models proposed for the study of concurrency theory. In this respect, it can be compared with other formalisms with a similar goal, such as: *automata-based approaches* (like Kahn's networks [Kah74], Nivat/Arnold's networks of automata [Arn02], or Lynch's I/O

automaton [LT88]); *process calculi*, also called *process algebras* (such as the Communicating Sequential Processes calculus (CSP) of Hoare [Hoa78] or the Calculus of Communicating Systems (CCS) of Milner [Mil80]).

The choice of a particular formalism is not necessarily the dominant factor in model checking. Indeed, it is often possible to translate or interpret a model from one formalism to another. We use such examples in our experiments, where many Petri nets have been obtained by translation from an original LOTOS model [BB87; ISO89; GS90]. See also [Bas98] for a comparison between Petri nets and process algebra, for system design, and [BDK96] for methods to translate models between the two.

But each theory also comes with its idiosyncrasies; its own distinctive features and verification approaches. Typically, Petri net theory considers states and transitions as "equal" (and even often as dual notions), whereas process algebra, for instance, tries to abstract away the notion of state, and rather focuses on behavioral equivalences.

Another key aspect of Petri net theory is to largely rely on linear and integer programming techniques, with specific concepts such as the *state equation*, or the use of *structural invariants* for instance. As a matter of fact, we will make good use of integer arithmetic theory in our work.

Finally, some Petri nets may have an unbounded number of reachable states; which means that, in such cases, the reachability problem provides an interesting use case for the analysis of infinite-state systems. The methods that we define in our work are, in most cases, suitable for unbounded nets. Infinite-state systems are usually not the norm with other formalisms. Notable exceptions include extensions of automata with counters or stacks; process calculi with data or operators for recursive definition (see, for instance, [BPS01]); or higher-order formalisms that center around the dynamic generation of new processes, or new communication channels, like in Milner's π-calculus [MPW92].

Reachability and Symbolic Techniques. The reachability problem for Petri nets has been extensively studied, both from a theoretical and a practical point of view, with many tools implementing specific reachability methods. One of the most celebrated results of Petri net theory is that the reachability problem is decidable, which was first stated by Mayr in 1981 [May81]. It is pretty remarkable that we had to wait another 40 years before obtaining a complete characterization of this problem's complexity. Indeed, it was not until 2022 that Leroux and Czerwiński independently settled [CO22; Ler22] that the problem is Ackermann-complete, and therefore inherently more complex than, say, the coverability problem, which is EXPSPACE-complete [Esp98].

A consequence of the "inherently high complexity" of the reachability problem for Petri nets, and a general consensus among the Petri net community, is that we should not expect to find a one-size-fits-all algorithm for solving it. A better strategy is to try to improve the performances in some cases—for example by developing new tools, or optimizations, that may

perform better on some examples—or try to enlarge the class of handled models (out-the-scope of current techniques).

An approach that has proved quite successful in this respect is to avoid exhaustive, state-enumeration, techniques and rather rely on a symbolic representation of the state space (like with decision diagrams [Bur+92]), or an abstraction of the problem, for instance with the use of logical approaches, like SAT and SMT (Satisfiability Modulo Theory) solving. Furthermore, we can also benefit from optimizations related to the underlying model. For instance, when analyzing Petri nets, we can make use of techniques for decomposing and simplifying nets, a method pioneered by Berthelot [Ber87], known as *structural reduction*.

Structural Reductions for Reachability. Berthomieu recently proposed a new abstraction technique (what we will call *polyhedral reduction* in this thesis), based on structural reductions [BLD18; BLD19]. The idea is to compute reductions of the form (N, E, N'), where: N is an initial net (that we want to analyze); N' is a residual net (hopefully much simpler than N); and E is a predicate of linear constraints. The idea is to preserve enough information in E so that we can rebuild the reachable markings of N knowing only the ones of N'. In a nutshell, we capture and abstract the effect of reductions using linear constraints between the places of N and N'.

This technique has been previously applied in a symbolic model checker, called Tedd, that uses Set Decision Diagrams [Thi+09] in order to generate an abstract representation for the state space of a net N. In practice, an initial Petri net N with n places, can often be reduced to a residual net, N', with far fewer places, say n'. Hence, this approach makes it possible to represent the state space of N, which is a subset of integer vectors of dimension n, as the "inverse image", by the linear system E, of a subset of vectors of dimension n'. This technique can result in a very compact representation of the state space. This effect has been observed in practice, during the recent editions of the Model Checking Contest (MCC) [Amp+19], where Tedd finished at first place for five consecutive years in the *state space* category.

The goal of this thesis is to study how a similar technique could be applied together with SMT solvers, and in the context of the reachability problem, rather than for state space generation. As a result, we propose new automatic verification techniques combining model checking with methods from the fields of convex analysis and automated reasoning. We also address related issues: how to formalize our approach (we define a new notion of marking equivalence); how to efficiently integrate this approach with existing tools (we define an automatic "projection" method that works on both models and formulas); and how to add trust in our verification framework (we define a method that can generate verdict certificates and also define an automated method for proving the soundness of new reduction rules).

I High-Level Description of My Contributions

In this thesis, I propose and study a method for *accelerating* the verification of reachability problems in Petri nets, based on structural reductions, that I called *polyhedral reduction*. The approach relies on a state space abstraction that combines structural reductions and linear arithmetic constraints on the marking of places.

The correctness of this method is based on a new notion of behavioral equivalence between nets that I define in Chapter 3. Combined with an SMT-based model checker, one can transform a reachability problem about some Petri net into the verification of an updated reachability property on a reduced version of this net (Chapters 3 and 5). I also propose in Chapter 7 an automated procedure to prove that such an abstraction is correct, exploiting a connection with a known class of Petri nets with Presburger-definable reachability sets [Ler13].

In addition, I present in Chapter 4 a data structure, called Token Flow Graph (TFG), that captures the particular structure of constraints stemming from structural reductions. I leverage TFGs to solve two problems efficiently. First, in Chapter 5 to eliminate quantifiers that appear during our transformation in the updated formula to be checked on the reduced net. Second, in Chapter 6 to compute the concurrency relation of a net, i.e., all pairs of places that can be marked simultaneously in some reachable marking. This work led to a collaboration with the Convecs team at INRIA Grenoble [ABG24].

I apply my approach to several symbolic model checking procedures. A result of this work is the definition of a new semi-decision procedure for checking reachability properties on Petri nets based on the Property Directed Reachability (PDR) method; see Chapter 2. A distinctive feature of this PDR method is its ability to generate verdict certificates in Presburger arithmetic that can be verified using an external SMT solver [Ler10].

This thesis describes my efforts to conduct research that combines theoretical advances with concrete implementations. I have implemented my approach and algorithms in four open-source tools described in Chapter 8: SMPT for checking reachability properties; Kong for accelerating the computation of concurrent places; Octant for eliminating quantifiers; and Reductron for automatically proving the correctness of polyhedral reductions. I have also studied their effectiveness in extensive experimental evaluations, both for bounded and unbounded nets, using a significant benchmark provided by the Model Checking Contest [Kor+23], an international competition for model checking tools. This has also led me to take part in the *reachability* category of the three last editions of the Model Checking Contest. My tool, SMPT, has obtained the bronze medal during the last two editions.

II Open Science and Reproducibility

I would now like to turn my attention to an institutional aspect. Over the last three years, as a young scientist, I have become aware of the importance of an open and reproducible

science. Obtaining papers, and experimenting with other methods or tools have often been a difficult and tiring task. I wanted this thesis to be part of a movement to liberate science. This is why I provide an artifact accompanying this work, which should ease the process of getting a working environment to use my tools and run the experiments included in this thesis.

I describe below some tenets that I tried to follow when producing this work.

Making papers accessible. I uploaded my articles to the HAL and arXiv platforms. This gives everyone free access to my work, with no restrictions on access to publishers. In addition, I was able to provide the full proof of my results in appendices, which is often not possible with the page restriction in conference papers. I would also like to commend the philosophy of EpiSciences journals, such as Fundamenta Informaticae in which I published a paper, and which enforces the publication on HAL or arXiv in its submission process.

Experimenting on accessible benchmarks. I performed my experiments on an accessible and peer-reviewed benchmark, that is mainly composed of models and formulas used during the Model Checking Contest. When I developed new instances, I submitted them to the contest organizers:

[**ADH23a**] N. Amat, S. Dal Zilio, and T. Hujsa. *Model entitled "CryptoMiner" proposed for the Model Checking Contest*. 2023. URL: https://mcc.lip6.fr/2023/pdf/CryptoMiner-form.pdf (visited on 10/10/2023)

[**ADH23b**] N. Amat, S. Dal Zilio, and T. Hujsa. *Model entitled "Murphy" proposed for the Model Checking Contest*. 2023. URL: https://mcc.lip6.fr/2023/pdf/Murphy-form.pdf (visited on 10/10/2023)

[**ADH23c**] N. Amat, S. Dal Zilio, and T. Hujsa. *Model entitled "PGCD" proposed for the Model Checking Contest*. 2023. URL: https://mcc.lip6.fr/2023/pdf/PGCD-form.pdf (visited on 10/10/2023)

In addition, I also contributed to other benchmark suites, such as the SMT-LIB repository used in the SMT-COMP, a competition for SMT solvers:

[**ADH23a**] N. Amat. *A QF-LIA Benchmark Suite from Polyhedral Reductions of Petri Nets*. Research report. LAAS-CNRS, 2023

Producing available tools and artifacts. All my tools are open-source under a GPLv3 license. I added labels to the GitHub repositories corresponding to the versions used in my papers when experimenting. The following is a list of my tools related to this thesis:

[**Ama20b**] N. Amat. *SMPT: The Satisfiability Modulo Petri Nets Model Checker. An SMT-based model checker for Petri nets focused on reachability problems that takes advantage*

of polyhedral reduction. 2020. URL: https://github.com/nicolasAmat/SMPT (visited on 10/10/2023)

[**Ama20a**] N. Amat. *Kong: The Koncurrent Places Grinder. A tool to accelerate the computation of the concurrency relation of a Petri net using polyhedral reduction.* 2020. URL: https://github.com/nicolasAmat/Kong (visited on 10/10/2023)

[**Ama23c**] N. Amat. *Octant: The Reachability Formula Projector. A tool to project Petri net reachability properties on reduced nets using polyhedral reduction.* 2023. URL: https://github.com/nicolasAmat/Octant (visited on 10/10/2023)

[**Ama23d**] N. Amat. *Reductron: The Polyhedral Abstraction Prover. A tool to automatically prove the correctness of polyhedral equivalences for Petri nets.* 2023. URL: https://github.com/nicolasAmat/Reductron (visited on 10/10/2023)

Although making its code free is valuable for open science, I think it is not a necessary condition for reproducibility. What really counts, in my opinion, is to leave previous versions of tools easily accessible. This allows the community to compare approaches in the best way. In this idea of making my tools easily accessible to a certain version, I also provided three artifacts, for conferences having an artifact evaluation in their submission process (TACAS, FM and VMCAI):

[**ADH22a**] N. Amat, S. Dal Zilio, and T. Hujsa. *Artifact for TACAS 2022 Paper: Property Directed Reachability for Generalized Petri Nets.* Zenodo, 2022. DOI: 10.5281/zenodo.5863379

[**AD22**] N. Amat and S. Dal Zilio. *Artifact for FM 2023 Paper: SMPT: A Testbed for Reachability Methods in Generalized Petri Nets.* Zenodo, 2022. DOI: 10.5281/zenodo.7341426

[**ADL23a**] N. Amat, S. Dal Zilio, and D. Le Botlan. *Artifact for VMCAI 2024 Paper "Project and Conquer: Fast Quantifier Elimination for Checking Petri Net Reachability".* Zenodo, 2023. DOI: 10.5281/zenodo.10061156

As mentioned previously, this thesis is also accompanied of an artifact, permitting to reproduce all my experimental work:

[**Ama23b**] N. Amat. *Artifact for PhD thesis: "A polyhedral framework for reachability problems in Petri nets".* Zenodo, 2023. DOI: 10.5281/zenodo.8349546

Participating in competitions. I participated in the three last editions of the Model Checking Contest, or MCC for short, in the reachability category. My main takeaway is that competition is not just about competing. Of course, this makes it possible to compare different approaches, which is of immediate scientific interest. However, it also makes it possible to improve the reliability of tools; to obtain a common format for nets and formulas; and to provide (like artifacts) “ready-to-use” disk images for users. But above all, this experience

has been scientifically enriching for me. It gave me the opportunity to exchange ideas with competitors and allowed me to transform my model checker, SMPT, from a prototype to a tool that can be reused by others. It is partly due to this competition that many of the results of my work are partly "tool-oriented", rather than only theoretical.

The results of my three participations (discussed in Chapter 8) can be found on the MCC website:

[**Kor+21a**] F. Kordon et al. *Complete Results for the 2021 Edition of the Model Checking Contest*. 2021. URL: http://mcc.lip6.fr/2021/results.php (visited on 10/10/2023)

[**Kor+22**] F. Kordon et al. *Complete Results for the 2022 Edition of the Model Checking Contest*. 2022. URL: http://mcc.lip6.fr/2022/results.php (visited on 10/10/2023)

[**Kor+23**] F. Kordon et al. *Complete Results for the 2023 Edition of the Model Checking Contest*. 2023. URL: https://mcc.lip6.fr/2023/results.php (visited on 10/10/2023)

III Outline

We give a brief summary of the content of each chapter in this document. To get a clearer picture, we describe in Fig. 1 the dependencies between chapters and their relations to our tools (depicted on the left). Each chapter starts with a short abstract; and an introduction describing the context, its challenge and our proposal. Even though we provide a general overview of the related work in the next chapter, we give a specific state-of-the-art description at the end of each when discussing the contributions.

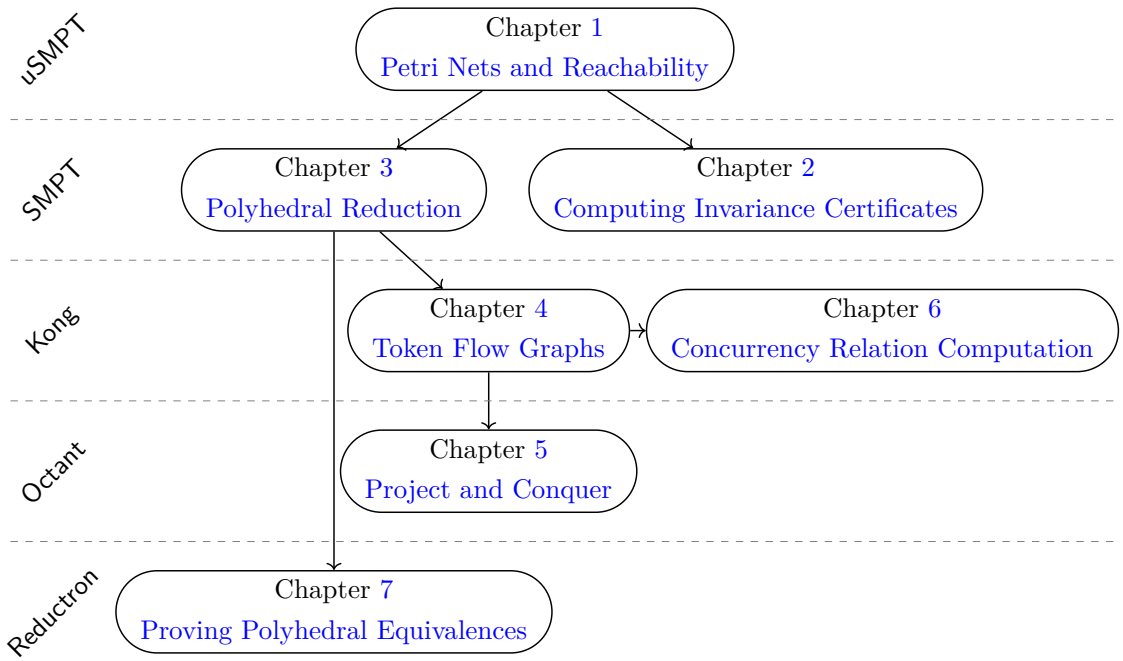


Fig. 1 Chapter and tool dependency graph.

Chapter 1 Petri Nets and Reachability: This chapter introduces the definitions and notations used throughout the thesis. It also includes a general overview of related work, which we compare with the contributions of this thesis. We rely on a presentation of Petri net semantics that emphasizes the relation to Presburger arithmetic. We also choose, as much as possible, to follow a syntax that corresponds to the theory for "quantifier-free linear integer arithmetic", used in SMT solvers.

A contribution to this chapter is an educational project:

[**Ama23e**] N. Amat. *uSMPT: an educational project, targeting Master and PhD students to showcase the application of SMT methods in system verification, by developing a Petri net model checker for the reachability problem.* 2023. URL: https://github.com/nicolasAmat/uSMPT (visited on 10/10/2023)

Chapter 2 Computing Invariance Certificates: In this chapter, we propose a semi-decision procedure for checking reachability properties on Petri nets that is based on the Property Directed Reachability (PDR) method. We present three different versions that vary depending on the method used for abstracting possible witnesses. We have implemented our methods in our model checker SMPT and give empirical evidences that our approach can handle problems that are difficult or even impossible to check with current state-of-the-art tools.

Related publication:

[**ADH22b**] N. Amat, S. Dal Zilio, and T. Hujsa. "Property Directed Reachability for Generalized Petri Nets". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. vol. 13243. Lecture Notes in Computer Science. Springer, 2022. DOI: 10.1007/978-3-030-99524-9_28

Chapter 3 Polyhedral Reduction: This is in this chapter that we define a method for taking advantage of polyhedral reduction in combination with an SMT-based model checker. The approach consists in transforming a reachability problem about some Petri net into the verification of an updated reachability property on a reduced version of this net. We prove the correctness of this method using a new notion of equivalence between nets, called *polyhedral abstraction equivalence*. We provide a complete framework to define and check the correctness of equivalence judgments, prove that this relation is a congruence, and give examples of basic equivalence relations that derive from structural reductions. This framework has also been implemented in the tool SMPT. As a testbed, we propose an adaptation of the Bounded Model Checking (BMC) method. Experimental results show that our approach works well, even when

we only have a moderate amount of reductions.

Related publications:

[**ABD21**] N. Amat, B. Berthomieu, and S. Dal Zilio. "On the Combination of Polyhedral Abstraction and SMT-Based Model Checking for Petri Nets". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. vol. 12734. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-76983-3_9

[**ABD22**] N. Amat, B. Berthomieu, and S. Dal Zilio. "A Polyhedral Abstraction for Petri Nets and its Application to SMT-Based Model Checking". In: *Fundamenta Informaticae* 187.2-4 (2022), pp. 103–138. DOI: 10.3233/FI-222134

Chapter 4 Token Flow Graphs: The objective of this chapter is to propose a data structure, called a Token Flow Graph (TFG) that captures the particular structure of constraints stemming from polyhedral reductions. To illustrate the use of this new data structure, we propose to accelerate the reachability check of a given marking. The approach is implemented in a tool, called Kong.

Related publications:

[**ADL21**] N. Amat, S. Dal Zilio, and D. Le Botlan. "Accelerating the Computation of Dead and Concurrent Places Using Reductions". In: *Model Checking Software (SPIN)*. vol. 12864. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-84629-9_3.

[**ADL23c**] N. Amat, S. Dal Zilio, and D. Le Botlan. "Leveraging polyhedral reductions for solving Petri net reachability problems". In: *International Journal on Software Tools for Technology Transfer* 25.1 (2023), pp. 95–114. DOI: 10.1007/s10009-022-00694-8;

Chapter 5 Project and Conquer: As a continuation of Chapter 3, we propose a method for checking reachability properties on Petri nets that takes advantage of polyhedral reductions, but this time that can be used transparently, as a preprocessing step of existing model checkers. The approach is based on a new procedure that can project a reachability property, about an initial Petri net, into an equivalent formula that only refers to the reduced version of this net. In particular, the projection is defined as a quantifier elimination procedure for Presburger arithmetic tailored to the specific kind of constraints we handle in Token Flow Graphs (defined in Chapter 4). The procedure is implemented in a tool, called Octant.

Related publication:

[**ADL24**] N. Amat, S. Dal Zilio, and D. Le Botlan. "Project and Conquer: Fast Quantifier Elimination for Checking Petri Nets Reachability". In: *Verification, Model Checking, and*

Abstract Interpretation (VMCAI). Lecture Notes in Computer Science. Springer, 2024. DOI: 10.1007/978-3-031-50524-9_5.

Chapter 6 Concurrency Relation Computation: In this chapter, we also leverage Token Flow Graphs to efficiently compute the concurrency relation of a net, a fundamental problem for computing decompositions into networks of automata. It consists of enumerating all pairs of places that can be marked simultaneously in some reachable marking. The "acceleration" algorithm is also implemented in the tool Kong.

Related publications are the same as those for Chapter 4.

Chapter 7 Proving Polyhedral Equivalences: This chapter contains the last theoretical contributions. We propose an automated procedure to prove the correctness of some polyhedral reduction. Our approach relies on an encoding into a set of SMT formulas whose satisfaction implies that the equivalence holds. For completeness, we exploit a connection with a class of Petri nets that have Presburger-definable reachability sets. We have implemented our procedure in a tool, called Reductron, and we illustrate its use in several examples.

Related publication:

[ADL23b] N. Amat, S. Dal Zilio, and D. Le Botlan. "Automated Polyhedral Abstraction Proving". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. vol. 13929. Lecture Notes in Computer Science. Springer, 2023. DOI: 10.1007/978-3-031-33620-1_18.

Chapter 8 Tools and Reproducibility: This chapter is the experimental counterpart of the previous chapters, which are more theoretical in nature. We give a thorough presentation of our tools, the benchmark suite from the Model Checking Contest used in our experiments, and explain how to reproduce the whole experiments using the accompanying artifact. It is also an opportunity to take stock of our three participations in the Model Checking Contest.

Related publications:

[AC22] N. Amat and L. Chauvet. "Kong: a Tool to Squash Concurrent Places". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. vol. 13288. Springer, 2022. DOI: 10.1007/978-3-031-06653-5_6.

[AD23] N. Amat and S. Dal Zilio. "SMPT: A Testbed for Reachabilty Methods in Generalized Petri Nets". In: *Formal Methods (FM)*. vol. 14000. Lecture Notes in Computer Science. Springer, 2023. DOI: 10.1007/978-3-031-27481-7_25;

[**ABG24**] N. Amat, P. Bouvier, and H. Garavel. "A Toolchain to Compute Concurrent Places of Petri Nets". In: *Transactions on Petri Nets and Other Models of Concurrency XVII*. Lecture Notes in Computer Science 14150 (2024), pp. 1–26. DOI: 10.1007/978-3-662-68191-6_1;

The **artifact** accompanying this thesis is freely available on Zenodo:

https://doi.org/10.5281/zenodo.8349546

Chapter 1

Petri Nets and Reachability

Definitions, Related Work and Comparison

> Computer Science is a science of abstraction—creating the right model for a problem and devising the appropriate mechanizable techniques to solve it.
>
> Alfred Aho

This chapter introduces the definitions and notations used throughout the text. It also includes a general overview of related work. We rely on a presentation of Petri net semantics (Sect. 1.1) that emphasizes the relation to Presburger Arithmetic (PA) (Sect. 1.2). We also choose, as much as possible, to follow a syntax that corresponds to the theory for "Quantifier-Free Linear Integer Arithmetic" (QF-LIA), used in SMT solvers, which translates to the quantifier-free fragment of Presburger Arithmetic (∃PA). We will sometimes use universal quantification in our formulas, but we are careful to point out this fact when this is the case.

The rest of the chapter is as follows. We describe our main research goal, the *generalized reachability problem*, in Sect. 1.3, then in Section 1.4, we list some known theoretical results associated with it. Since our work is partly tool-oriented, we also use this chapter as an opportunity to give an overview of some state-of-the-art tools solving the reachability problem for Petri nets and describe the methods that they implement (Sects. 1.5 and 1.6). Finally, a comparison with the thesis contributions is made in Sect. 1.7.

We believe that our presentation of Petri net semantics is better tailored than more classical approaches, for instance [Mur89], when describing our results. The choice of a language very close to the input format of SMT solvers is also motivated by our goal to provide an actual implementation for all our methods. As a result, the formulas that we define in this work are often the exact equivalent of the SMT-LIB script generated by our tools. One of my original

contributions in this context is a sandbox Petri net model checker for the reachability problem, called uSMPT (https://github.com/nicolasAmat/uSMPT) (a "micro"-version of my full-fledged tool SMPT), targeted at teaching SMT-based symbolic methods to postgraduate students. Formulas generated with uSMPT correspond exactly to the definitions that can be found in our presentation of the Petri net semantics and some of the methods described in Sect. 1.5.

Note. We made the choice to consider specifically related work on the Petri net reachability problem. This is why we do not discuss other formal methods and their associated techniques, or provide a large overview of different formal verification methods and their benefits in the development of complex systems. Some works, e.g., [Gar12], do it much better than what we could pretend to achieve in an introductory chapter.

1.1 Petri Nets

Petri nets, also called *Place/Transition (P/T) nets* (see Definition 1.1), are a formal model of concurrent and reactive systems, introduced by Carl Adam Petri.

Intuitively, Petri nets provide a *calculus*—what could be described as some sort of formal assembly language—for the modeling and the analysis of discrete systems. In this context, what we mean by calculus is a mathematical model that aims to be succinct, with as few and as simple rules as possible, yet expressive enough to reason about interesting properties. And in this case, Petri nets can be used to reason about concepts such as concurrency and nondeterminism, or notions such as causality and temporal logic properties.

In practice, Petri nets and their extensions have been used in various application domains related to computer science. For instance for reasoning about software [GS92; BE12; KKW14] hardware [BKY00; LAG15], database [Boj+11], real-time [Hla+21], or robotic [CL07] systems. But also for more general domains, such as the analysis of biological [Bal+10], chemical [ADS11], ecological [PTG22] or business [Aal15] processes.

The basic idea behind Petri nets is to describe a system as a relation between the current state of its "resources", modeled using *places*, and its possible actions, or events, modeled using *transitions*. Places and transitions are connected together by *arcs* that describe what resources are needed to carry on an action and how they are modified when this action actually happens. The state of a system, called a *marking*, is defined by the local state of all its *places*, that can contain an arbitrary number of tokens. Then the system can change its state by firing a transition. If a condition on the number of tokens in the *input places* is met, the transition can *fire*. In this case, some tokens are removed from the *input places*, and others are added to the *output places*. A complete formalization of Petri nets can be found in [Mur89; Dia09; Rei12].

In the following, we denote $\mathbb{Z}$, the set of integers, and $\mathbb{N}$, the set of natural numbers. We may also use $m..n$ for the set of integers (the interval) between m and n included. Assuming P

is a finite, totally ordered set $\{p_1, \ldots, p_n\}$, we denote by $\mathbb{N}^P$ the set of mappings from $P \to \mathbb{N}$, and we overload the addition, subtraction, and comparison operators $(=, \geqslant, \leqslant)$ to act as their component-wise equivalent on mappings.

Definition 1.1 (Petri Net)**.** *A Petri net N is a tuple $(P, T, \mathrm{Pre}, \mathrm{Post})$ where:*

- $P \triangleq \{p_1, \ldots, p_n\}$ *is a finite set of places;*
- $T \triangleq \{t_1, \ldots, t_k\}$ *is a finite set of transitions (disjoint from P);*
- $\mathrm{Pre} : T \to (P \to \mathbb{N})$ *and* $\mathrm{Post} : T \to (P \to \mathbb{N})$ *are the pre- and post-condition functions (also called the flow functions of N).*

The *pre-set* of a transition $t \in T$ is denoted ${}^\bullet t \triangleq \{p \in P \mid \mathrm{Pre}(t,p) > 0\}$, the *post-set* of a transition t is denoted $t^\bullet \triangleq \{p \in P \mid \mathrm{Post}(t,p) > 0\}$. The values of $\mathrm{Pre}(t,p)$ (respectively $\mathrm{Post}(t,p)$) define the *weight* of the arc from p to t (respectively from t to p). An arc of weight 0 is considered absent. A Petri net is called *ordinary* when its weights belong to $\{0,1\}$. These notations can be extended to the *pre-set* and *post-set* of a place p, with ${}^\bullet p \triangleq \{t \in T \mid \mathrm{Post}(t,p) > 0\}$ and $p^\bullet \triangleq \{t \in T \mid \mathrm{Pre}(t,p) > 0\}$.

1.1.1 States

A state m of a net, also called a *marking*, is a mapping $m : P \to \mathbb{N}$ that assigns a number of *tokens*, $m(p)$, to each place p in P. When we write a marking, we list the marking of all non-empty places, using the notation $p{*}k$ to state that place p has k tokens. Finally, a marked net (N, m_0) is a pair composed of a net N and an initial marking m_0.

1.1.2 Behavior

A transition $t \in T$ is *enabled* at marking $m \in \mathbb{N}^P$ when $m(p) \geqslant \mathrm{Pre}(t,p)$ for all places p in P. We can also simply write $m \geqslant \mathrm{Pre}(t)$. A marking $m' \in \mathbb{N}^P$ is reachable from a marking $m \in \mathbb{N}^P$ by firing transition t, denoted $(N,m) \xrightarrow{t} (N,m')$ or simply $m \xrightarrow{t} m'$ when N is obvious from the context, if: (1) transition t is enabled at m; and (2) $m' = m - \mathrm{Pre}(t) + \mathrm{Post}(t)$. When the identity of the transition is unimportant, we simply write this relation $m \to m'$. The difference between m and m' is a mapping $\Delta(t) \triangleq \mathrm{Post}(t) - \mathrm{Pre}(t)$ in $\mathbb{Z}^P$, called the *displacement* of t.

By extension, we say that a *firing sequence* $\varrho \triangleq t_1 \ldots t_k \in T^*$ can be fired from m, denoted $(N,m) \xrightarrow{\varrho} (N,m')$ or simply $m \xrightarrow{\varrho} m'$, if there exist markings $m_0, \ldots, m_k$ such that $m = m_0$, $m' = m_k$ and $m_i \xrightarrow{t_{i+1}} m_{i+1}$ for all i in the range $0..(k-1)$. In this case, the displacement of ϱ is the mapping $\Delta(\varrho) \triangleq \Delta(t_1) + \cdots + \Delta(t_k)$.

More generally, marking m' is reachable from m in N, denoted $m \to^\star m'$ if there is a (possibly empty) sequence of transitions such that $m \to \cdots \to m'$. We denote $R(N, m_0)$ the set

of markings reachable from m_0 in N:

$$R(N, m_0) \triangleq \{m \mid m_0 \rightarrow^\star m\} \tag{1.1}$$

The *semantics* of a marked net is the Labelled Transition System (LTS), with nodes in $R(N, m_0)$ and edges between states (m, m') whenever $m \xrightarrow{t} m'$ for some transition $t \in T$. We focus mostly on reachable states in our work and will, therefore, seldom refer to the LTS of the net.

Finally, we may use the *Parikh vector* of some sequence ϱ (also called the *Parikh image*) that is the mapping $\wp(\varrho) \in \mathbb{N}^T$, counting the number of occurrences of each transition in ϱ. Then, $\wp(\varrho) \triangleq (|\varrho|_{t_1}, \ldots, |\varrho|_{t_k})$, where $|\varrho|_{t_i}$ denotes the number of occurrences of transition t_i in the sequence ϱ.

1.1.3 Boundedness

A marking m is k-bounded when each place has at most k tokens: property $\bigwedge_{p \in P} m(p) \leqslant k$ is true. Likewise, a marked Petri net (N, m_0) is bounded when there is a constant k such that all reachable markings are k-bounded. We will see in Sect. 1.4 that deciding whether a net is bounded is a decidable problem.

A net is *safe* when it is 1-bounded. In our work, we consider *generalized* Petri nets (in which net arcs may have weights larger than 1) and we do not restrict ourselves to bounded nets.

Conversely, if no bound exists, we say that the net is *unbounded*. This is a necessary and sufficient condition for its reachability set to be infinite.

1.1.4 Labels and Observations

In the following, we will often consider that each transition is associated with a label (a symbol taken from an alphabet Σ). In this case, we assume that a net is associated with a labeling function $l : T \to \Sigma \cup \{\tau\}$, where τ is a special symbol for the silent action name. Every net has a default labeling function l_N such that $\Sigma = T$ and $l_N(t) \triangleq t$ for every transition $t \in T$.

We can lift any labeling function $l : T \to \Sigma \cup \{\tau\}$ to a mapping of sequences from T^* to Σ^*. Specifically, we define inductively $l(\varrho.t) \triangleq l(\varrho)$ if $l(t) = \tau$ and $l(\varrho.t) \triangleq l(\varrho).l(t)$ otherwise, where the dot operator $(.)$ stands for concatenation, and $l(\epsilon) \triangleq \epsilon$, where ϵ is the empty sequence, verifying $\epsilon.\sigma = \sigma.\epsilon = \sigma$ for any $\sigma \in \Sigma^*$. Given a sequence of labels $\sigma \in \Sigma^*$, we write $(N, m_0) \stackrel{\sigma}{\Rightarrow} (N, m')$ if there exists a firing sequence $\varrho \in T^*$ such that $(N, m_0) \xrightarrow{\varrho} (N, m')$ and $\sigma = l(\varrho)$. In this case, σ is referred to as an *observable sequence* of the marked net (N, m_0).

1.1.5 Graphical Notations

We use the standard graphical notation for nets, where places are depicted as circles and transitions as squares. We may use black dots or figures to represent the number of tokens contained in places.

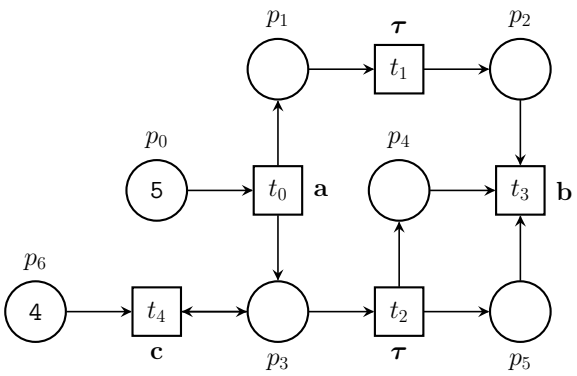


Fig. 1.1 An example of labeled Petri net, (N, m_0).

With the net displayed in Fig. 1.1, and with our convention, the initial marking is $m_0 \triangleq p_0{*}5\ p_6{*}4$ (places p_0 and p_6 have 5 and 4 tokens respectively). We have $m_0 \xrightarrow{\varrho} m$ with $\varrho \triangleq t_0\, t_0\, t_1\, t_1\, t_2\, t_3\, t_4$ and $m \triangleq p_0{*}3\ p_2{*}1\ p_3{*}1\ p_6{*}3$; and therefore $m_0 \xRightarrow{\mathbf{a\,a\,b\,c}} m$ when we only look at (observable) labels.

1.1.6 Relation to Linear Arithmetic Constraints

Many results in Petri net theory are based on a relation to linear algebra and linear programming techniques [Mur89; STC96]. A famous example is that the *potentially reachable markings* (an over-approximation of the reachable markings) of a net (N, m_0) are non-negative, integer solutions to the *state equation* problem [Mur77], $m = I \cdot \sigma + m_0$, with I an integer matrix defined from the flow functions of N called the incidence matrix and σ a vector in $\mathbb{N}^k$. We give more details about it in Section 1.5.

It is also known that solutions to the system of linear equations $\sigma^T \cdot I = \mathbf{0}$ lead to *place invariants*, that is, $\sigma^T \cdot m = \sigma^T \cdot m_0$ for each reachable marking m. This can provide some information used in verification techniques. For example, for the net N (Fig. 1.1), we can compute the invariant $p_4 - p_5 = 0$. This gives us the information that both places always contain the same number of tokens, and so can be marked together for some reachable marking—what is called concurrent in [BG21]—if we prove that one of them is nondead. Likewise, an invariant of the form $p + q = 1$ is enough to prove that p and q are 1-bounded and that both places are nonconcurrent.

Unfortunately, invariants provide only an over-approximation of the set of reachable markings, and it may be difficult to find whether a net is part of the few known classes where the set of reachable markings equals the set of potentially reachable ones [Huj+20a; Huj+20b].

Our *polyhedral approach* shares some similarities with this kind of reasoning. A main difference is that we will use linear constraints to draw a relation between the reachable markings of two nets, not to express constraints about (potentially) reachable markings inside one net.

1.2 Presburger Arithmetic and Petri Net Semantics

Petri nets and Presburger arithmetic [Pre29] share a fascinating relation (see Sect. 1.4). Presburger arithmetic allows us to reason about markings (integer vectors), and so to express properties or relations on them. We first define Presburger formulas:

Definition 1.2 (Presburger Arithmetic and Presburger Formula)**.** *Presburger arithmetic (PA) is the first-order theory of the natural numbers, $\mathbb{N}$, with addition, equality, and the standard axioms of arithmetic. We use the notation $F(x_1, \ldots, x_n)$ for a generic Presburger formula F with free variables $x_1, \ldots, x_n$ (meaning variables not bound by the scope of a quantifier).*

There are different presentations of PA, some relying on integers ($\mathbb{Z}$) instead of natural numbers ($\mathbb{N}$), or using comparison ($<$) instead of equality. We choose to concentrate on natural numbers since we reason on the markings of places. In practice, we will use arbitrary Boolean combinations (using $\vee, \wedge, \neg$) of quantified ($\forall$, $\exists$) atomic propositions of the form $\alpha \sim \beta$, where $\sim$ is one of $=, \leqslant$, or $\geqslant$, and α, β are *linear expressions* with coefficients in $\mathbb{Z}$. In the following, we should often consider formulas F with support in the set of places P of a net and simply use the term *linear constraint* to describe F.

We refer to *quantifier-free formulas* (sometimes also called *short Presburger formulas*) as the subset of formulas without quantifiers, which are simply Boolean combinations of literals. Given a formula F, we denote $\mathrm{FV}(F)$ the set of free variables contained in it. We shall also often concentrate on formulas in *Disjunctive Normal Form* (DNF), for quantifier-free formulas expressed as the disjunction of *cubes*, which are conjunctions of literals.

Presburger himself proved that the truth of a Presburger sentence (a quantifier-free formula) is decidable [PJ91]. When it comes to complexity, we know that the satisfiability problem for full PA is somewhere between 2EXPTIME [FR98] and 3EXPTIME [Opp78]. In contrast, for the quantifier-free fragment of Presburger arithmetic, it is easy to prove that the problem is NP-complete [Pap81]—then with an EXPTIME upperbound—, and that any satisfiable formula has some satisfying assignment of size at most polynomial in the size of the formula [BT76].

1.2.1 Notations

In the remainder, we use the notation $F(\boldsymbol{p})$ for the declaration of a formula F with variables in $\boldsymbol{p}$, instead of the more cumbersome notation $F(p_1, \ldots, p_n)$. We also simply use $F(\boldsymbol{\alpha})$ for the substitution $F\{p_1 \leftarrow \alpha_1\} \ldots \{p_n \leftarrow \alpha_n\}$, with $\boldsymbol{\alpha} \triangleq (\alpha_1, \ldots, \alpha_n)$ a sequence of linear expressions.

We say that a mapping m of $\mathbb{N}^P$ is a *model* of F, denoted $m \models F$, if the ground formula $F(m) \triangleq F(m(p_1), \ldots, m(p_n))$ is true. Hence, we can also interpret F as a predicate over markings. Finally, we define the semantics of F as the set $[\![F]\!] \triangleq \{m \in \mathbb{N}^P \mid m \models F\}$. As usual, we say that a predicate F is `valid` (we also say a tautology), denoted $\models F$, when all its

interpretations are true ($[\![F]\!] = \mathbb{N}^P$); and that F is *unsatisfiable* (or simply `unsat`), denoted $\not\models F$, when $[\![F]\!] = \emptyset$.

In the following, we often express that some predicate F always implies a predicate G. It corresponds to the logical notations: $\forall \boldsymbol{p} \,.\, F(\boldsymbol{p}) \implies G(\boldsymbol{p})$ satisfiable, $\models F(\boldsymbol{p}) \implies G(\boldsymbol{p})$ or $[\![F]\!] \subseteq [\![G]\!]$. In the following, we often prefer the equivalent query to be checked using an SMT solver, that is, $F(\boldsymbol{p}) \wedge \neg G(\boldsymbol{p})$ `unsat` where variables $\boldsymbol{p}$ are implicitly existentially quantified. Finally, we may write $F \equiv G$ when $[\![F]\!] = [\![G]\!]$.

1.2.2 Presburger-Definable Sets

Given a Presburger formula, we can define its set of solutions, that is called a *Presburger set*. We will often say that a set S is *Presburger-definable* if it corresponds to a Presburger set.

Definition 1.3 (Presburger Set)**.** *A set $S \subseteq \mathbb{N}^d$ is a Presburger set if there exists a Presburger formula $F(\boldsymbol{x})$ with $\boldsymbol{x}$ a vector of dimension d such that $S = [\![F]\!]$.*

There are other characterizations of Presburger sets. For example, a set is Presburger-definable if and only if it is semilinear (i.e., it is a finite union of linear sets) [GS66].

1.2.3 Encoding of Petri Net Semantics

We can define many properties on the markings of a net N using Boolean combinations of linear constraints with integer variables (that are quantifier-free Presburger formulas with support on P), and so revisit the semantics of Petri nets. Assume that we have a marked net (N, m_0) with set of places $P \triangleq \{p_1, \ldots, p_n\}$. To any marking m over P, we can associate a linear formula $\underline{m}(x_1, \ldots, x_n)$, below, whose unique model in $\mathbb{N}^P$ is m. In this context, an equation $x_i = k$ means that there must be k tokens in place p_i. Formula $\underline{m}$ is obviously a conjunction of literals, what is called a *cube* in [Bra11].

$$\underline{m}(x_1, \ldots, x_n) \triangleq (x_1 = m(p_1)) \wedge \cdots \wedge (x_k = m(p_k)) \tag{1.2}$$

We often use place names as variables (or parameters) and use $\boldsymbol{p}$ for the vector $(p_1, \ldots, p_n)$ or $F(\boldsymbol{p})$ for a formula with variables in P. We also often use $\underline{m}$ instead of $\underline{m}(\boldsymbol{p})$.

Definition 1.4 (Model of a Formula)**.** *We say that a marking m is a model of (or m* satisfies*) property F, denoted $m \models F$, when formula $F(\boldsymbol{p}) \wedge \underline{m}(\boldsymbol{p})$ is satisfiable. In that case, we may also write $F(m)$ holds (or simply $\models F(m)$).*

We can use this approach to reframe many properties on Petri nets. For instance, the notion of safe markings, described previously: a marking m is safe when $m \models \mathrm{BND}_1(\boldsymbol{p})$, where BND_k

is a predicate defined as:

$$\mathrm{BND}_k(\boldsymbol{p}) \triangleq \bigwedge_{i \in 1..n} (p_i \leqslant k) \tag{1.3}$$

Likewise, the property that some transition t is enabled corresponds to the predicate ENBL_t below, in the sense that t is enabled at m when $m \models \mathrm{ENBL}_t(\boldsymbol{p})$.

$$\mathrm{ENBL}_t(\boldsymbol{p}) \triangleq \bigwedge_{i \in 1..n} (p_i \geqslant \mathrm{Pre}(t, p_i)) \tag{1.4}$$

Another example is the definition of *deadlocks*, which are characterized by the formula:

$$\mathrm{DEAD}(\boldsymbol{p}) \triangleq \bigwedge_{t \in T} \neg \mathrm{ENBL}_t(\boldsymbol{p}) \tag{1.5}$$

We can also define a linear predicate to describe the relation between the markings before and after some transition fires. To this end, we use a vector $\boldsymbol{p'}$ of "primed variables" $(p'_1, \ldots, p'_n)$, where p'_i will stand for the marking of place p_i after a transition is fired. Hence, we define formulas with $2\,n$ variables, and we use the notation $\psi(\boldsymbol{p}, \boldsymbol{p'})$ as a shorthand for $\psi(p_1, \ldots, p_n, p'_1, \ldots, p'_n)$. With this convention, formula $\mathrm{FIRE}_t(\boldsymbol{p}, \boldsymbol{p'})$, defined next in Equation (1.9), is such that $\mathrm{FIRE}_t(m, m')$ entails $m \xrightarrow{t} m'$ when t is enabled at m.

With all these notations, we can define a predicate $\mathrm{T}(\boldsymbol{p}, \boldsymbol{p'})$, see Equation (1.10), that "encodes" the effect of firing one transition in the net N. By construction, formula $\underline{m}(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \wedge \underline{m'}(\boldsymbol{p'})$ is true when $m \to m'$. Note that $\underline{m}(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'})$ is not satisfiable if no transition is enabled at m, in which case $\mathrm{T}_{\mathrm{EQ}}(\boldsymbol{p}, \boldsymbol{p'})$ encodes the effect of firing *at most* one transition, and then is true when $m \xrightarrow{t} m'$ for some transition $t \in T$ or $m = m'$.

$$\begin{aligned}
\mathrm{GEQ}_m(\boldsymbol{p}) &\triangleq \textstyle\bigwedge_{i \in 1..n} (p_i \geqslant m(p_i)) && (1.6)\\
\mathrm{ENBL}_t(\boldsymbol{p}) &\triangleq \textstyle\bigwedge_{i \in 1..n} (p_i \geqslant \mathrm{Pre}(t)(p_i)) && (1.7)\\
\Delta_t(\boldsymbol{p}, \boldsymbol{p'}) &\triangleq \textstyle\bigwedge_{i \in 1..n} (p'_i = p_i + \mathrm{Post}(t)(p_i) - \mathrm{Pre}(t)(p_i)) && (1.8)\\
\mathrm{FIRE}_t(\boldsymbol{p}, \boldsymbol{p'}) &\triangleq \mathrm{ENBL}_t(\boldsymbol{p}) \wedge \Delta_t(\boldsymbol{p}, \boldsymbol{p'}) && (1.9)\\
\mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) &\triangleq \textstyle\bigvee_{t \in T} \mathrm{FIRE}_t(\boldsymbol{p}, \boldsymbol{p'}) && (1.10)\\
\mathrm{EQ}(\boldsymbol{p}, \boldsymbol{p'}) &\triangleq \textstyle\bigwedge_{i \in 1..n} (p'_i = p_i) && (1.11)\\
\mathrm{T}_{\mathrm{EQ}}(\boldsymbol{p}, \boldsymbol{p'}) &\triangleq \mathrm{EQ}(\boldsymbol{p}, \boldsymbol{p'}) \vee \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) && (1.12)
\end{aligned}$$

1.2.4 SMT Theories

SMT solvers [KS08] (for Satisfiability Modulo Theories) are a modern technology that extends the benefits of both SAT solvers and solvers for Integer Linear Programming (ILP) by offering more flexibility in the way constraints are expressed.

Most of the predicates that we define in the remainder of this chapter are unquantified (or can be thought of as having top-level existential quantification.) Hence, they can be expressed using the *Quantifier-Free Linear Integer Arithmetic* (QF-LIA) theory in SMT solvers. Compared to other possible choices, such as the theory of *fixed size Bit Vectors* (BV), QF-LIA has also the advantage of supporting unbounded nets.

In practice, we rely on the SMT-LIB format [BFT17] with the `z3` solver [MB08; Bjø]. When we give a query to the solver, it can answer satisfiable (`sat`) or unsatisfiable (`unsat`). If the query is satisfiable we can extract a model from the current stack. For some unsatisfiable queries, we may ask for a subset of clauses whose conjunction is still unsatisfiable, what is called an *unsatisfiable core* (`unsat core`) of the original formula.

1.3 Reachability Problems

In our work, we focus on the verification of *reachability properties*, meaning properties on the states that a marked net (N, m_0) can reach. We can not only check the reachability of a given state but also if it is possible to reach a marking that satisfies a combination of linear constraints between places. In fact, we support two categories of queries (that we characterize using modalities from *Computation Tree Logic* [CE81]): $\mathrm{EF}\, F$, which is true only if F is reachable; and $\mathrm{AG}\, F$, which is true when F is an invariant, where F is a quantifier-free Presburger formula with support on the set of places (it has no modalities). We have the classic relation that $\mathrm{AG}\, F \equiv \neg\, (\mathrm{EF}\, \neg F)$. At various times, we will use the fact that F is invariant if and only if its negation is not reachable: $\mathrm{EF}\, \neg F$ is false.

Definition 1.5 (Invariant and Reachable Properties)**.** *Property F is an invariant on (N, m_0) if and only if we have $m \models F$ for all $m \in R(N, m_0)$. We say that F is reachable when there exists $m \in R(N, m_0)$ such that $m \models F$.*

A *witness* for property $\mathrm{EF}\, F$ is a reachable marking such that $m \models F$; it is a *counterexample* for $\mathrm{AG}\, \neg F$. We can deal with any property that can be expressed using a linear predicate. Examples of properties we can express in this way include: whether some marking m is reachable; whether some transition t is enabled, commonly known as *quasi-liveness* (Definition 1.9); whether there is a deadlock (Definition 1.6); whether some generalized constraint between place markings is true, such as $(p_0 + p_1 = p_2 + 2) \wedge (p_1 \leqslant p_2)$; etc.

Definition 1.6 (Deadlock)**.** *We say that a net (N, m_0) admits a deadlock, if and only if there is a marking m in $R(N, m_0)$ such that $m \not\geqslant \mathrm{Pre}(t)$ for all t in T.*

1.3.1 Coverability Properties

A subproblem of the reachability problem is that of *coverability* (Definition 1.7). It consists in deciding whether a given marking is included in some reachable marking of a given Petri net.

Definition 1.7 (Marking Coverability)**.** *We say that a marking m of (N, m_0) is coverable if and only if there is m' in $R(N, m_0)$ such that $m' \geqslant m$.*

This problem can be generalized to *monotonic* formulas, for which we may refer to *coverability properties* (Definition 1.8).

Definition 1.8 (Coverability Property)**.** *Property F is a coverability property, that we also call monotonic formula, if and only if $m \models F$ implies $m' \models F$ for all markings $m' \geqslant m$.*

Using coverability properties, it is obviously possible to encode the standard marking coverability problem, but also the quasi-liveness of some transition t (Definition 1.9).

Definition 1.9 (Quasi-liveness)**.** *We say that a transition t of (N, m_0) is quasi-live if and only if there is m in $R(N, m_0)$ such that $m \geqslant \mathrm{Pre}(t)$.*

1.3.2 The Concurrent Places Problem

In this work, we also study more complex reachability problems, such as the *concurrent places* problem [BG21]. We say that places p, q of a net N are concurrent when there exists a reachable marking m with both p and q marked. The *concurrent places* problem consists of enumerating all such pairs of places. This problem is fundamental for computing decompositions into networks of automata [BGP20; BG21]. Note that this problem can be encoded by coverability properties.

Definition 1.10 (Dead and Concurrent Places)**.** *We say that a place p of (N, m_0) is* nondead *if there is m in $R(N, m_0)$ such that $m(p) > 0$. Similarly, we say that places p, q are* concurrent*, denoted $p \parallel q$, if there is m in $R(N, m_0)$ such that both $m(p) > 0$ and $m(q) > 0$. By extension, we use the notation $p \parallel p$ when p is nondead. We say that p, q are nonconcurrent, denoted $p \# q$, when they are not concurrent.*

1.4 Theoretical Results

We now present some theoretical results about Petri nets and the reachability problem. Some references may refer to *Vector Addition System* (VAS for short, introduced by Karp and Miller [KM69]), or even additionally considering control state, corresponding to *Vector Addition*

System with States (VASS) [Gre78; HP79]. These models are computationally equivalent to Petri nets since they can simulate each other. Note that a Petri net can be transformed by a straightforward polynomial-time translation (see [Sch16]) to an equivalent VASS by preserving the reachability set, so the formal presentation of VASS can be skipped.

1.4.1 Decidability

One of the most important results in concurrency theory is the decidability of the reachability problem for Petri nets proved by Mayr in 1981 [May81]. The proof was then simplified by Kosarauju [Kos82] and Lambert [Lam92]. Even if this result is based on a constructive proof, and its "construction" streamlined over time [Ler09], the classical Kosaraju-Lambert-Mayr-Sacerdote-Tenney approach does not lead to a workable algorithm. It is in fact a feat that this algorithm has been implemented at all, e.g., see the tool KReach [DL20]. A recent approach developed by Leroux differs by considering Presburger inductive invariants [Ler09; Ler11].

A particular belief in the Petri net community is that reachability is at the frontier of decidability. The following quote from James L. Peterson is a good illustration:

> "In general, it seems that any extension which does not allow zero testing will not actually increase the modeling power (or decrease the decision power) of Petri nets but merely result in another equivalent formulation of the basic Petri net model. (Modeling convenience may be increased.)" [Pet81]

Actually, the most popular extensions to Petri nets, when they significantly increase their expressiveness power, are Turing-complete, and so entail the undecidability of the reachability problem. (We do not consider the issue of succinctness in our work.) For instance, the addition of *inhibitor* (or *zero test*) arcs add the ability to test for the absence of tokens in a place and result in a Turing equivalent model [Age74]. A less trivial example is with *reset* arcs (for which undecidability has been proved in [AK76]), which consume all tokens from their input place when the transition fires. The same holds for *transfer* arcs, used to transfer all tokens from one place to another when a transition fires [DFS98].

It is also interesting to look at other decidability problems for Petri nets [EN94], outside reachability. For instance, checking that a net is bounded, e.g., the reachability set is finite, is decidable [KM69], and can be done by constructing the *coverability tree*. Some other interesting problems are known as equivalent to the reachability problem [AK76], and so decidable. For instance, Hack showed that the *liveness* problem (that is checking whether any transition t can always be eventually fired from every reachable marking) is recursively equivalent to the reachability problem [Hac76]. As presented previously in Definition 1.6, the *deadlock-freedom* problem can be easily reduced to the reachability problem [CEP95]. Another interesting problem for our work consists of deciding if the reachability set of a given Petri net is semilinear, and so Presburger-definable. The decidability was proved by Hauschildt [Hau90] and Lambert [Lam90].

However, these works do not provide an algorithm for computing Presburger formulas denoting the reachability set when it exists.

If we consider equivalence problems instead of reachability, we find that most of them are undecidable. Hack proved in [Hac76] that *marking equivalence*, that is checking whether two nets have the same reachability set, is undecidable. Similarly, trace and language equivalence can be reduced to the marking equivalence problem [Hac76], even if more direct approaches have been proposed to tackle the problem [AK76]. Finally, checking that the reachability graphs of two nets are (strongly) bisimilar, what is referred to as the *bisimulation equivalence* problem, is also undecidable [Jan94].

1.4.2 Complexity

A first lower bound for the complexity of the reachability problem was stated as early as 1976 by Lipton [Lip76], which gives an EXPSPACE lower bound. This bound has been improved to NONELEMENTARY over 40 years after [Cze+20]. Finally, the complexity of the reachability problem has been recently fully characterized as Ackermann-complete by Leroux [Ler22], and by Czerwiński and Orlikowski [CO22] using two different constructions (the upper bound was already stated as Ackemaniann [LS19]). Note that the subclass of coverability problems has a "far simpler", even though still EXPSPACE-complete, theoretical complexity [Lip76; Rac78].

A practical consequence of this "inherent complexity", and a consensus among the Petri net community, is that we should not expect to find a one-size-fits-all algorithm that could be usable in practice. A better strategy is to try to improve the performances on some cases—for example by developing new tools, or optimizations, that may perform better on some examples—or try to extend the class of problems we can handle—by finding algorithms that can manage new cases.

This wisdom is illustrated by the current state-of-the-art at the Model Checking Contest. As a matter of fact, the top three tools in recent competitions—ITS-Tools [Thi15], TAPAAL [Dav+12], and LoLA [Wol18]—all rely on a portfolio of approaches, and mix different methods.

1.4.3 Relation to Presburger Arithmetic

In general, the reachability set of a Petri net is not Presburger-definable. A well-known example is a net introduced by Hopcroft and Pansiot [HP79], see Figure 1.2, associated to a reachability set $R(N, m_0)$ characterized as follows:

$$\left\{ (p_0, p_1, p_2, p_3, p_4) \in \mathbb{N}^5 \;\middle|\; \vee \begin{array}{l} (p_0 = 1 \wedge p_3 = 0 \wedge 1 \leqslant p_1 + p_2 \leqslant 2^{p_4}) \\ (p_0 = 0 \wedge p_3 = 1 \wedge 1 \leqslant p_1 + 2p_2 \leqslant 2^{p_4+1}) \end{array} \right\}$$

However, a specific class of Petri nets with Presburger-definable reachability sets have been defined; called *flat nets* in [Bar+03; LS05; Bar+08; Ler13]. This class can be characterized

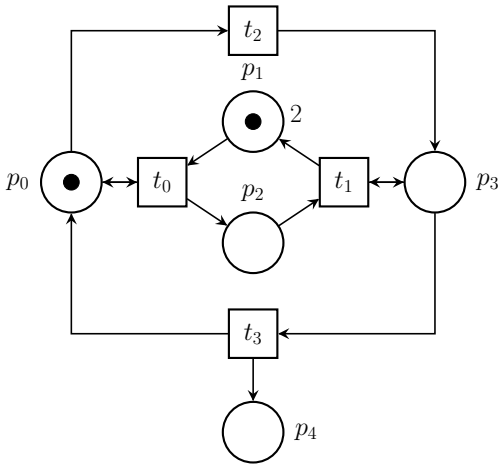


Fig. 1.2 Hopcroft and Pansiot Petri net example.

using a structural criterion at the level of VASS. Namely, flat nets are Petri nets that admit a corresponding VASS that can be unfolded into a VASS without nested cycles, called a *flat* VASS. Equivalently, a net N is flat if its language is flat, that is, there exists some finite sequence $\varrho_1 \dots \varrho_k \in T^*$ such that for every reachable marking m in $R(N, m_0)$ there is a sequence $m_0 \xrightarrow{\varrho} m$ with $\varrho \in \varrho_1^* \dots \varrho_k^*$. In short, all reachable markings can be reached by simple sequences, belonging to the language: $\varrho_1^* \dots \varrho_k^*$. Last but not least, Leroux stated in [Ler13], Theorem 1.1, that a net is flat if and only if its reachability set is Presburger-definable.

Theorem 1.1 ([Ler13])**.** *For every VASS V, for every Presburger set C_{in} of configurations, the reachability set* $\mathrm{ReachV}(C_{in})$ *is Presburger if, and only if, V is flattable from C_{in}.*

Even if the reachability set of a net is not Presburger-definable, following a property proved by Leroux [Ler09; Ler10], for any non-reachable marking m (or property F) "*there exist checkable certificates of non-reachability in the Presburger arithmetic*". A certificate of non-reachability (that we also call *certificate of invariance* in our work) is an inductive predicate that contains the initial marking m_0 but does not contain m (or intersect F). However, this result does not explain how to effectively compute such an invariant, but we propose a semi-decision procedure in Chapter 2.

Theorem 1.2 ([Ler10])**.** *For every VASS V, for every Presburger sets of configurations C_{in}, C_{out}, either $c_{\mathrm{in}} \xrightarrow{\varrho} c_{\mathrm{out}}$ for some configurations $c_{\mathrm{in}} \in C_{\mathrm{in}}$ and $c_{\mathrm{out}} \in C_{\mathrm{out}}$ and some word ϱ of transitions, or there exists a Presburger inductive invariant C that contains C_{in} and disjoints from C_{out}.*

Note that the formulation of Theorems 1.1 and 1.2 is taken from an invited talk of Leroux [Ler21] that provides an overview of results about the reachability problem for VASS related to Presburger arithmetic.

1.5 Model Checking Methods and Optimizations

First symbolic model checkers were based on Binary Decision Diagrams (BDDs) [Bur+92], a data structure able to efficiently represent the truth table of Boolean functions in a compact way, and therefore also sets of Boolean vectors. One example is the model checker SMV [McM93], used for the verification of the IEEE Future+ cache coherence protocol, which is one of the celebrated breakthroughs in the area of formal verification listed in [Gar12]. Many state-of-the-art model checkers today use BDD-like structures (called decision diagrams) to encode sets of states and state transitions. For example, there are extensions of decision diagrams that can handle multivalued logics, often on finite domains, for instance with bounded intervals of integers. Examples include the Multivalued Decision Diagram (MDD) [ADC20] used in GreatSPN [Amp+16; ADG22] or the Hierarchical Set Decision Diagrams (SDD) [Thi+09] used in Tedd, part of the Tina toolbox [BRV04]. However, such techniques are limited to bounded nets, and in practice do not measure up with other approaches used in the Model Checking Contest for reachability properties [Amp+19].

In this work, we compare ourselves with three tools: ITS-Tools [Thi15], TAPAAL [Dav+12] and LoLA [Wol18], that have in common to be in the top trio of the Model Checking Contest [Amp+19] (and therefore accept a common syntax for nets and properties, which allow for a fair comparison).

Methods available in such portfolio tools include the use of symbolic techniques, such as Bounded Model Checking [Bie+99], k-induction [Thi20; Thi21]; abstraction refinement [CJL17]; the use of standard optimizations with Petri nets, like partial order reductions [Wol07] or structural reductions [Thi20; Thi21]; the use of the "state equation" method [Thi20; Thi21]; reduction to integer linear programming problems; or even simulation techniques, which are often very effective at finding witnesses or counter-examples [Wol18; Thi20; BHO21; Thi21; Hen+23].

The results obtained during the MCC highlight the very good performances achieved when putting all these techniques together, on bounded and unbounded nets, with a collection of randomly generated properties.

In the remainder of the section, we propose an overview of these different techniques. We assume that we have a marked net (N, m_0) with sets of places $P \triangleq \{p_1, \ldots, p_n\}$ and transitions $T \triangleq \{t_1, \ldots, t_k\}$.

1.5.1 Random Walk State Space Exploration

Random walk state space exploration [Wol18; Thi20; BHO21; Thi21; Hen+23], or *random walk* for short, is the simplest model checking technique for finding witnesses to some property F. It consists in randomly exploring the state space of the net, without the need to keep track of the previously visited states, until a witness is found. This technique is implemented in most of the portfolio model checkers and performs well since they are not memory-bound and can

therefore reach a very high throughput of visited states. We are in an instance that illustrates the famous saying by Dijkstra that "testing can be used to show the presence of bugs, but never to show their absence".

Description of the Semi-Algorithm

The semi-algorithm (see Algorithm 1.1) starts from the initial marking m_0, and checks if it is a model for a given formula F, which usually models a set of "feared events". If not, it successively picks some random transition (RANDOMCHOICE) among the set of enabled transitions at the current marking m (ENABLEDTRANSITIONS(m)); and fires it until a witness is found. The procedure may restart from the initial marking after exploring a trace of length MAXTRACELENGTH, or if a deadlock is reached (ENABLEDTRANSITIONS(m) $= \emptyset$). Of course, termination is not guaranteed, even if property F is reachable, and the procedure can never prove F as not reachable (i.e., $\neg F$ invariant) since we never know if we visited all the reachable markings.

Algorithm 1.1 WALK(EF F)

In: F: a linear predicate.
Out: if $\top$ then F is reachable.

```
1:  m ← m_0
2:  i ← 0
3:
4:  while m ⊭ F do
5:      ;; Restart if the maximum length trace is reached or no transition is enabled at m.
6:      if i = MAXTRACELENGTH or ENABLEDTRANSITIONS(m) = ∅ then
7:          i ← 0
8:          m ← m_0
9:      else
10:         ;; Fire a random transition enabled at m.
11:         i ← i + 1
12:         t ← RANDOMCHOICE(ENABLEDTRANSITIONS(m))
13:         m ← FIRE(m, t)
14:
15: ;; A reachable marking m satisfying F has been found.
16: return ⊤
```

Tools may implement search heuristics to increase the likelihood of finding some witness to property F. In ITS-Tools [Thi20; Thi21], the random explorer is more likely to fire a transition again if it is still enabled after one firing (encouraging to fully empty places); and can prefer newly enabled transitions (*Depth-First Search*), or transitions that have been enabled a long time (*Breadth-First Search*). Authors in [Hen+23] proposed another search heuristic, called *Random Potency-First Search* (RPFS), which combines a heuristic search based on distance

function together with randomness. It consists of learning which transitions are more likely to contribute to achieving the reachability goal during the exploration, and dynamically modiying the selection function.

1.5.2 Bounded Model Checking (BMC)

Bounded Model Checking (BMC) is an iterative method for exploring the state space of systems by unrolling their transitions [Bie+99]. The method was originally based on an encoding of transition systems into (a family of) propositional logic formulas and the use of SAT solvers to check these formulas for satisfiability [Cla+01]. Several works adapt BMC to Petri nets, such as [Hel01]. More recently, this approach was extended to more expressive models, and richer theories, using SMT solvers [AMP06].

Description of the Semi-Algorithm

In BMC, as with random walk, we try to find a reachable marking m that is a model for a given formula F. The semi-algorithm (see Algorithm 1.2) starts by computing a formula, say ϕ_0, representing the initial marking and checking whether $\phi_0 \wedge F$ is satisfiable (meaning F is initially true). If the formula is unsatisfiable, we compute a formula ϕ_1 representing all the markings reachable in one step, from the initial marking and check $\phi_1 \wedge F$. This way, we compute a sequence of formulas $(\phi_i)_{i\in\mathbb{N}}$ until either $\phi_i \wedge F$ is satisfiable—in which case a witness is found—or we have $\phi_{i+1} \implies \phi_i$—in which case we have reached a fixed point and no witness exists. This stopping test is not performed in practice since k-induction, presented next, is preferred when we try to prove an invariant. The BMC method is not complete since it is not possible, in general, to bound the number of iterations needed to give an answer. Also, when the net is unbounded, we may very well have an infinite sequence of formulas $\phi_0 \subsetneq \phi_1 \subsetneq \dots$ However, in practice, this method can be very efficient in finding a witness (or, depending on the context, a counter-example) when it exists.

The crux of the method is to compute formulas ϕ_i that represent the set of markings reachable using firing sequences of length i. We show how we can build such formulas incrementally.

Formula ϕ_i is the result of connecting i successive occurrences of formulas of the form $\mathrm{T}(\boldsymbol{p_j}, \boldsymbol{p_{j+1}})$. We define the formulas inductively, with a base case (ϕ_0) that states that only m_0 is reachable initially. To define the ϕ_i's, we assume that we have a collection of (pairwise disjoint) sequences of variables, $(\boldsymbol{p_i})_{i\in\mathbb{N}}$. In the following listings, we use the auxiliary function freshVariables to iterate over this family of variable vectors (and that takes the variables' domain as input).

$$\phi_0 \triangleq \underline{m_0}(\boldsymbol{p}_0) \qquad \phi_{i+1} \triangleq \phi_i \wedge \mathrm{T}(\boldsymbol{p_i}, \boldsymbol{p_{i+1}})$$

We can prove that this family of BMC formulas provides a way to check reachability properties, meaning that formula F is reachable in (N, m_0) if and only if there exists $i \geqslant 0$ such that $F(\boldsymbol{p_i}) \wedge \phi_i(N, m_0)$ is satisfiable. The approach we describe here is well-known (see, for

Algorithm 1.2 BMC(EF F)

In: F: a linear predicate.
Out: if $\top$ then F is reachable.

1: $\boldsymbol{p} \leftarrow$ FRESHVARIABLES($\mathbb{N}^P$)
2: $\phi \leftarrow \underline{m_0}(\boldsymbol{p})$
3:
4: **while** UNSAT($\phi \wedge F(\boldsymbol{p})$) **do**
5: *;; Formula $\phi \wedge F$ is unsatisfiable, then consider a longer sequence for ϕ.*
6: $\boldsymbol{p'} \leftarrow$ FRESHVARIABLES($\mathbb{N}^P$)
7: $\phi \leftarrow \phi \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'})$
8: $\boldsymbol{p} \leftarrow \boldsymbol{p'}$
9:
10: *;; $\phi \wedge F$ is satisfiable, then F is reachable.*
11: **return** $\top$

instance, [Bie+99]). It is also quite simplified. Actual model checkers that rely on BMC apply several optimization techniques, such as compositional reasoning; acceleration methods; or the use of invariants on the underlying model to add extra constraints.

1.5.3 Induction and k-Induction

While random walk and BMC are suited to find counter-examples or witnesses, other methods can be used to prove that some property F is an invariant (that is, $\neg F$ is not reachable).

Induction is a basic method that checks if a property is an inductive invariant, meaning it is not possible to escape the invariant by firing any transition. This property is "easy" to check, even though interesting properties are seldom inductive.

Definition 1.11 (Inductive Predicate)**.** *A linear predicate F is inductive if $m_0 \models F$ for the initial marking m_0 and, for all markings m and m' we have $(m \models F \wedge m \to m')$ entails $m' \models F$.*

A property F is inductive if and only if both properties hold: (1) $\underline{m_0}(\boldsymbol{p}) \wedge \neg F(\boldsymbol{p})$ is unsatisfiable; and (2) $F(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \wedge \neg F(\boldsymbol{p'})$ is unsatisfiable. Note that checking condition (2) is equivalent to proving that $(F(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'})) \implies F(\boldsymbol{p'})$ is a tautology.

But still, some simple invariants may not be inductive by unrolling transitions only once. For example, $F \triangleq (q = 0)$ is clearly an invariant of the (dead) net depicted in Fig. 1.3. Yet, condition (2) does not hold: $(p = 1 \wedge q = 0) \xrightarrow{t} (p' = 0 \wedge q' = 1)$. However, by considering sequences of k transitions, some invariants can be shown to hold. This is the idea behind k-induction [SSS00], an extension of the BMC and "induction" methods. In fact, considering our example, the invariant holds after two steps, i.e., $\models \forall \boldsymbol{p}, \boldsymbol{p'}, \boldsymbol{p''} \,.\, F(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \wedge F(\boldsymbol{p'}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p''}) \implies F(\boldsymbol{p''})$. In this case, we say that property F is 2-inductive.

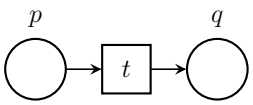


Fig. 1.3 Net example for k-induction.

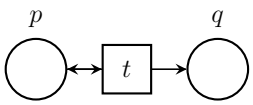


Fig. 1.4 Net example for the state equation.

Description of the Semi-Algorithm

The semi-algorithm (Algorithm 1.3) starts by computing a formula $\psi_0(\boldsymbol{p_0}, \boldsymbol{p_1}) \triangleq F(\boldsymbol{p_0}) \wedge \mathrm{T}(\boldsymbol{p_0}, \boldsymbol{p_1})$, and check whether $\psi_0(\boldsymbol{p_0}, \boldsymbol{p_1}) \wedge \neg F(\boldsymbol{p_1})$ is unsatisfiable or not. If it is unsatisfiable, we must ensure that the first iteration ($i = 0$) of BMC does not find a witness. If not, we proved that F is an invariant with exactly the same queries as the induction method. In the other case, if $\psi_0(\boldsymbol{p_0}, \boldsymbol{p_1}) \wedge \neg F(\boldsymbol{p_1})$ is satisfiable, we continue by unrolling the transitions and computing a formula ψ_1 representing the states reachable by firing two transitions consecutively from F as: $\psi_1(\boldsymbol{p_0}, \boldsymbol{p_1}, \boldsymbol{p_2}) \triangleq \psi_0(\boldsymbol{p_0}, \boldsymbol{p_1}) \wedge F(\boldsymbol{p_1}) \wedge \mathrm{T}(\boldsymbol{p_1}, \boldsymbol{p_2})$ and check whether $\psi_1 \wedge \neg F(\boldsymbol{p_2})$ is unsatisfiable or not. The iteration continues until $\psi_i \wedge \neg F$ is unsatisfiable. But again, the procedure is not complete and may not terminate.

Algorithm 1.3 k-Induction(AG F)

In: F: a linear predicate.
Out: if $\top$ then F is an invariant,
if $\bot$ then $\neg F$ is reachable.

```
1:  p ← freshVariables(ℕ^P)
2:  p' ← freshVariables(ℕ^P)
3:  φ ← m_0(p)
4:  ψ ← F(p) ∧ T(p, p')
5:
6:  while ⊤ do
7:      if sat(φ ∧ (¬F)(p)) then
8:          ;; Formula φ ∧ ¬F is satisfiable, then ¬F is reachable (meaning F is not an invariant).
9:          return ⊥
10:     else if unsat(ψ ∧ (¬F)(p')) then
11:         ;; Formula ψ ∧ ¬F is unsatisfiable, then F is invariant.
12:         return ⊤
13:     else
14:         ;; Consider a longer sequence for both formulas φ and ψ.
15:         p'' ← freshVariables(ℕ^P)
16:         φ ← φ ∧ T(p, p')
17:         ψ ← ψ ∧ F(p') ∧ T(p', p'')
18:         p ← p'
19:         p' ← p''
```

1.5.4 State Space Over-Approximation

A classical result from Petri net theory is the possibility to over-approximate the reachability set, by only looking at the solutions of a linear system, called the *state equation* [Mur77]. Checking that these solutions, called *potentially reachable markings*, do not violate a predicate F is sufficient for stating the invariance of F. If property F is true on all the solutions of the state equation then necessarily F is an invariant.

Definition 1.12 (State Equation)**.** *Given a Petri net (N, m_0), and its incidence matrix I in $\mathbb{Z}^{P\times T}$, defined by $I_{p,t} \triangleq \mathrm{Post}(t,p) - \mathrm{Pre}(t,p)$. The state equation is the system $m_0 + I \cdot \boldsymbol{x} = \boldsymbol{p}$, with $\boldsymbol{p}$, $\boldsymbol{x}$ vectors over $\mathbb{N}^P$ and $\mathbb{N}^T$. The solutions of $\boldsymbol{p}$ are called the potentially reachable markings, and the ones of $\boldsymbol{x}$ are Parikh images of some sequences ϱ in T^*.*

Recent works study Petri nets subclasses for which the set of reachable markings equals the set of potentially reachable ones, a property called the PR-R equality in [Huj+20a; Huj+20b]. A well-known class of Petri nets for which this property holds are live marked graphs; ordinary nets such that every place has exactly one input and one output transition ($\forall p \in P$, $|{}^\bullet p| = |p^\bullet| = 1$). Note that liveness is easy to check for marked graphs [Com+71]. Recent work has studied more general classes, for instance by allowing weights on the arcs, or by considering the case of live weighted marked graphs with a single shared place. On such subclasses, the state equation method is complete and also allows us to state when some property F is reachable (if and only if the solutions of the state equation satisfies F).

Description of the Algorithm

The algorithm (Algorithm 1.4) starts by checking if the query $(\underline{m_0}(\boldsymbol{p}) + I \cdot \boldsymbol{x} = \boldsymbol{p}) \wedge \neg F(\boldsymbol{p})$ has solutions. If unsatisfiable, the potentially reachable markings from the state equation are sufficient to conclude that F is invariant. If satisfiable, we can proceed with a refinement by adding additional constraints, preserving the reachability set.

For example, the net depicted in Fig. 1.4 has $q = 0$ for invariant. The state equation is ineffective in proving it. Indeed, $(p = 0) \wedge (q = x_t) \wedge \neg(q > 0)$ is satisfiable, with $p = 0, q = x_t = 1$ as a model. It is a known limitation with the state equation since this method does not provide an accurate approximation when we test a place (there is both an input and an output arc to the same place from a transition). The author in [Thi20; Thi21] proposes to constrain the state equation with read arc contraints; see Definition 1.13. It requires that for any transition t used in the candidate Parikh vector, that reads from an initially insufficiently marked place p, there must be a transition t' with a positive Parikh count that feeds p. If we go back to our example, place p is insufficiently marked for firing t, and there does not exist such transition t' feeding p, so we learn $x_t \implies \bot$. In this case, this refinement is enough to prove that F is an invariant.

Definition 1.13 (Read Arc Constraints [Thi21]). *For each transition $t \in T$, for every initially insufficiently marked place it reads from, i.e., $\forall p \in {}^\bullet t$ such that $\Delta(t,p) = 0$ and $\mathrm{Pre}(t,p) > m_0(p)$, we assert that:*

$$x_t > 0 \implies \bigvee_{t' \in p^\bullet \setminus \{t\} \,|\, \Delta(p,t') > 0} x_{t'} > 0$$

Another well-known example of refinement relies on *trap constraints* [EM00; Esp+14]. A trap (Definition 1.14) is a subset of places that "once marked, will always stay marked": any transition that consumes tokens from the (places in the) trap must have an output arc into the trap. If we consider initially marked traps, we can discard unfeasible behavior that is otherwise compatible with the state equation. Namely, if the trap is initially marked then it is not possible to reach some marking for which the trap is unmarked.

Definition 1.14 (Trap). *A trap $S \subseteq P$ is a subset of places such that any transition-consuming tokens from the set S must also feed this set.*

$$\forall p \in S \;.\; \forall t \in p^\bullet \;.\; \exists p' \in t^\bullet \;.\; p' \in S$$

The author in [Thi21] proposes a method that starts from a potential candidate marking, m, that is a counter-example according to the state equation. Then we try to contradict this fact by finding if there is a trap initially marked, in m_0, but not marked in m. This is done iteratively and lazily since there can be an exponential number of traps in a net. The idea is to use one Boolean variable b_i per place p_i, that will be true when p_i is in the trap; and checking the following set of constraints: (1) *trap is initially marked*: $\bigvee_{p_i \in P \,|\, m_0(p) > 0} b_i$; (2) *trap definition is satisfied*: $\forall p_i \in P \;.\; (b_i \implies \bigwedge_{t \in p_i^\bullet} (\bigvee_{p_j \in t^\bullet} b_j))$; and (3) *consider only unmarked places for the trap candidate*: $\bigwedge_{p_i \in P \,|\, m(p) > 0} \neg b_i$. If the constraints are satisfiable, we have found a trap S from which we can derive a constraint expressed as $\bigvee_{p \in S} p > 0$ that can be added to the main procedure. The procedure is iterated until no more useful trap constraints are found, or the refinement is sufficient to conclude.

The state equation method can be mixed with others. For instance, even if the net does not satisfy the PR-R equality, and the solutions of the state equation do not permit concluding, we can always learn a "Parikh vector" from the satisfiability of $\mathcal{C}(\boldsymbol{p}, \boldsymbol{x}) \wedge \neg F(\boldsymbol{p})$ (line 2 of Algorithm 1.4); namely information about the number of occurrences of each transition inside a potential execution sequence starting from m_0 and leading to a (potential) witness. This vector is not necessarily the image of a realizable sequence, but it is still a good heuristic to use in order to "guide" a random walk state space exploration for instance.

The state equation method can also be used to improve k-induction. Indeed, the state equation provides a good starting point for k-induction, by iterating from $\mathcal{C} \wedge F$ instead of only F.

Algorithm 1.4 OVER-APPROXIMATION(AG F)

In: F: a linear predicate.
Out: if $\top$ then F is an invariant.

$\boldsymbol{p}, \boldsymbol{x} \leftarrow$ FRESHVARIABLES($\mathbb{N}^P$), FRESHVARIABLES($\mathbb{N}^T$)

;; State equation.
$\mathcal{C}(\boldsymbol{p}, \boldsymbol{x}) \leftarrow \underline{m_0}(\boldsymbol{p}) + I \cdot \boldsymbol{x} = \boldsymbol{p}$

;; Is there a potentially reachable marking that contradicts F?
if UNSAT($\mathcal{C}(\boldsymbol{p}, \boldsymbol{x}) \wedge \neg F(\boldsymbol{p})$) **then**
    ;; If not, F is an invariant.
    return $\top$

;; Refine by adding read arc constraints.
for all t_i in T **do**
    for all p in ${}^\bullet t$ such that $\Delta(p,t) = 0$ and $\mathrm{Pre}(p,t) > m_0(p)$ **do**
        $\mathcal{C} \leftarrow \mathcal{C} \wedge (x_i > 0 \implies \bigvee_{t_j \in {}^\bullet p \setminus \{t\} \,|\, \Delta(p,t_j) > 0} x_j > 0)$

while SAT($\mathcal{C}(\boldsymbol{p}, \boldsymbol{x}) \wedge \neg F(\boldsymbol{p})$) **do**
    ;; Extract a potential counter-example m.
    $m \leftarrow$ GETMODEL($\mathcal{C}(\boldsymbol{p}, \boldsymbol{x}) \wedge \neg F(\boldsymbol{p})$)
    ;; Search for an initially marked trap that contradicts m.
    $\boldsymbol{b} \leftarrow$ FRESHVARIABLES($\{0,1\}^T$)
    $\mathcal{T}(\boldsymbol{b}) \leftarrow (\bigvee_{p_i \in P \,|\, m_0(p) > 0} b_i) \wedge (\forall p_i \in P \,.\, (b_i \implies \bigwedge_{t \in p_i^\bullet} (\bigvee_{p_j \in t^\bullet} b_j))) \wedge (\bigwedge_{p_i \in P \,|\, m(p) > 0} \neg b_i)$
    if SAT($\mathcal{T}(\boldsymbol{b})$) **then**
        ;; If such trap is found, then refine by adding a trap constraint.
        $S \leftarrow$ GETMODEL($\mathcal{T}(\boldsymbol{b})$)
        $\mathcal{C} \leftarrow \mathcal{C} \wedge \bigvee_{S(b_i) = \top} p_i > 0$
    else
        ;; Otherwise, the verdict is unknown.
        return unkonwn

;; Refinements are sufficient to prove F invariant.
return $\top$

1.5.5 Counter-Example Guided Abstraction Refinement (CEGAR)

Counter-Example Guided Abstraction Refinement (CEGAR) approaches [Cla+00] combine two key ideas: abstraction, by considering an over-approximation of the state space; and refinement, as a way to incrementally improve the approximation of the state space. The purpose of abstraction refinement is to exclude, or block, potential counter-examples (at the abstraction level) that are not genuine in the actual system, without losing actual witnesses. The CEGAR approach can be adapted to different formal models. In the context of Petri net, an adequate choice for the initial abstraction is to choose the potentially reachable markings from the state equation.

Description of the Semi-Algorithm

Initially, CEGAR (Algorithm 1.5) starts with the state equation and attempts to verify the property of interest. As previously, if the state equation does not violate the predicate F, the verification is considered successful, and we proved F invariant. However, if the intersection is not empty, we can extract a Parikh vector π, which describes a potential sequence of transitions ϱ (up-to permutation of the transitions) leading to a counter-example. There are two cases. If such sequence ϱ is feasible from m_0 then we have an actual counter-example, and so $\neg F$ is reachable. Otherwise, we can refine the state equation in order to block π from the set of potential solutions. This test is repeated until we find a feasible counter-example or until we can prove that F is an invariant. Note that we use a brute force approach (we test all possible permutations) in order to check if at least one sequence is feasible. We could use a more clever method, but it is not always possible to avoid iterating over a large number of permutations in order to check if a feasible sequence exists.

To block spurious sequences (and so refine the state equation), the CEGAR approach uses linear inequalities over transitions, called *constraints*. Authors in [WW12] refer to two kinds of constraints: *jump constraints* of the form $|t_i| < n$ where $|t_i|$ represents the firing count of the transition t_i, and *increment constraints* of the form $\Sigma_{i \in 1..k} n_i.|t_i| \geqslant n$. A precise description of the CEGAR method is outside the scope of this chapter, and we refer interested readers to [Haj14] for more information. We also describe several methods for blocking "potential counter-examples" in our instantiation of PDR for Petri nets (see Chapter 2). A similar approach could be applied to derive a new instantiation of CEGAR for unbounded Petri nets.

The initial approach developed in [WW12] was not entirely correct, and authors in [Haj14] exhibited a counter-example. The same authors also proposed a correction. The tool TAPAAL also includes a CEGAR-like approach, called *Trace Abstraction Refinement* (TAR) [HHP09]. Unfortunately, only their approach for time Petri nets has been published [CJL17], and no documentation for standard Petri nets is available.

Algorithm 1.5 CEGAR(AG F)

In: F: a linear predicate.
Out: if $\top$ then F is an invariant,
if $\bot$ then $\neg F$ is reachable.

$\boldsymbol{p} \leftarrow$ FreshVariables($\mathbb{N}^P$)
$\boldsymbol{x} \leftarrow$ FreshVariables($\mathbb{N}^T$)

;; State equation.
$\mathcal{C}(\boldsymbol{p}, \boldsymbol{x}) \leftarrow \underline{m_0}(\boldsymbol{p}) + I \cdot \boldsymbol{x} = \boldsymbol{p}$

;; Is there a potentially reachable marking that contradicts F?
while SAT($\mathcal{C}(\boldsymbol{p}, \boldsymbol{x}) \wedge \neg F(\boldsymbol{p})$) **do**
    ;; If it is the case, extract a model m and a Parikh vector π.
    $m, \pi \leftarrow$ getModel($\mathcal{C}(\boldsymbol{p}, \boldsymbol{x}) \wedge \neg F(\boldsymbol{p})$)
    for all firing sequences ϱ such that $\wp(\varrho) \leqslant \pi$. **do**
        if $\wp(\varrho) = \pi$ **then**
            ;; Formula $\neg F$ is reachable by firing a sequence ϱ such that $\wp(\varrho) \leqslant \pi$.
            return $\bot$
        else
            ;; Refine by blocking all sequences ϱ such that $\wp(\varrho) = \pi$.
            $\mathcal{C}(\boldsymbol{p}, \boldsymbol{x}) \leftarrow \mathcal{C}(\boldsymbol{p}, \boldsymbol{x}) \wedge$ generateConstraint($\mathcal{C}, \pi$)

;; Refinements are sufficient to prove F invariant.
return $\top$

1.5.6 Optimizations

The previous model checking methods can be accelerated by using reduction techniques: structural reductions, partial order reductions, slicing, etc. We give an overview of these different techniques.

Structural Reductions

Structural reductions refer to a collection of well-known transformations that can reduce the size of a net while preserving the property that we want to check, with the result of accelerating the model checking phase. These approaches can be seen as *reduction theorems* [Lip75; CL98], that allow deducing properties of an initial model (N) from properties of a simpler, coarser-grained version (N^R). This technique has become a conventional optimization integrated into several model checking tools [BLD18; Bøn+19; Thi21].

The concept was introduced by Berthelot [BL85; Ber87] and Murata [MK80; Mur89], mainly for removing redundant places and transitions. Reduction is performed by applying successive graph transformations, on subparts of the net satisfying given structural conditions. The reduction rules preserve the properties of interest, such as liveness, boundedness, or deadlock-freedom. Next, this approach has also been extended for LTL model checking [ES01]. Concerning reachability, Thierry-Mieg [Thi20; Thi21] recently proposed a large set of reductions rules, which are sound when places of interest (the support of the formula) are untouched; something started earlier in the tool TAPAAL [Jen+16]. Regarding Petri nets extensions, structural reductions have also been adapted for colored nets [EHP05; HP06].

Partial Order Reductions and Symmetries

The aim of partial order reductions and symmetries is to build a reduced state space preserving the designed properties.

Partial order techniques allow to reduce the part of combinatorial explosion due to the representation of parallelism by interleaving. This is done by identifying transitions that are independent of each other; meaning the order in which they are fired does not influence the property of interest. In the case of *persistent sets reductions* [Val88; GW94], this is achieved by considering only a subset of enabled transitions within each marking.

The reduction approaches based on symmetries [Sta91; Hub+85; Sch00], make it possible to exploit the symmetries present in the system, in its architecture or in its data, to explore only part of the accessible states while preserving the verification capabilities.

Slicing

We can also mention other approaches where the system is simplified with respect to a given property, for instance by eliminating parts that cannot contribute to its truth value, like with

the *slicing* [Wei84] or *cone of influence* [CGP99] abstractions used in some model checkers. Slicing methods are divided into two categories: *static*, when they do not consider the initial marking, or *dynamic*, otherwise. A variety of slicing algorithms have been proposed, for many types of problems [KKG18]: CTL*, LTL, liveness, boundedness, reachability, etc.

Regarding reachability properties, Rakow [Rak12] proposed a static algorithm that takes the support of the property of interest as a criterion and gets rid of some subparts of the net that do not affect such places. Finding such "parts" (places and transitions) in a Petri net is not always easy, especially when the formula involves many places.

Decomposition in Network of Automata

Another example of valuable optimization is the decomposition of nets into automata networks, i.e., sets of sequential components (such as finite-state machines). Such components execute asynchronously, synchronize with each other, and exhibit the same global behavior as the original Petri net.

In this thesis, we consider Nested-Unit Petri Nets (NUPNs, for short) [Gar15; Gar19] that are an extension of Petri nets for expressing *locality* and *hierarchy* properties of concurrent systems. Places can be grouped into units that express sequential components. Units can be recursively nested to reflect both the concurrent and the hierarchical nature of complex systems. The concept of NUPN is not recent (see, e.g., [GS90]), but it has been adopted by recent Petri net analysis tools, which increase their performance by exploiting the NUPN information.

1.6 Well-Formed Nets

During our presentation, we considered standard Petri nets. But, the Model Checking Contest also provides in its benchmark *well-formed* nets (also called *symmetric* nets) [Chi+91; Chi+93; IR93] that are a restriction of *high-level* Petri nets [Jen83].

Well-formed nets are standard Petri nets, where information is attached to each token, and this information can be inspected or modified when a transition fires. All types have finite domains and expressions are limited to a restricted set of operators. Note that well-formed nets have the same modelling power as general *colored* Petri Nets [Jen81; Jen87]—that is why in the MCC, as we will do, often refer to colored nets for well-formed nets.

Some model checking techniques and optimizations are specific to colored nets, such as some structural reductions [EHP05; HP06], or use of symmetries [Hub+85; Jen96] as mentioned before. But another interesting method is the use of the *skeleton* [Vau87] that simply turns the colored tokens into "standard tokens". The obtained net is an over-approximation of the initial one, since some transitions may become enabled when not considering the constraints on colored tokens. But still, this approximation is much simpler to analyze and may be sufficient for proving invariants using the state equation as in Sect. 1.5.4. We can also find some works

about deadlock-preserving skeleton [Fin92], or more recent works on the ACTL* logic in the model checker LoLA [WW22].

We will not consider colored Petri nets in the rest of this work. In fact, actual model checkers mainly rely on an *unfolding* to standard Petri nets (P/T nets) when dealing with colored nets. A naive approach is often enough. We can unfold each colored place into a collection of P/T net places for each possible color; and similarly for transitions, considering all possible combinations of the input places. While the size of the unfolded net may be exponentially larger than its colored counterpart, many works have proposed more elaborated unfolding algorithms [Mäk01; KLP06; LHY12; Dal20; Bil+21], leading to standalone tools that can be used as preprocessors, such as MARIA [Mäk01], CPN-AMI [Ham+06] or mcc [Dal20].

1.7 Comparison with Thesis Contributions

In this chapter, we have laid the foundations of this work. We now discuss how the contributions of this thesis relate to what we have presented.

Model Checker Development

One of the objectives of this thesis is to propose new model checking methods. To this end, we developed a model checker, SMPT, used as a testbed in this work. This tool includes all the techniques previously presented (random walk, BMC, k-induction, state equation, etc.), except for CEGAR. In the absence of CEGAR, we propose an adaptation of the Property Directed Reachability method (see Chapter 2), which also starts from a state space over-approximation, and consecutively blocks potential counter-examples. The aim here is to answer queries hitherto unanswerable by state-of-the-art tools while providing checkable certificates. As mentioned, we are interested in the reachability problem (expressed as linear formulas over the marking of places), but we also address a specific subproblem in Chapter 6, that is the concurrent places problem. This problem turns out to be useful for the decomposition into NUPNs, which is itself used in the MCC benchmark for safe Petri nets. For this problem, we have developed a specific tool called Kong.

A recent approach to model checking is the development of preprocessors that can be used with any out-of-the-box model checker. One example is the work of Thierry-Mieg, who experimented at the Model Checking Contest [Kor+23] by connecting the ITS-Tools preprocessing phase to various model checkers (including LoLA). The idea is that the preprocessor either returns a verdict to the query; or returns a simpler problem, for which any model checker can try to answer. Our work on polyhedral reduction is in line with this philosophy. For example, given a formula F to be checked on a net N, our tool Octant (see Chapter 5) can return a simpler formula F' to be checked on reduced net N'. This simpler problem be handled by any existing verification tool without modifying its algorithms.

Polyhedral Reduction

Like structural reductions, polyhedral reductions that we present in this thesis can be interpreted as an example of reduction theorem [Lip75], that allows to deduce properties of an initial model N from properties of a simpler, coarser-grained version N'. But our notion of reduction is more complex and corresponds to the one pioneered by Berthelot [Ber87], but with the addition of some predicate of linear constraints E that permits to rebuild the reachable markings of N, knowing only the ones of N'. A difference with previous works on structural reductions, e.g., [Ber87], is that our approach is not tailored to a particular class of properties—such as the absence of deadlocks—but could be applied to more general problems. While these works are related, they also mainly focus on reductions where one can group a sequence of transitions into a single, atomic action. Hence, in our context, they correspond to a restricted class of reductions, similar to a subset of the agglomeration rules presented in Chapter 3.

What is more, polyhedral reductions are interesting since it means that we can apply more aggressive reduction techniques than, say, slicing [Rak12; Llo+17; KKG18], cone of influence [CGP99], or other methods [GRV08; KBJ21] that seek to remove or gather together places that are not relevant to the property we want to check (and so cannot contribute to its truth value). We do not share this restriction in our approach, since we reduce nets beforehand and can therefore reduce places that occur in the initial property. We could argue that approaches similar to slicing only simplify a model with respect to a formula, whereas we simplify both the model and the formula using our new method. This is more efficient when we need to check several properties on the same model and, in any case, nothing prevents us from applying slicing techniques on the result of our reduction. Concerning the slicing or cone of influence abstractions used in some model checkers, it is not always easy to find such "parts" (places and transitions) in a Petri net, especially when the formula involves many places. This is not a problem with our approach, since we can always abstract away a place, as long as its effect is preserved in the predicate E.

Relation to Presburger Arithmetic

Finally, in this thesis, we exhibit results on the relation between Petri nets and Presbuger arithmetic, which is why we have paid particular attention to related results. Of course, our SMT methods all rely on Linear Integer Arithmetic (LIA) predicates. But a first concrete example is the Property Directed Reachability method that we adapt to Petri nets in Chapter 2. This semi-decision procedure provides certificates of invariance expressed as Presburger formulas. This contribution is in accordance with Theorem 1.2 of Leroux.

We also show that the main philosophy of polyhedral reduction is to capture "flat" sub-parts of nets (with Presburger-definable reachability sets). This idea is clearly highlighted in Chapter 7, where we propose an automated procedure to prove that some polyhedral reduction is correct.

This chapter is part of an **educational project**, called uSMPT, targeting Master and PhD students. The goal of this project is to showcase the application of SMT methods in system verification by developing a Petri net model checker for the reachability problem.

https://github.com/nicolasAmat/uSMPT

Chapter 2

Computing Invariance Certificates

With Property Directed Reachability

> It is fair to state, that in this digital era
> correct systems for information
> processing are more valuable than gold.
>
> Henk Barendregt

In this chapter, we propose a semi-decision procedure for checking reachability properties on Petri nets that is based on the Property Directed Reachability (PDR) method. We define three different versions that vary depending on the method used for abstracting possible witnesses, and that can handle problems of increasing difficulty. For each method, we present their limitations with a small example. We have implemented our methods in our model checker SMPT and give empirical evidence that our approach can handle problems that are difficult or even impossible to check with current state-of-the-art tools.

2.1 Introduction

This chapter introduces the first contributions of this thesis by describing a new semi-decision procedure for the reachability problem in Petri nets. We have chosen it as a starting point in order to introduce how to solve our problem of interest; while the following chapters focus on accelerating the computation. Note that the results of this chapter are not a prerequisite for understanding the rest of the thesis.

Context. While BMC is the right choice when we try to find witnesses, it usually performs poorly when we want to check an invariant property, $\mathrm{AG}\,\neg F$ true, or equivalently checking F not reachable, $\mathrm{EF}\,F$ false. Some techniques are better suited to prove *inductive invariants*

in a transition system; that is a property that is true initially and stays true after firing any transition.

But as mentioned in the previous chapter, a practical consequence of the Ackermannian complexity of the reachability problem is that we should not expect to find a one-size-fits-all algorithm that could be usable in practice. A better strategy is to try to improve the performances on some cases—for example by developing new tools, or optimizations, that may perform better on some examples—or try to improve "expressiveness"—by finding algorithms that can manage new cases, that no other tool can handle.

Challenge. With this work, we seek improvements in terms of both *performance* and *expressiveness*. We also target what we consider to be a difficult, and less studied area of research: procedures to prove that a property is an invariant, that works on unbounded nets or when the state space cannot be fully explored. We also focus on verifying "genuine" reachability constraints, which are not instances of a coverability problem. These properties are seldom studied in the context of unbounded nets. Interestingly enough, this work provides a simple explanation of why coverability problems are also "simpler" in the case of PDR, what we associated with the notion of *monotonic formulas* (see Definition 1.8).

Proposal. We propose a new semi-decision procedure for checking reachability properties on generalized Petri nets, meaning that we do not impose constraints on the weights of the arcs and do not require a finite state space. As formally presented in Sect. 1.3, we also consider a generalized notion of reachability, in the sense that we do not only check the reachability of a given state but also if it is possible to reach a marking that satisfies a combination of linear constraints between places, such as $(p_0 + p_1 = p_2 + 2) \wedge (p_1 \leqslant p_2)$ for example. Another interesting feature of our approach is that we are able to return a *certificate of invariance*, in the form of an *Presbuger inductive invariant*, when we find that a constraint is true on all the reachable markings. To the best of our knowledge, there is no other tool able to compute such certificates for Petri nets in the general case.

Our approach is based on an extension of the Property Directed Reachability (PDR) method, originally developed for hardware model checking [Bra11; Bra12], to the case of Petri nets. One of the key steps in PDR is to generalize potential witnesses found by the SMT solver. A generalization is defined as a linear subset of antecedents of a certain reachability property F. We actually define three variants of our algorithm that vary based on the method used for generalizing possible witnesses and can handle problems of increasing difficulty.

Let us return to our two main objectives. Concerning performances, we propose a method based on a well-tried symbolic technique, PDR, that has proved successful with unbounded model checking and when used together with SMT solvers [Cim+14; HB12]. Concerning expressiveness, we define a small benchmark of "difficult nets": a set of synthetic examples, representative of patterns that can make the reachability problem harder.

Outline and Contributions. The chapter is organized as follows. We define additional background material in Sect. 2.2. Section 2.3 describes our decision method, based on PDR and SMT solvers, for checking the satisfiability of linear invariants over the reachable states of a Petri net. Our method builds sequences of incremental invariants using both a property that we want to disprove, and a stepwise approximation of the reachability relation. It also relies on a generalization step where we can abstract possible "bad states" into clauses that are propagated in order to find a counter-example or to block inconsistent states.

We describe a first generalization method (Sect. 2.3.2), based on the upward closure of markings, that can deal with coverability properties. We propose a new, dual variant (Sect. 2.3.3), based on the concept of *hurdles* [Hac76], that is without restrictions on the properties. In this method, the goal is to block bad sequences of transitions. We show how this approach can be further improved by defining a notion of saturated transition sequences (Sect. 2.3.4), at the cost of adding universal quantification in our SMT problems. But in Sect. 2.3.5, we show that some problems still cannot be solved by this method (hence the algorithm may never terminate), whatever the generalization method used.

We have implemented our approach in our tool SMPT, and we compare it in Sect. 2.4 with other existing tools participating in the Model Checking Contest. In this context, one of our contributions is the definition of a set of difficult nets that characterizes classes of increasingly difficult reachability problems.

2.2 Linear Reachability Constraints

In this section, we provide additional background to reason about nets using Presburger arithmetic. We first present *invariance certificates* (Sect. 2.2.1), which can be computed using our adaptation of PDR. To achieve this goal, we define additional linear predicates for representing sequences (Sect. 2.2.2), in order to generalize scenarios leading to some property F that is the crux of the PDR method (Sect. 2.2.3).

2.2.1 Invariance Certificates

In Sect. 1.5.3, we showed that it is possible to characterize inductive predicates using our logical framework. Indeed, F is inductive (Definition 1.11) if and only if the QF-LIA formulas (i) $F(m_0)$ and (ii) $F(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \implies F(\boldsymbol{p'})$ are valid. As a consequence, a sufficient condition for a predicate F to be invariant is to have both conditions (i) and (ii); conditions that can be checked using an SMT solver. Unfortunately, the predicates that we need to check are often not inductive. In this case, the next best thing is to try to build an inductive invariant, say R, such that $[\![R]\!] \subseteq [\![F]\!]$ (i.e., $\models R(\boldsymbol{p}) \implies F(\boldsymbol{p})$ or equivalently, to simplify, $R(\boldsymbol{p}) \wedge \neg F(\boldsymbol{p})$ `unsat`). This predicate provides a certificate of invariance that can be checked independently.

Proposition 2.1 (Invariance Certificate)**.** *A sufficient condition for a given predicate F to be invariant on (N, m_0) is to exhibit a Presburger predicate R that is (i) inductive: $R(m_0)$ `valid` and $R(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \wedge \neg R(\boldsymbol{p'})$ `unsat`; and (iii) entails F, that is, $R(\boldsymbol{p}) \wedge \neg F(\boldsymbol{p})$ `unsat`.*

This result is in line with Theorem 1.2 proved by Leroux [Ler09; Ler10], which states that when a predicate $\neg F$ is not reachable (i.e., F is invariant) there exists a Presburger inductive invariant that contains m_0 but is disjoint from $[\![\neg F]\!]$. This result does not explain how to effectively compute such an invariant. Moreover, in our case, we provide a method that works with general linear predicates, and not only with single configurations. On the other side of the coin, given the known results about the complexity of the problem, we do not expect our procedure to be complete in the general case.

2.2.2 Expressing Sequences

While reachable states are computed by adding a linear combination of "displacements" (vectors in $\mathbb{Z}^P$), the set $R(N, m_0)$ is not necessarily semilinear or, equivalently, definable using Presburger arithmetic [GS66; Ler09]. This is a consequence of the constraint that transitions must be enabled before firing. But there is still some structure to the set $R(N, m_0)$, like for instance the following monotonicity constraint:

$$\forall m \in \mathbb{N}^P \,.\, m_1 \xrightarrow{\varrho} m_2 \quad \text{implies} \quad m_1 + m \xrightarrow{\varrho} m_2 + m \tag{H1}$$

We have other such results, such as with the notion of *hurdle* [Hac76]. Just as $\mathrm{Pre}(t)$ is the smallest marking for which a given transition t is enabled, there is a smallest marking at which a given firing sequence ϱ can be fired. This marking, denoted by $H(\varrho)$, has a simple inductive definition:

$$H(t) \triangleq \mathrm{Pre}(t) \quad \text{and} \quad H(\varrho_1 \cdot \varrho_2) \triangleq \max\left(H(\varrho_1), H(\varrho_2) - \Delta(\varrho_1)\right) \tag{H2}$$

Given this notion of hurdles, we obtain that $m \xrightarrow{\varrho} m'$ if and only if (1) the sequence ϱ is enabled: $m \geqslant H(\varrho)$, and (2) $m' = m + \Delta(\varrho)$. We use this result in the second variant of our method.

We can go a step further and characterize a necessary and sufficient condition for firing the sequence $\varrho.\varrho^k$, meaning firing the same sequence more than once. We call this the *saturation* of the sequence ϱ, where k is a *saturation variable*. The saturation of a sequence ϱ is also called *acceleration* in [FL02], *meta-transitions* in [BW94; Boi98], or *exact widening* in the field of abstract interpretation.

Given $\Delta(\varrho)$, a place p with a negative displacement (say $-d$) means that we "loose" d token each time we fire ϱ. Hence, we should budget d tokens in p for each new iteration. On the opposite, nothing is needed for places with a positive displacement, which accrue tokens.

Therefore, we have $m \xrightarrow{\varrho} \xrightarrow{\varrho^k} m'$ if and only if (1) $m \geqslant H(\varrho) + k \cdot \max(\mathbf{0}, -\Delta(\varrho))$, and (2) $m' = m + (k+1) \cdot \Delta(\varrho)$. Equivalently, if we denote by Δ^+ the "positive" part of mapping Δ, such that $\Delta^+(p) \triangleq 0$ when $\Delta(p) \leqslant 0$ and $\Delta^+(p) \triangleq \Delta(p)$ otherwise, we have:

$$H(\varrho^{k+1}) = \max\left(H(\varrho), H(\varrho) - k \cdot \Delta(\varrho)\right) = H(\varrho) + k \cdot (-\Delta(\varrho))^+ \tag{H3}$$

Examples

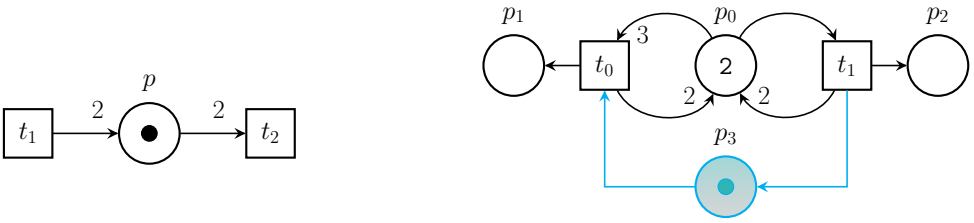


Fig. 2.1 Two examples of Petri nets: Parity (left) and PGCD [ADH23c] (right).

We give two simple examples of unbounded nets in Fig. 2.1, which are both part of our benchmark. Parity (left) has a single place, hence its state space can be interpreted as a subset of $\mathbb{N}$: with an initial marking of 1, this is exactly the set of odd numbers (and therefore state (0) is not reachable). We are in a special case where the set $R(N, m_0)$ is semilinear. For instance, it can be seen as a solution to the constraint $\exists k \,.\, (p = 2k + 1)$, or equivalently $p \equiv 1 \pmod 2$. However, it cannot be expressed with a linear constraint involving only the variable p without quantification or modulo arithmetic. This example can be handled by most of the tools used in our experiments, e.g., with the help of k-induction.

In the net PGCD (right), transitions t_0/t_1 can decrement/increment the marking of p_0 by 1. Nonetheless, with this initial state, it is the case that the number of occurrences of t_0 is always less than the one of t_1 in any feasible sequence ϱ. Hence, the two predicates $p_0 \geqslant 2$ and $p_2 \geqslant p_1$ are valid invariants. (Since some tools do not accept literals of the form $p \geqslant q$, we added the "redundant" place p_3, so we can restate our second invariant as $p_3 \geqslant 1$.) These invariants cannot be proved by reasoning only on the displacements of traces (using the state equation) and are already out of reach for both tools LoLA and TAPAAL.

2.2.3 Generalizing Scenarios

In the next section, we show how to (potentially) find such certificates using an adaptation of the PDR method. An essential component of PDR is to abstract a "scenario" leading to the model of some property $F(\boldsymbol{p})$—say a sequence $m \xrightarrow{\varrho} m'$ with $m' \models F$—into a predicate that

contains m (and potentially many more similar scenarios). More generally, a *generalization* of the trio (m, ϱ, F) is a predicate G satisfied by m that corresponds to a subset of the antecedents of F.

Definition 2.1 (Generalization)**.** *Assume that we have a trio (m, ϱ, F) such that $m \xrightarrow{\varrho} m'$ and $m' \models F$. We call such a trio a scenario and we say that a predicate G is a generalization of (m, ϱ, F) if and only if: (1) $m \models G$; and (2) for every marking $m_1 \models G$ there is some sequence ϱ' such that $m_1 \xrightarrow{\varrho'} m_2$ and $m_2 \models F$, meaning (m_1, ϱ', F) is also a scenario.*

We can use properties (H1)–(H3), defined earlier, to build three generalizations.

Theorem 2.2 (Three Generalizations)**.** *Assume that we have $m \xrightarrow{\varrho} m'$ and that $m' \models F$. Each property (H1)–(H3) leads to a generalization of scenario (m, ϱ, F).*

(G1) *If property F is monotonic then $\mathrm{GEQ}_m(\boldsymbol{p})$ is a generalization of (m, ϱ, F).*

(G2) *$\mathrm{GEQ}_{H(\varrho)}(\boldsymbol{p}) \wedge F(\boldsymbol{p} + \Delta(\varrho))$ is a generalization of (m, ϱ, F).*

(G3) *Assume that a, b are mappings of $\mathbb{N}^P$ such that $a = H(\varrho)$ and $b = (-\Delta(\varrho))^+$, with the notations used in (H3). Then*

$$\exists k \;.\; \left(\left[\bigwedge_{i \in 1..n} (p_i \geqslant a(i) + k \cdot b(i)) \right] \wedge F(\boldsymbol{p} + (k+1) \cdot \Delta(\varrho)) \right)$$

is a generalization of (m, ϱ, F).

Proof. Assume a scenario (m, ϱ, F) and m' a marking such that $m \to m'$ and $m' \models F$. We prove each generalization separately.

(G1) Assume that F is monotonic and take a marking m_1 such that $m_1 \models \mathrm{GEQ}_m(\boldsymbol{p})$. Property (H1) implies that there exists $m_2 \geqslant m'$ such that $m_1 \xrightarrow{\varrho} m_2$. Since F is monotonic, we also have $m_2 \models F$.

(G2) Take a marking m_1 such that $m_1 \models \mathrm{GEQ}_{H(\varrho)}(\boldsymbol{p}) \wedge F(\boldsymbol{p} + \Delta(\varrho))$. By construction of (H2) there exists some marking m_2 such that $m_1 \xrightarrow{\varrho} m_2$. Since $\models F(m_1 + \Delta(\varrho))$ and $m_2 = m_1 + \Delta(\varrho)$, we also have $m_2 \models F$.

(G3) Let ϕ the generalization and take a marking m_1 such that $m_1 \models \phi$. By construction of (H3), there exists some $k \geqslant 0$ and marking m_2 such that $m_1 \xrightarrow{\varrho^{k+1}} m_2$. As previously, since $m_2 = m_1 + (k+1) \cdot \Delta(\varrho)$, we have $m_2 \models F$.

For each generalization, it is clear that m is a valid candidate for m_1 since $m \xrightarrow{\varrho} m'$. □

Property (G3) is the first and only instance of a linear formula using an extra variable, k, that is not in P. The result is still a linear formula though, since we never need to use the product of two variables. This generalization is used when we want to "saturate the sequence

ϱ". This is the only situation where we may need to deal with quantified Presburger formulas. Another solution would be to replace each quantification with the use of modulo arithmetic, but this operation may be costly and could greatly increase the size of our formulas. It would also not cut down the complexity of the SMT problems [Coo72].

2.3 Property Directed Reachability

As mentioned, some symbolic model checking procedures, such as BMC [Bie+99] or k-induction [SSS00], are a good fit when we try to find counter-examples on infinite-state systems. Unfortunately, they may perform poorly when we want to check an invariant. In this case, adaptations of the PDR method [Bra11; Bra12] (also known as IC3, for "Incremental Construction of Inductive Clauses for Indubitable Correctness") have proved successful.

We assume that we start with an initial state m_0 satisfying a linear property, $\mathbb{I}$, and that we want to prove that property $\mathbb{P}$ is an invariant of the marked net (N, m_0). (We use blackboard bold symbols to distinguish between parameters of the problem and formulas that we build for solving it.) When checking for the reachability from the initial state, we can simply choose $\mathbb{I}$ such that $[\![\mathbb{I}]\!] = \{m_0\}$.

We define $\mathbb{F} \triangleq \neg\mathbb{P}$ as the "set of feared events"; such that $\mathbb{P}$ is not an invariant if we can find m in $R(N, m_0)$ such that $m \models \mathbb{F}$. We may freely assume that $\mathbb{F}$ is in Disjunctive Normal Form (DNF). However, to simplify the presentation, we focus on a single cube, meaning that $\mathbb{P}$ is a clause.

PDR is a combination of induction, over-approximation, and SAT or SMT solving. The goal is to build an incremental sequence of predicates $F_0, \ldots, F_k$ that are "inductive relative to stepwise approximations" such that $m \models F_i$ and $m \to m'$ entails $m' \models F_{i+1}$, but not $m' \models \mathbb{F}$. The method stops when it finds a counter-example, or when we find that one of the predicates F_i is inductive.

We adapt the PDR approach to Petri nets, using linear predicates and SMT solvers for the QF-LIA and LIA logics in order to learn, generalize, and propagate new clauses. The most innovative part of our approach is the use of specific "generalization algorithms" that take advantage of the Petri nets theory, like the use of hurdles for example.

2.3.1 Description of the Algorithm

Our implementation follows closely the algorithm for IC3 described in [Bra11; Bra12], and we only give a brief sketch of it.

The main function, Prove (Algorithm 2.1), computes an *Over Approximated Reachability Sequence* (OARS) $(F_0, \ldots, F_k)$ of linear predicates, called *frames*, with variables in $\boldsymbol{p}$. An OARS meets the following constraints: (1) it is monotonic: $F_i \wedge \neg F_{i+1}$ `unsat` for $0 \leqslant i < k$; (2) it

contains the initial states: $\mathbb{I} \wedge \neg F_0$ `unsat`; (3) it does not contain feared states: $F_i \wedge \mathbb{F}$ `unsat` for $0 \leqslant i \leqslant k$; and (4) it satisfies *consecution*: $F_i(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \wedge \neg F_{i+1}(\boldsymbol{p'})$ `unsat` for $0 \leqslant i < k$.

By construction, each frame F_i in the OARS is defined as a set of clauses, $\mathrm{CL}(F_i)$, meaning that F_i is built as a formula in CNF: $F_i \triangleq \bigwedge_{cl \in \mathrm{CL}(F_i)} cl$. We also enforce that $\mathrm{CL}(F_{i+1}) \subseteq \mathrm{CL}(F_i)$ for $0 \leqslant i < k$, which means that the monotonicity property between frames is trivially ensured.

The body of function Prove (Algorithm 2.1) contains a *main iteration* (line 6) that increases the value of k (the number of levels of the OARS). At each step, we enter a second, minor iteration (line 2 in function Strengthen, Algorithm 2.2), where we generate new minimal inductive clauses that will be propagated to all the frames. Hence, both the length of the OARS, and the set of clauses in its frames, increase during computation. The procedure stops when we find an index i such that $F_i = F_{i+1}$ (lines 12–13 in Prove). In this case, we know from (2) and (3) that F_i is an inductive invariant satisfying $\mathbb{P}$. We can also stop during the iteration if we find a counter-example (a model m of $\mathbb{F}$); see lines 8–9 of Strengthen. In this case, we can also return a trace leading to m.

Algorithm 2.1 Prove($\mathbb{I}, \mathbb{F}$: linear predicate)

Ensure: $\bot$ if $\mathbb{F}$ is reachable ($\mathbb{P} \equiv \neg\mathbb{F}$ is not an invariant), otherwise $\top$.

```
1:  if SAT(𝕀(p) ∧ T_EQ(p, p') ∧ 𝔽(p')) then
2:      return ⊥
3:
4:  k ← 1, F_0 ← 𝕀, F_i ← ℙ for all i ⩾ 1
5:
6:  while ⊤ do
7:      if Strengthen(k) then
8:          return ⊥
9:
10:     PropagateClauses(k)
11:
12:     if CL(F_i) = CL(F_{i+1}) for some 1 ⩽ i ⩽ k then
13:         return ⊤
14:
15:     k ← k + 1
```

When we start the first minor iteration, we have $k = 1$, $F_0 = \mathbb{I}$ and $F_1 = \mathbb{P}$. If we have $F_k(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \wedge \mathbb{F}(\boldsymbol{p'})$ `unsat`, it means that $\mathbb{P}$ is inductive, so we can stop and return that $\mathbb{P}$ is an invariant (lines 12–13 of Prove, where condition line 12 trivially holds in this case). Otherwise, we proceed with the strengthen phase, where each model of $F_k(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \wedge \mathbb{F}(\boldsymbol{p'})$ becomes a potential counter-example, or witness, that we need to "block" (lines 3–5 of function Strengthen).

Instead of blocking only one witness, we first generalize it into a predicate that abstracts similar dangerous states (see the call to GeneralizeWitness). This is done by applying one

of the three generalization results in Theorem 2.2. We give more details about this step later. By construction, each generalization is a cube s (a conjunction of literals). Hence, when we block it, we learn new clauses from $\neg s$ that can be propagated to the previous frames.

Algorithm 2.2 StrengthenOneStep(k: current level)

```
1: try
2:     while (m --t--> m') ⊨ F_k(p) ∧ T(p, p') ∧ F(p') do
3:         s ← GeneralizedWitness(m, t, F)
4:         n ← InductivelyGeneralize(s, k − 2, k)
5:         PushGeneralization({(s, n + 1)}, k)
6:     return ⊤
7:
8: catch counter-example
9:     return ⊥
```

Before pushing a new clause, we test whether s is reachable from previous frames. We take advantage of this opportunity to find if we have a counter-example and, if not, to learn new clauses in the process. This is the role of functions InductivelyGeneralize (Algorithm 2.3) and PushGeneralization (Algorithm 2.4).

We find a counter-example (in the call to InductivelyGeneralize) if the generalization from a witness found at level k, say s, reaches level 0 and $F_0(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \wedge s(\boldsymbol{p'})$ is satisfiable (line 1 in InductivelyGeneralize). Indeed, it means that we can build a trace from $\mathbb{I}$ to $\mathbb{F}$ by going through $F_1, \ldots, F_k$.

Algorithm 2.3 InductivelyGeneralize(s : cube, min: level, k: level)

```
1: if min < 0 and sat(F_0(p) ∧ T(p, p') ∧ s(p')) then
2:     raise counter-example
3:
4: for i ← max(1, min + 1) to k do
5:     if sat(F_i(p) ∧ T(p, p') ∧ ¬s(p) ∧ s(p')) then
6:         GeneralizeClause(s, i − 1, k)
7:         return i − 1
8:
9: GeneralizeClause(s, k, k)
10:
11: return k
```

The method relies heavily on checking the satisfiability of linear formulas in QF-LIA, which is achieved with a call to an SMT solver. In each function call, we need to test if predicates of the form $F_i \wedge T \wedge G$ are `unsat` and, if not, enumerate its models. To accelerate the strengthening of frames, we also rely on the `unsat-core` of properties in order to compute a *minimal inductive clause* (MIC).

Our approach is parametrized by a generalization function (GeneralizeWitness) that is crucial if we want to avoid enumerating a large, potentially unbounded, set of witnesses. This can be the case, for example, in line 7 of PushGeneralization. In this particular case, we find a state m at level n (because $m \models F_n$), and a transition t that leads to a problematic clause in F_{n+1}. Therefore, we have a sequence ϱ of size $k - n + 1$ such that $m \xrightarrow{\varrho} m'$ and $m' \models \mathbb{F}$. We consider three possible methods for generalizing the trio $(m, \varrho, \mathbb{F})$, that correspond to properties (G1)–(G3) in Theorem 2.2.

Algorithm 2.4 PushGeneralization(*states*: set of (state, level), k: level)

```
1: while ⊤ do
2:     (s, n) ← from states minimizing n
3:
4:     if n > k then
5:         return
6:
7:     if (m -t-> m') ⊨ F_n(p) ∧ T(p, p') ∧ s(p') then
8:         p ← GeneralizedWitness(m, t, s)
9:         l ← InductivelyGeneralize(p, n − 2, k)
10:        states ← states ∪ {(p, l + 1)}
11:    else
12:        l ← InductivelyGeneralize(s, n, k)
13:        states ← states \ {(s, n)} ∪ {(s, l + 1)}
```

Algorithm 2.5 PropagateClauses(k: level)

```
1: for i ← 1 to k do
2:     for all cl ∈ CL(F_i) do
3:         if ⊭ F_i(p) ∧ T(p, p') ∧ ¬cl(p') then
4:             CL(F_{i+1}) ← CL(F_i) ∪ {cl}
```

Algorithm 2.6 GenerateClause(s : cube, i: level, k: level)

```
1: cl ← ¬ UNSAT-CORE(¬s(p) ∧ F_i(p) ∧ T(p, p') ∧ s(p'))
2:
3: for j ← 1 to i+1 do
4:     CL(F_j) ← CL(F_j) ∪ {cl}
```

2.3.2 State-Based Generalization

A special case of the reachability problem is when the predicate $\mathbb{F}$ is monotonic, meaning that $m_1 \models \mathbb{F}$ entails $m_1 + m_2 \models \mathbb{F}$ for all markings m_1, m_2. A sufficient (syntactic) condition is for $\mathbb{F}$ to be a positive formula with literals of the form $\sum_{i \in I} p_i \geqslant a$. This class of predicates coincides

with what we called a *coverability property* in Sect. 1.3 (Definition 1.8), for which there exists specialized verification methods (see, e.g., [Fin91; FHK21]).

By property (G1), if we have to block a witness m such that $m \xrightarrow{\varrho} m'$ and $m' \models \mathbb{F}$, we can as well block all the states greater than m. Hence, we can choose the predicate GEQ_m to generalize m. This is a very convenient case for verification and one of the optimizations used in previous works on PDR for Petri nets [Klo+13; KBJ21]. First, the generalization is very simple, and we can easily compute a MIC when we block predicate GEQ_m in a frame. Also, we can prove the completeness of the procedure when $\mathbb{F}$ is monotonic. An intuition is that it is enough, in this case, to check the property on the minimal coverability set of the net, which is always finite [Fin91]. The procedure is also complete for finite transition systems. These are the only cases where we have been able to prove that our method always terminates.

2.3.3 Transition-Based Generalization

We propose a new generalization based on the notion of hurdles. This approach can be used when $\mathbb{F}$ is not monotonic, for example when we want to check an invariant that contains literals of the form $p = k$ (e.g., the reachability of a fixed marking) or $p \geqslant q$ with p and q being two places.

Assume that we need to block a witness scenario of the form $m \xrightarrow{\varrho} m'$ and $m' \models s$. Typically, s is a cube in $\mathbb{F}$, or a state resulting from a call to PushGeneralization. By property (G2), we can as well block all the states satisfying $G_\varrho(\boldsymbol{p}) \triangleq \mathrm{GEQ}_{H(\varrho)}(\boldsymbol{p}) \wedge s(\boldsymbol{p} + \Delta(\varrho))$. This generalization is interesting when property s does not constrain all the places, or when we have few equality constraints. In this case, G_ϱ may have an infinite number of models. It should be noted that using the duality between "feasible traces" and hurdles is not new. For example, it was used recently [FHK21] to accelerate the computation of coverability trees. Nonetheless, to the best of our knowledge, this is the first time that this generalization method has been used with PDR.

2.3.4 Saturated Transition-Based Generalization

We still assume that we start from a witness scenario $m \xrightarrow{\varrho} m'$ and $m' \models s$. Our last method relies on property (G3) and allows us to consider several iterations of ϱ. If we fix the value of k, then a possible generalization is $G^k_\varrho \triangleq (\bigwedge_{i \in 1..n}(p_i \geqslant a(i) + k \cdot b(i))) \wedge s(\boldsymbol{p} + (k+1) \cdot \Delta(\varrho))$, where a, b are the mappings of $\mathbb{N}^P$ defined in Theorem 2.2. (Note that $G^1_\varrho = G_\varrho$.) More generally the predicate $G^{\leqslant k}_\varrho \triangleq G^1_\varrho \vee \cdots \vee G^k_\varrho$ is a valid generalization for the scenario (m, ϱ, s), in the sense that if $m_1 \models G^{\leqslant k}_\varrho$ then there is a trace $m_1 \to^\star m_2$ such that $m_2 \models s$. At the cost of using existential quantification (and therefore a "top-level" universal quantification when we negate the predicate to block it in a frame), we can use the more general predicate $G^\star_\varrho \triangleq \exists k \,.\, G^k_\varrho$, which is still linear and has its support in P.

Saturation Heuristics

In practice, it is not always useful to saturate a trace and, in our implementation, we use heuristics to limit the number of quantification introduced by this operation.

A possible heuristic is to saturate sub-sequences of transitions instead of every transition in some scenario sequence ϱ. In fact, when iterating backward, we can assume that we already performed a generalization of the form: $\varrho_c.\varrho_1^*..\varrho_l^*$. Then, we decide to saturate ϱ_c if one of the following conditions is met: (1) $\mathbb{F}$ is neither monotonic nor anti-monotonic (i.e., $\neg\mathbb{F}$ not monotonic) ; (2) $\mathbb{F}$ is monotonic and $\Delta(\varrho_c, p) > 0$ for all $p \in \mathrm{FV}(\mathbb{F})$; (3) $\mathbb{F}$ is anti-monotonic and $\Delta(\varrho_c, p) < 0$ for all $p \in \mathrm{FV}(\mathbb{F})$; or (4) for all p such that $H(\varrho_1..\varrho_l, p) \neq 0$ we have $\Delta(\varrho_c, p) > 0$. Informally, we do not saturate ϱ_c if (1') $\mathbb{F}$ is monotonic (resp. anti-monotonic) and ϱ_c decrease (resp. increase) the number of tokens in at least one place in the support of $\mathbb{F}$; and (2') firing ϱ_c an infinite number of times does not lead to an infinite firing of $\varrho_1^*..\varrho_l^*$.

Actually, nothing prevents us from mixing our different kinds of generalization together, and there is still much work to be done in order to find good tactics in this case.

2.3.5 Incompleteness

We know examples of invariants where the PDR method does not terminate except when using saturation. A simple example is the net Parity, used as an example in Sect. 2.2.2, with the invariant $\mathbb{P} \triangleq (p \geqslant 1)$. In this case, $\mathbb{F} \equiv \neg\mathbb{P} \equiv (p = 0)$. Hence, we are looking for witnesses such that $m \to^\star 0$. The simplest example is $p{*}2 \xrightarrow{t_2} p{*}0$, which corresponds to the "blocking clause" $p \neq 2$. In this case, we have $H(t_2) = p{*}2$ and $\Delta(t_2) = -p{*}2$. Hence, the transition-based generalization is $(p \geqslant 2) \wedge (p - 2 = 0) \equiv (p = 2)$, which does not block new markings. At this point, we try to block $(p = 0) \vee (p = 2)$. The following minor iteration of our method will consider the witness $4 \xRightarrow{t_2.t_2} 0$, etc. Hence, after k minor iterations, we have $F_k \equiv (p \neq 0) \wedge (p \neq 2) \wedge \cdots \wedge (p \neq 2k)$. If we saturate t_2, we find in one step that we should block $\exists k \,.\, (p - 2 \cdot (k + 1) = 0)$. This is enough to prove that $(p \geqslant 1)$ is an invariant as soon as the initial marking is an odd number.

This example proves that PDR is not complete, without saturation, in the general case. Even though example Parity is extremely simple, it is also enough to demonstrate the limit of our method without saturation. Indeed, when we only allow unquantified linear predicates with variables in P, it is not possible to express all the possible semilinear sets in $\mathbb{N}^P$. (We typically miss some periodic sets.)

However, the saturation is still not sufficient to obtain a complete procedure with PDR. A small example is the inverse net (we invert all the arcs), say N^{-1}, of the Hopcroft and Pansiot marked net, say (N, m_0), depicted Fig. 1.2.

A basic property of inverse nets (Full reflection theorem in [SJJ91]) is that for any marking m we have $m \in R(N, m_0)$ if and only if $m_0 \in R(N^{-1}, m)$. As a corollary, given a marking m if $m \notin R(N, m_0)$ then $m_0 \notin R(N^{-1}, m)$. The idea of our example is to choose an initial marking

m'_0 for N^{-1} that is not reachable in N from m_0. Hence, m_0 is also not reachable in (N^{-1}, m'_0); and the whole reachability set of (N, m_0) corresponds to witnesses of $\mathbb{F} \equiv \underline{m_0}$.

We recall the (non Presburger-definable) reachability set of Hopcroft and Pansiot net, (N, m_0):

$$\left\{ (p_0, p_1, p_2, p_3, p_4) \in \mathbb{N}^5 \;\middle|\; \begin{array}{l} (p_0 = 1 \wedge p_3 = 0 \wedge 1 \leqslant p_1 + p_2 \leqslant 2^{p_4}) \\ \vee\ (p_0 = 0 \wedge p_3 = 1 \wedge 1 \leqslant p_1 + 2p_2 \leqslant 2^{p_4+1}) \end{array} \right\}$$

Assume that $m'_0 \triangleq p_0{*}0\ p_1{*}1\ p_2{*}1\ p_3{*}1\ p_4{*}0$ is the initial marking of the inverse net N^{-1} (see Fig. 2.2). By construction, m'_0 does not belong to $R(N, m_0)$. Hence, $\neg\underline{m_0}$ is an invariant on (N^{-1}, m'_0), and we can consider $\mathbb{F} \equiv \underline{m_0}$ as "feared states". Now consider the minor iteration (line 2) of function Strengthen. This iteration stops when all the witnesses leading to $\mathbb{F}$ in one step from $\neg\mathbb{F}$ are blocked. This set of states corresponds to the reachability set of (N, m_0), which is not Presburger-definable. Then, no Presburger generalization (as we proposed) can be sufficient to obtain a complete PDR procedure because this minor iteration never terminates.

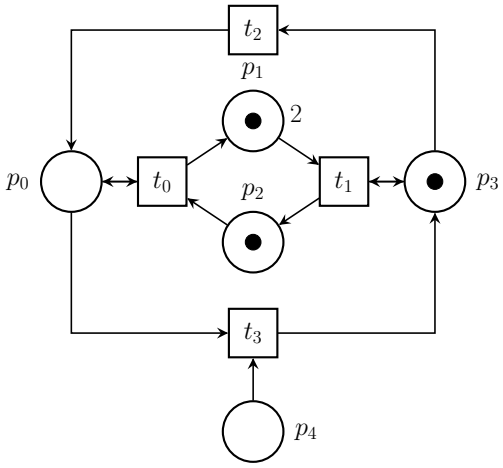


Fig. 2.2 Inverse Hopcroft and Pansiot net, (N^{-1}, m'_0), with $m'_0 \notin R(N, m_0)$.

2.4 Experimental Results

We have implemented our complete approach in our tool SMPT, which relies on the SMT solver z3 to answer `sat` and `unsat-core` queries. In this section, we propose an experimental evaluation divided into two considerations, expressiveness (Sect.2.4.1) and performance (Sect. 2.4.2). We also report on the tool output in Sect. 2.4.3, which provides checkable invariance certificates when the invariant is true.

2.4.1 Evaluation on Expressiveness

It is difficult to find benchmarks with unbounded Petri nets. To quote Blondin et al. [BHO21], "due to the lack of tools handling reachability for unbounded state spaces, benchmarks arising in the literature are primarily coverability instances". It is also very difficult to randomly generate a true invariant that does not follow, in an obvious way, from the state equation. For this reason, we decided to propose our own benchmark, made of five synthetic examples of nets, each with a given invariant. This benchmark is freely available and has been partly joined to the benchmark used in the MCC [ADH23a; ADH23b; ADH23c].

Our benchmark is made of deceptively simple nets that have been engineered to be difficult or impossible to check with current techniques. We already depicted our two first examples in Fig. 2.1. We display all our other examples in Figs. 2.3, 2.4 and 2.5.

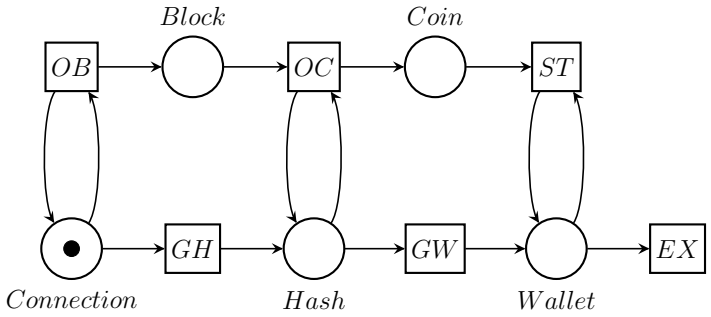


Fig. 2.3 CryptoMiner net [ADH23a] with $\mathbb{P} \triangleq \neg(Block = 4 \wedge Connection = 1 \wedge Coin = 10)$

We give a brief description of the nets composing our "reachability" benchmark (except for Parity and PGCD from Fig. 2.1 already described previously). Each of our examples is quite small, with less than 10 places or transitions, and is representative of patterns that can make the reachability problem harder: the use of self-loops; dead transitions that cannot be detected with the state equation; weights that are relatively prime; etc. Also, most of our examples can be turned into families of nets using parameters such as the initial marking, weights on the arcs, or by adding copies of a subnet.

- **CryptoMiner** [ADH23a] (Fig. 2.3) describes the, simplified, daily schedule of someone mining bitcoins. The net is composed of two disjoint state machines synchronized by self-loops (trivial cycles of weight 1). Removing the self-loops does not modify the incidence matrix, and so does not change the solutions of the state equation. The difficulty when analyzing this net lies in the presence of constraints that cannot be derived from the state equation alone. For instance, the presence of tokens in *Coin* implies *Connection* empty.
- **Process** (Fig. 2.4) is a net composed of three subnets coupled by self-loops on the places p_2, p_3 and p_4. The component at the bottom includes a dead transition (t_6); it will never be enabled, although the state equation ensures at least one possibility of firing it. Like with our previous example, reasoning only on the state equation is not enough to capture

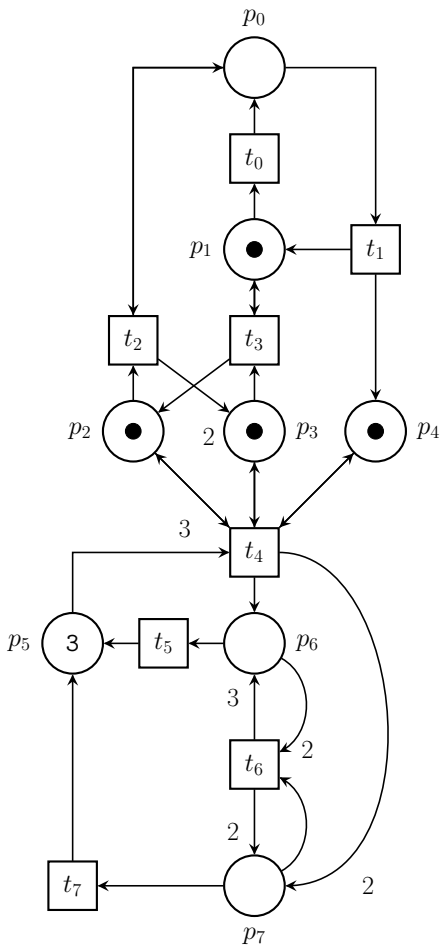


Fig. 2.4 Process net with $\mathbb{P} \triangleq (p_2 + p_3 + p_4 \geqslant 1 \wedge p_7 \leqslant 2)$

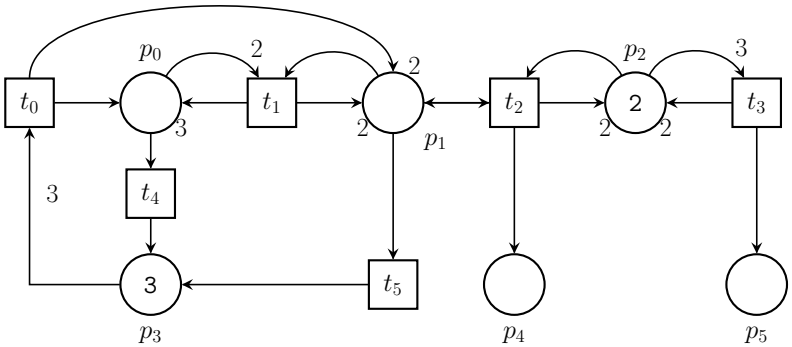


Fig. 2.5 Murphy net [ADH23b] with $\mathbb{P} \triangleq (p_1 \leqslant 2 \wedge p_4 \geqslant p_5)$

the exact behavior of this net. For instance, the state equation allows getting 3 tokens in p_7, which would contradict our invariant.

- **Murphy** [ADH23b] (Fig. 2.5) is a net combining PGCD with the "bottom component" of net Process.

We compared SMPT against ITS-Tools [Thi15], LoLA [Wol18], and TAPAAL [Dav+12]; and give our results in Table 2.1. All results are computed using 4 cores, a limit of 16 GB of RAM, and a timeout of 1 h. A result of TLE stands for "Time Limit Exceeded". For SMPT, we marked with an asterisk (*) the results computed using our saturation-based generalization. Our results show that SMPT is able to answer queries on several classes of examples that are out of reach for some, or all the other tools; often by orders of magnitude.

Instance	SMPT	ITS-Tools	LoLA	TAPAAL
Murphy	**0.75** *	TLE	TLE	TLE
PGCD	**0.11** *	139.08	TLE	TLE
CryptoMiner	0.19 *	5.92	TLE	**0.18**
Parity	0.40 *	3.36	**0.01**	4.16
Process	83.39	TLE	**0.03**	0.18

Table 2.1 Computation time on our synthetic examples (time in seconds).

We also experimented with two other recent tools for reachability: KReach [DL20], which provides a complete implementation of Kosaraju's original decision procedure, and FastForward [BHO21], a tool for efficiently finding counter-examples in unbounded Petri nets (but that may report that an invariant is true in some cases). We do not include these tools in our findings since they were unable to answer any of our problems.

2.4.2 Evaluation on Performance

Since it is not sufficient to use only a small number of hand-picked examples to check the performance of a tool, we also provide results obtained on a set of 30 problems (a net together with an invariant) that are borrowed from test cases used by the tool Sara [WW12; Wim13] (examples test$\{3, 4, 12\}$) and a similar software, called Reach, that is part of the Tina toolbox [BRV04; LAA23] (examples 1, 3u, ..., zz). Most of these problems can be easily answered, but they are interesting to test our reliability on a relatively even-handed benchmark.

Our benchmark also includes 6 examples of bounded nets obtained by limiting the number of times we can fire transitions in the nets PGCD and CryptoMiner. (This is achieved by adding a new place that loses a token when a transition is fired.)

The experiments were performed with the same conditions as previously but with a timeout of only 255 s. We display our results in the chart of Fig. 2.6, which gives the number of feasible

problems, for each tool, when we change the timeout value (we use a logarithmic scale for the time value). We also provide the computation times, for the same dataset, in Table 2.2. We observe that our performances are on par with TAPAAL, which is the fastest among our three reference tools on this benchmark.

Instance	SMPT	ITS-Tools	LoLA	TAPAAL
1	**0.15**	0.78	5.01	0.17
3u	1.84 *	0.80	**0.01**	0.16
5pi	6.86	0.88	**0.01**	0.17
6pi	0.21	0.88	**0.01**	0.16
7pi	**0.15**	0.78	5.00	0.16
Crypto	0.20 *	4.94	TLE	**0.16**
Crypto-10000	0.25 *	4.88	**0.02**	0.16
Crypto-50	0.22 *	1.04	**0.01**	0.18
Crypto-500	0.24 *	4.57	**0.01**	0.17
PGCD-10000	**0.14** *	142.63	TLE	96.59
PGCD-50	0.10 *	0.87	**0.01**	0.17
PGCD-500	0.11 *	1.13	**0.08**	0.30
b	**0.09**	0.79	5.02	0.16
kw2	0.18	0.78	5.01	**0.16**
mtx	0.60	0.86	0.00	**0.16**
nope	**0.12**	0.78	5.01	0.16
nope2	**0.10**	0.76	5.01	0.17
test12	0.10	0.76	5.00	**0.05**
test3	**0.15**	0.91	5.02	0.18
test4	0.15	0.82	**0.01**	0.17
u	**0.09**	0.77	5.00	0.17
w	**0.10**	0.79	5.02	0.16
w1	0.10	0.75	5.01	**0.05**
w2	**0.10**	0.80	5.00	0.17
wb	0.29 *	0.80	5.00	**0.17**
we	**0.16**	0.78	5.01	0.17
x	1.24 *	0.84	**0.01**	0.16
z	**0.10**	1.22	5.00	0.30
ze	0.71	0.88	5.03	**0.17**
zz	**0.12** *	1.64	5.01	0.25

Table 2.2 Computation time on existing benchmarks (time in seconds).

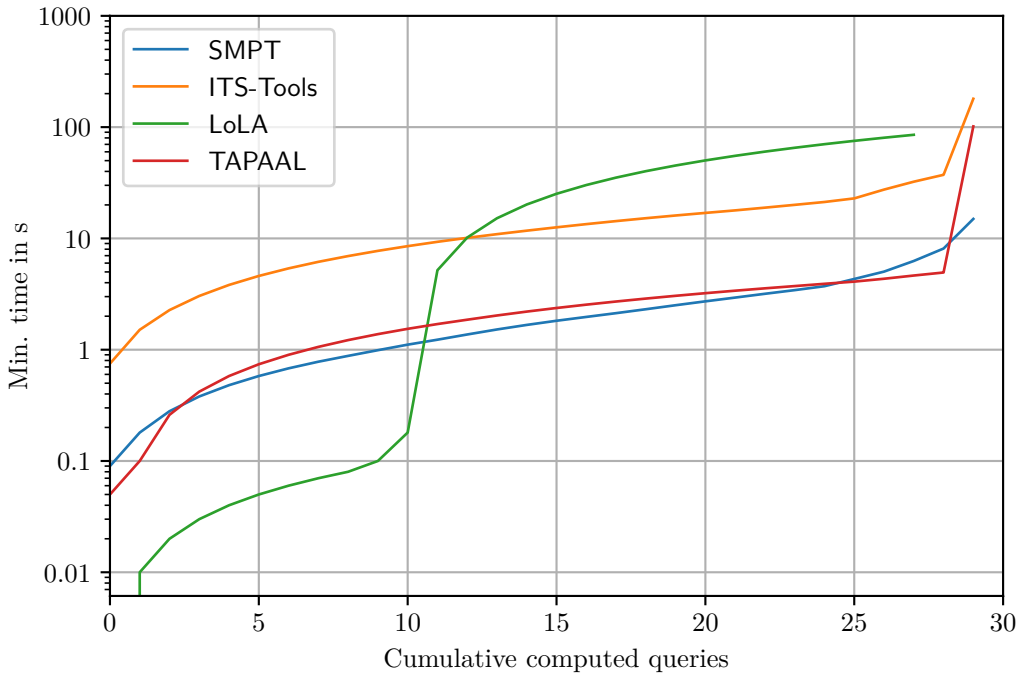


Fig. 2.6 Minimal timeout to compute a given number of queries.

2.4.3 Computation of Invariance Certificates

A distinctive feature of SMPT is the ability to output a linear inductive invariant for reachability problems: when we find that $\mathbb{P}$ is invariant, we are also able to output an inductive formula $\mathbb{C}$, of the form $\mathbb{P} \wedge G$, that can be checked independently with an SMT solver. We can find the same capability in the tool Petrinizer [Esp+14] in the case of coverability properties.

To get a better sense of this feature, we give the actual outputs computed with SMPT on the two nets of Fig. 2.1. The invariant for the net Parity is $\mathbb{P}_1 \triangleq (p_0 \geqslant 1)$, and for PGCD it is $\mathbb{P}_2 \triangleq (p_1 \leqslant p_2)$.

The certificate for property $\mathbb{P}_1$ on Parity (see Fig. 2.7) is $\mathbb{C}_1 \equiv (p_0 \geqslant 1) \wedge \forall k \,.\, ((p_0 < 2\,k + 2) \vee (p_0 \geqslant 2\,k + 3))$, which is equivalent to $(p_0 \geqslant 1) \wedge (\forall k \geqslant 1) \,.\, (p_0 \neq 2.k)$, meaning the marking of p_0 is odd. This invariant would be different if we changed the initial marking to an even number.

```
[PDR] Certificate of invariance
# (not (p0 < 1))
# (forall (k1) ((p0 < (2 + (k1 * 2))) or (p0 + (-2 * (k1 + 1))) >= 1))
```

Fig. 2.7 Certificate of invariance for the Parity net in Fig. 2.1.

The certificate for property $\mathbb{P}_2$ on PGCD (see Fig. 2.8) is $\mathbb{C}_2 \equiv (p_1 \leqslant p_2) \wedge \forall k \,.\, ((p_0 < k+3) \vee (p_2 - p_1 \geqslant k+1))$ and may seem quite inscrutable. It happens actually that the saturation "learned" the invariant $p_0 + p_1 = p_2 + 2$ and was able to use this information to strengthen property $\mathbb{P}_2$ into an inductive invariant.

```
[PDR] Certificate of invariance
# (not (p1 > p2))
# (forall (k1) ((p0 < (3 + (k1 * 1))) or ((p1 + (1 * (k1 + 1))) <= p2)))
```

Fig. 2.8 Certificate of invariance for the PGCD net in Fig. 2.1.

2.5 Discussion

An important result in concurrency theory is the decidability of reachability for Petri nets [May81; Kos82; Lam92]. However, the Kosaraju-Lambert-Mayr-Sacerdote-Tenney approach does not lead to a workable algorithm. It is in fact a feat that this algorithm has been implemented at all, see, e.g., the tool KReach [DL20]. While the (very high) complexity of the problem means that no single algorithm could work efficiently on all inputs, it does not prevent the existence of methods that work well on some classes of problems. For example, several algorithms are tailored for the discovery of counter-examples. We mention the tool FastForward [BHO21] in our experiments, which explicitly targets the case of unbounded nets.

We propose a method that works as well on bounded as on unbounded ones; that behaves well when the invariant is true; and that works with "genuine" reachability properties, and not only with coverability. But there is of course no panacea. Our approach relies on the use of linear predicates, which are incrementally strengthened until we find an invariant based on: the transition relation of the net; the property we want to prove (it is "property-directed"); and constraints on the initial states. As a reminder, this is in line with a property proved by Leroux [Ler09; Ler10], which states that when a final configuration is not reachable then "*there exist checkable certificates of non-reachability in the Presburger arithmetic*".

This is not something new. Many tools rely on the use of integer programming techniques to check reachability properties. We can mention the tool Sara [WW12] that is now integrated inside LoLA [Wol18] and can answer reachability problems on unbounded nets; or tools like FAST [Bar+08], designed for the analysis of systems manipulating unbounded integer variables. An advantage of our method is that we proceed in a lazy way. We never explicitly compute the structural invariants of a net, never switch between a Presburger formula and its representation as a semilinear set (useful when one wants to compute the "Kleene closure" of a linear constraint), ... and instead let an SMT solver work its magic.

We can find other related works, such as [Klo+13; Esp+14; KBJ21]. Nonetheless, they all focus on coverability properties. Coverability is not only a subclass of the general reachability problem, it has a far simpler theoretical complexity (EXPSPACE [Lip76] vs Ackermannian [CO22; Ler22]). It is also not expressive enough for checking the absence of deadlocks or for complex invariants, for instance involving a comparison between the marking of two places, such as $p < q$. The idea we advocate is that approaches based only on the generalization of markings (such as (G1)) are not enough. This is why we believe that abstractions (G2) and (G3) defined in Theorem 2.2 are noteworthy.

We can also compare our approach with tools oriented to the verification of bounded Petri nets; since many of them integrate methods and semi-decision procedures that can work in the unbounded case. We compared ourselves with three tools: ITS-Tools [Thi15], TAPAAL [Dav+12] and LoLA, that have in common to be the top trio in the Model Checking Contest [Amp+19]. Our main contribution in this context is to provide a new benchmark of nets and properties that can be used to evaluate future reachability algorithms "for expressiveness".

The methods closest to ours in these portfolios are Bounded Model Checking and k-induction [SSS00], which are also based on the use of SMT solvers. We can mention the case of ITS-Tools that can build a symbolic over-approximation of the state space, represented as sets of constraints [Thi20]. This approximation is enough when it is included in the invariant that we check, but inconclusive otherwise. A subtle and important difference between PDR and these methods is that PDR needs only $2n$ variables (the $\boldsymbol{p}$ and $\boldsymbol{p'}$), whereas we need n fresh variables at each new iteration of k-induction (so $kn + 1$ variables in total). This contributes to the good performances of PDR since the complexity of the SMT problems is in part relative to the number of variables involved. Another example of over-approximation is the use of the state equation method [Mur77], already described in Sect.1.5.4, that can strengthen the computations of inductive invariants by adding extra constraints, such as place invariants [STC96], traps [EM00; Esp+14], causality constraints, etc. We exploit similar constraints with SMPT in the MCC to better refine our invariants.

To conclude, our experiments confirm that we benefit from using a more diverse set of techniques, and are still in need of new techniques, able to handle new classes of problems. For instance, we can attribute the good results of TAPAAL, in our experiments, to their implementation of a Trace Abstraction Refinement (TAR) techniques, guided by counter-examples [CJL17]. The same can be said with LoLA, which also uses a CEGAR-like method [WW12]. We believe that our approach could be a useful addition to these techniques, which is why we integrated PDR in the workflow of SMPT during the Model Checking Contest.

And last, but not least, our extension of PDR provides a constructive method for computing certificates of invariance, when it terminates.

This work has been published in:

- N. Amat, S. Dal Zilio, and T. Hujsa. "Property Directed Reachability for Generalized Petri Nets". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. vol. 13243. Lecture Notes in Computer Science. Springer, 2022. DOI: 10.1007/978-3-030-99524-9_28

A conference artifact is available on Zenodo:

- N. Amat, S. Dal Zilio, and T. Hujsa. *Artifact for TACAS 2022 Paper: Property Directed Reachability for Generalized Petri Nets*. Zenodo, 2022. DOI: 10.5281/zenodo.5863379

The tool related to this chapter is:

- SMPT https://github.com/nicolasAmat/SMPT

Chapter 3

Polyhedral Reduction

Definition and Straightforward Application

> The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.
>
> Edsger Wybe Djikstra

In this chapter, we define a method for taking advantage of net reduction in combination with an SMT-based model checker. The approach consists in transforming a reachability problem about some Petri net into the verification of an updated reachability property on a reduced version of this net. This method, called polyhedral reduction, relies on a new state space abstraction based on linear constraints.

We prove the correctness of this method using a new notion of equivalence between nets, called *polyhedral abstraction equivalence*. We provide a complete framework to define and check the correctness of equivalence judgments, prove that this relation is a congruence, and give examples of basic equivalence relations that derive from structural reductions.

Our approach has been implemented in the tool SMPT. As a testbed, we propose an adaptation of the Bounded Model Checking (BMC) method. Experimental results show that our approach works well, even when we only have a moderate amount of reductions.

3.1 Introduction

This chapter is the starting point of our work on polyhedral reduction, which is the central topic of the following chapters (Chapters 4 to 7).

Context. Berthomieu recently proposed a new abstraction technique [BLD18; BLD19] based on reductions [BL85; Ber87]. The idea is to compute reductions of the form (N, E, N'), where: N is an initial net (that we want to analyze); N' is a residual net (hopefully much simpler than N); and E is a Presburger predicate. The idea is to preserve enough information in E to rebuild the reachable markings of N, knowing only the ones of N'. In a nutshell, we capture and abstract the effect of reductions using linear constraints between the places of N and N'. This has been previously applied technique in a symbolic model checker part of the Tina toolbox [BRV04], called Tedd, that uses Set Decision Diagrams [Thi+09] in order to generate an abstract representation for the state space of a net N.

Challenge. In this chapter, we want to show how this approach can be combined with SMT-based verification. In particular, we want a theoretical framework and an "elegant" way to integrate reductions into known verification procedures.

Proposal. We provide a complete theoretical framework based on the definition of a new equivalence relation between Petri nets and show how to use it for checking reachability properties. As with PDR, our method does not impose restrictions on the syntax of nets, such as constraints on the weights of arcs or bounds on the marking of places. In practice, we can often reduce a Petri net N with n places (from a high dimensional space) into a residual net N' with far fewer places, say n' (in a lower-dimensional space). Hence, with our approach, we can represent the state space of N as the "inverse image" by the Presburger predicate E of a subset of vectors of dimension n'. This technique can result in a very compact representation of the state space. In this chapter, we show that we can benefit from this "dimensionality reduction" effect when using automatic deduction procedures.

To adapt our approach to the theory of SMT solving, we define an abstraction based on Boolean combinations of linear constraints between integer variables (representing the marking of places). This results in a new relation $N \equiv_E N'$, the counterpart of the tuple (N, E, N') in an SMT setting. We named this relation a *polyhedral abstraction equivalence* (or just *polyhedral equivalence*) in reference to "polyhedral models" used in program optimization and static analysis [Fea96; BJT99]. Indeed, like in these works, we propose an algebraic representation of the relation between a model and its state space based on the sets of solutions to linear constraints. We should also often use the term *E-abstraction equivalence* to emphasize the importance of the linear predicate E. One of our main results is that, given a relation $N \equiv_E N'$, we can derive a formula $\tilde{E}$ such that F is reachable in N if and only if $\tilde{E} \wedge F$ is reachable in the net N'. Since the residual net may be much simpler than the initial one, we expect that checking $\tilde{E} \wedge F$ on N' is more efficient than checking F on N.

Our approach has been implemented in the tool SMPT, and computing experiments show that reductions are effective on the large benchmark of queries used in the Model Checking Contest, and our approach works well, even when we only have a moderate amount of reductions.

Outline and Contributions. The chapter is organized as follows. We start by defining our notion of polyhedral abstraction equivalence in Sect. 3.2 and prove several of its properties in Sect. 3.3. Section 3.4.1 describes some structural reductions used in our approach and shows how they correspond to axioms of our polyhedral equivalence. We also prove that polyhedral equivalences are preserved by composition and transitivity (Sect. 3.4.2), which gives a simple way to check the equivalence between two complex nets. We use these results in Sect. 3.5 to provide a general framework for taking advantage of polyhedral reduction with SMT-based verification methods; and then, we describe an adaptation of the Bounded Model Checking (BMC) method in Sect. 3.6 and prove its correctness. Before concluding, we report on experimental results on the extensive collection of nets and queries used in the Model Checking Contest (Sect. 3.7). Our results are pretty promising. For example, on our benchmark, we observe that we can compute twice as many results using reductions than without.

3.2 Polyhedral Reduction and E-Equivalence

We define a new notion, called *E-abstraction*, that states a correspondence between the set of reachable markings of two Petri nets "modulo" some Presburger predicate E. Basically, we have that (N_2, m_2) is an abstraction of (N_1, m_1) when, for every sequence $m_1 \overset{\sigma_1}{\Longrightarrow} m'_1$ in N_1, there must exist a sequence $m_2 \overset{\sigma_2}{\Longrightarrow} m'_2$ in N_2 such that $E \wedge \underline{m'_1} \wedge \underline{m'_2}$ is satisfiable. We also require that such a sequence exists for every marking m'_2 such that $E \wedge \underline{m'_1} \wedge \underline{m'_2}$ is satisfiable. Therefore, knowing E, we can compute the reachable markings of N_1 from those of N_2.

In this case, we also ask for the observable sequences (refer to Sect. 1.1.4), σ_1 and σ_2, to be equal. With the addition of this constraint, we prove that the reflexive and symmetric closure of an E-abstraction is also a congruence, which we call an E-equivalence and formally define later.

We can illustrate these notions using the two nets M_1, M_2 in Fig. 3.1 and the linear constraint $E_M \triangleq \exists a_1 \,.\, (p_5 = p_4) \wedge (a_1 = p_1 + p_2) \wedge (a_2 = p_3 + p_4) \wedge (a_1 = a_2)$. Net M_1 (left) is taken from Sect. 1.1.5 (Fig. 1.1) where we showed that marking $m'_1 \triangleq p_0{*}3\ p_2{*}1\ p_3{*}1\ p_6{*}3$ is reachable from the initial marking m_1 by firing the sequence $t_0t_0t_1t_1t_2t_3t_4$. Then we have $E_M \wedge \underline{m'_1}$ entails $(p_0 = 3) \wedge (p_6 = 3) \wedge (a_2 = 1)$. Hence, if we prove that (M_1, m_1) is E_M-equivalent to (M_2, m_2), we can conclude that the marking $m'_2 \triangleq a_2{*}1\ p_0{*}3\ p_6{*}3$ is reachable in M_2.

Conversely, we have several markings (exactly 4) in M_1 that correspond to the constraint $E_M \wedge \underline{m'_2}$, that is, $(p_5 = p_4) \wedge (p_1 + p_2 = 1) \wedge (p_3 + p_4 = 1)$. All these markings are reachable in M_1 using the same observable sequence **a a b c**.

In practice—but not directly entailed by our equivalence definition since the relation is symmetric—, each marking m'_2 reachable in N_2 can be associated with a unique subset of markings reachable in N_1, defined from the solutions to $E \wedge \underline{m'_2}$. We can show (Theorem 4.8 in Chapter 4) that this gives a partition of the reachable markings of (N_1, m_1) into "convex sets"—hence the name polyhedral abstraction—each associated to a reachable marking in

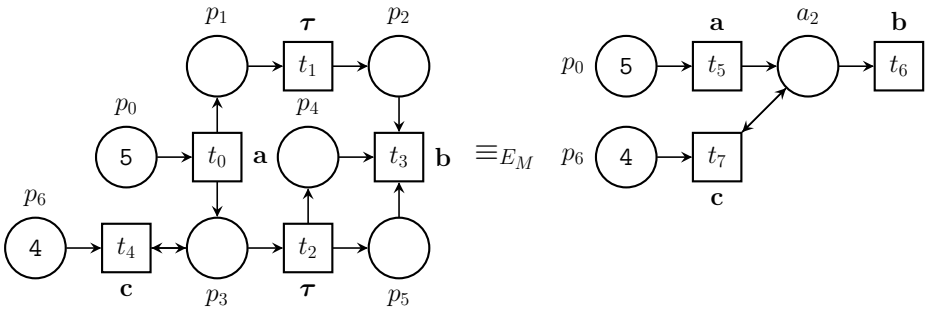


Fig. 3.1 An example of Petri net (Fig. 1.1), (M_1, m_1) (left), and one of its polyhedral reductions, (M_2, m_2) (right), with $E_M \triangleq \exists a_1 \,.\, (p_5 = p_4) \wedge (a_1 = p_1 + p_2) \wedge (a_2 = p_3 + p_4) \wedge (a_1 = a_2)$.

N_2. This motivates our choice of calling this relation a *polyhedral abstraction* (and *polyhedral abstraction equivalence*, or just *polyhedral equivalence*, when the relation holds in both ways).

Our approach is particularly useful when the state space of N_2 is very small compared to that of N_1. In the extreme case, we can even find examples where N_2 is the "empty" net (a net with zero places and, therefore, a unique marking), but this condition is not a requisite in our approach.

While our approach does not dictate a particular method for finding pairs of equivalent nets, we rely on an automatic approach based on the use of *structural net reductions*. When the net N_1 can be reduced, we will obtain a resulting net (N_2) and a condition (E) such that N_2 is a polyhedral reduction of N_1. In this case, E will always be expressed as a conjunction of equality and inequality constraints between linear combinations of integer variables (marking places and some existential variables). This is why we should often use the term *reduction constraints* when referring to E. We aim to transform any reachability problem on the net N_1 into a reachability problem on the (reduced) net N_2, which is typically much easier to check.

3.2.1 Solvable Predicates

Before formally defining our equivalence, we must introduce some constraints on the condition, E, used to correlate the markings of two different nets, N_1 and N_2. We say that a pair of markings (m_1, m_2) over $\mathbb{N}^{P_1} \times \mathbb{N}^{P_2}$ are *compatible* when they have the same number of tokens on their shared places, meaning $m_1(p) = m_2(p)$ for all p in $P_1 \cap P_2$. This is a necessary and sufficient condition for formula $\underline{m_1} \wedge \underline{m_2}$ to be satisfiable.

Moreover, if V is the set of free variables of $\underline{m_1}$, $\underline{m_2}$, and the free variables of E are included in V, we say that m_1 and m_2 are *related up-to* E, denoted $m_1 \equiv_E m_2$, when $E \wedge \underline{m_1} \wedge \underline{m_2}$ is satisfiable.

$$m_1 \equiv_E m_2 \quad \Leftrightarrow \quad \exists m \in \mathbb{N}^V \,.\, m \models E \wedge \underline{m_1} \wedge \underline{m_2} \tag{3.1}$$

This leads to the notion of *solvable predicate*, such that every reachable marking of N_1 can be paired with at least one reachable marking of N_2 to form a solution of E; and reciprocally.

Definition 3.1 (Solvable Predicate of Reduction Constraints)**.** *A Presburger predicate, E, is solvable for N_1, N_2 if and only if for all reachable markings m_1 in N_1 there exists at least one marking m_2 of N_2, related to m_1 up-to E (i.e., $m_1 \equiv_E m_2$), and vice versa for every reachable marking m_2 in N_2.*

This relation defines an equivalence between markings of two different nets ($\equiv_E \subseteq \mathbb{N}^{P_1} \times \mathbb{N}^{P_2}$) and, by extension, can be used to define an equivalence between nets themselves, that is called *polyhedral equivalence*, where all reachable markings of N_1 are related to reachable markings of N_2 (and conversely), as explained next.

3.2.2 E-Equivalence

In the following, when we use an E-abstraction equivalence between two marked nets (N_1, m_1) and (N_2, m_2), we ask that condition E be *solvable for* N_1, N_2 (see condition (A2)). This property will always be true for the reduction constraints generated with our method.

We define our notion of E-abstraction as an equivalence relation between the markings reached using equal "observable sequences". An E-abstraction equivalence (shortened as E-equivalence) is an abstraction in both directions.

Definition 3.2 (E-Abstraction and E-Abstraction Equivalence)**.** *Assume that $N_1 \triangleq (P_1, T_1, \mathrm{Pre}_1, \mathrm{Post}_1)$ and $N_2 \triangleq (P_2, T_2, \mathrm{Pre}_2, \mathrm{Post}_2)$ are two Petri nets with respective labeling functions l_1, l_2, over the same alphabet Σ, and that E is a Presburger formula whose free variables are included in $P_1 \cup P_2$. We say that the marked net (N_2, m_2) is an E-abstraction of (N_1, m_1), denoted $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$, if and only if:*

- **(A1)** *the initial markings are related up-to E, meaning $m_1 \equiv_E m_2$;*
- **(A2)** *for all firing sequences $(N_1, m_1) \xrightarrow{\varrho_1} (N_1, m_1')$ in N_1, there is at least one marking m_2' over P_2 such that $m_1' \equiv_E m_2'$ (i.e., solvable), and for all markings m_2' over P_2 such that $m_1' \equiv_E m_2'$ there exists a firing sequence $\varrho_2 \in T_2^\star$ such that $(N_2, m_2) \xrightarrow{\varrho_2} (N_2, m_2')$ and $l_1(\varrho_1) = l_2(\varrho_2)$.*

We say that (N_1, m_1) is E-equivalent to (N_2, m_2), denoted $(N_1, m_1) \equiv_E (N_2, m_2)$, when we have both $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$ and $(N_2, m_2) \sqsubseteq_E (N_1, m_1)$.

Although E-abstraction looks like a simulation, it is not, since the pair of reachable markings m_1', m_2' from the definition does not satisfy $(N_1, m_1') \sqsubseteq_E (N_2, m_2')$ in general. Therefore, this relation $\sqsubseteq_E$ is broader than a simulation, but suffices for our primary goal, Petri net reduction. Of course, $\equiv_E$ is not a bisimulation either. It is also quite simple to show that checking E-abstraction equivalence is undecidable in general.

Theorem 3.1 (Undecidability of E-Equivalence)**.** *The problem of checking whether a statement* $(N_1, m_1) \equiv_E (N_2, m_2)$ *is valid is undecidable.*

Proof. By contradiction, we suppose that there exists some algorithm, say $\mathcal{A}$, that checks the E-abstraction equivalence problem. More precisely, the input of $\mathcal{A}$ consists of two marked nets (N_1, m_1) and (N_2, m_2), as well as a Presburger formula E with free variables in the places of N_1 and N_2. The output of $\mathcal{A}$ is a Boolean, indicating whether $(N_1, m_1) \equiv_E (N_2, m_2)$ holds or not.

Let us consider another problem: given any pair of marked nets (N_1, m_1) and (N_2, m_2) with the same set of places, and equal initial markings (i.e., $m_1 = m_2$), check the marking equivalence of both nets, that is check if $R(N_1, m_1) = R(N_2, m_2)$ holds. This problem is known to be undecidable*. Yet, we will show that algorithm $\mathcal{A}$ is always able to answer to this problem, hence the contradiction.

Take any pair of marked nets (N_1, m_1) and (N_2, m_2) with the same set of places and $m_1 = m_2$. We equip each net with a labeling function l_1 (resp. l_2) such that $l_1(t) = \tau$ (resp. $l_2(t) = \tau$) for all transition t of N_1 (resp. N_2). Let us show first that: $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$ with the trivial constraint $E \triangleq$ True is equivalent to $R(N_1, m_1) \subseteq R(N_2, m_2)$.

Condition (A1) trivially holds since $m_1 = m_2$. We now show that condition (A2) is necessary and sufficient for $R(N_1, m_1) \subseteq R(N_2, m_2)$:

- Assume that condition (A2) holds and take a marking m'_1 in $R(N_1, m_1)$. We have $m'_1 \equiv_E m'_1$. Then, by condition (A2) we get $m'_1 \in R(N_2, m_2)$, and so $R(N_1, m_1) \subseteq R(N_2, m_2)$.
- Assume that $R(N_1, m_1) \subseteq R(N_2, m_2)$ and take a firing sequence $(N_1, m_1) \xrightarrow{\varrho_1} (N_1, m'_1)$. Since all transitions are silent we have $l_1(\varrho_1) = \epsilon$. Both nets share the same sets of places, thus m'_1 satisfies $m'_1 \equiv_E m'_1$ (and no other marking $m'_2 \neq m'_1$ satisfies the condition $m'_1 \equiv_E m'_2$). By assumption, $m'_1 \in R(N_2, m_2)$, meaning $(N_2, m_2) \xrightarrow{\varrho_2} (N_2, m'_1)$ for some firing sequence ϱ_2 such that $l_2(\varrho_2) = \epsilon$, and so, condition (A2) holds.

The statement above is proved. By immediate symmetry, we get that $R(N_1, m_1) = R(N_2, m_2)$ is equivalent to $(N_1, m_1) \equiv_E (N_2, m_2)$. As a consequence, checking the marking equivalence problem is equivalent to checking the E-equivalence problem on (N_1, m_1) and (N_2, m_2), with E the trivial constraint. Since algorithm $\mathcal{A}$ is supposed to answer to the latter, it equivalently answers to the former, which is a contradiction.

* Hack proved the undecidability of the marking equivalence between two subparts of nets N_1, N_2 given a pair of initial markings not necessary equal [Hac76]. However, his proof's construction leads to the same results when initial markings are equal. □

Note that condition (A2) in Definition 3.2 can be defined in an alternative way using directly observable sequences:

(A2') for all observable sequences σ such that $(N_1, m_1) \stackrel{\sigma}{\Rightarrow} (N_1, m_1')$, there is at least one marking m_2' over P_2 such that $m_1' \equiv_E m_2'$, and for all markings m_2' over P_2 such that $m_1' \equiv_E m_2'$ we must have $(N_2, m_2) \stackrel{\sigma}{\Rightarrow} (N_2, m_2')$.

Condition (A2) also entails two necessary (but not sufficient) conditions on the sets of reachable markings: (1) $E \wedge \underline{m}$ is satisfiable for all markings m in $R(N_1, m_1)$ or $R(N_2, m_2)$; and (2) assume that m_1', m_2' are markings of N_1, N_2, respectively, such that $E \wedge \underline{m_1'} \wedge \underline{m_2'}$ is satisfiable, then m_1' is reachable if and only if m_2' is reachable, i.e., $m_1' \in R(N_1, m_1) \Leftrightarrow m_2' \in R(N_2, m_2)$.

By definition, relation $\equiv_E$ is symmetric, but we expect that N_2 is a reduced version of N_1. In particular, we expect that $|P_2| \leqslant |P_1|$. That is why by convention, we will consider that the net on the right of an equivalent statement corresponds to the reduced net.

3.3 Basic Properties of Polyhedral Reduction

We prove that we can use E-equivalence to check the reachable markings of N_1 simply by looking at the reachable markings of N_2. We give a first property that is useful in the context of Bounded Model Checking (BMC) when we try to find a counter-example to a property by looking at firing sequences with increasing length. Our second property is useful for checking invariants and is at the basis of our implementation in SMPT of the PDR method for Petri nets.

Lemma 3.2 (Reachability Checking)**.** *Assume that* $(N_1, m_1) \equiv_E (N_2, m_2)$*. Then for all pairs of markings* m_1', m_2' *of* N_1, N_2 *such that* $m_1' \equiv_E m_2'$ *and* $m_2' \in R(N_2, m_2)$ *it is the case that* $m_1' \in R(N_1, m_1)$*.*

Proof. Take m_1', m_2' a pair of markings in N_1, N_2 such that $m_1' \equiv_E m_2'$ and $m_2' \in R(N_2, m_2)$. Hence, there is a firing sequence ϱ_2 such that $(N_2, m_2) \xrightarrow{\varrho_2} (N_2, m_2')$. By condition (A2), since $m_1' \equiv_E m_2'$, there must be a firing sequence in N_1, say ϱ_1, such that $(N_1, m_1) \xrightarrow{\varrho_1} (N_1, m_1')$. Hence, $m_1' \in R(N_1, m_1)$. □

Lemma 3.2 (see Fig. 3.2) can be used to find a counter-example m_1' to some property F in N_1 (where F is a formula whose variables are in P_1), just by looking at the reachable markings of N_2. Indeed, it is enough to find a marking m_2' reachable in N_2 such that $m_2' \models E \wedge \neg F$. This is the result we use in our implementation of the BMC method.

Our second property can be used to prove that every reachable marking of N_1 can be traced back to at least one marking of N_2 using the reduction constraints. (While this mapping is surjective, it is not a function since a state in N_2 could be associated with multiple states in N_1.)

Lemma 3.3 (Invariance Checking)**.** *Assume that* $(N_1, m_1) \equiv_E (N_2, m_2)$*. Then for all* m_1' *in* $R(N_1, m_1)$ *there is* m_2' *in* $R(N_2, m_2)$ *such that* $m_1' \equiv_E m_2'$*.*

Proof. Since m_1' is reachable, there must be a firing sequence ϱ_1 in N_1 such that $(N_1, m_1) \xrightarrow{\varrho_1} (N_1, m_1')$. By condition (A2), there must be some marking m_2' over P_2, related to m_1' up-to E, such that $(N_2, m_2) \xrightarrow{\varrho_2} (N_2, m_2')$ (for some firing sequence ϱ_2). Therefore, we have m_2' reachable in N_2 such that $m_1' \equiv_E m_2'$. □

Using Lemma 3.3 (see Fig. 3.3), we can easily extract an invariant on N_1 from an invariant on N_2. If property $E \wedge \neg F$ is not reachable on N_2, then we can prove that $\neg F$ is not reachable on N_1, meaning F is an invariant. This property (the *invariant conservation* theorem of Sect. 3.5) ensures the soundness of the model checking technique implemented in our tool.

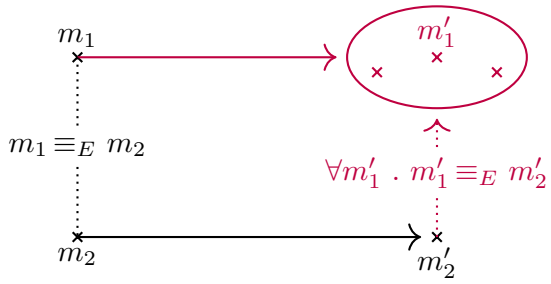


Fig. 3.2 Illustration of Lemma 3.2.

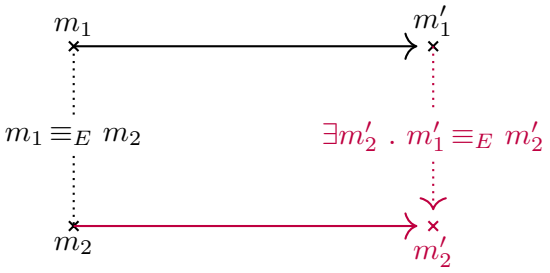


Fig. 3.3 Illustration of Lemma 3.3.

3.4 Deriving E-Equivalences Using Reductions

Before looking at the straightforward combination of the polyhedral equivalence with SMT-based methods, we present some reduction rules (Sect. 3.4.1) and show that we can infer an infinite number of equivalences from a single reduction rule using compositionality, transitivity, and structural modifications involving labels (Sect. 3.4.2). Therefore, each reduction rule can be interpreted as a schema for several polyhedral equivalences.

3.4.1 Reduction Rules

We define a simplified set of relations that can act as "axioms" in a system for deriving E-abstraction equivalences. Each of these axioms derives from a standard *structural reduction rule* (see, e.g., [Ber87; BLD19]), where labeled transitions play the role of interfaces with a possible outside "context".

Each rule is defined by a triplet $((N_1, m_1), E, (N_2, m_2))$ such that $(N_1, m_1) \equiv_E (N_2, m_2)$. A rule also defines possible values for the initial markings, which can be expressed using integer parameters, and may also include a condition that should be true initially.

Each of our rules corresponds to instances of the reduction system defined in a previous work on "counting reachable markings" [BLD19]. Hence, they also correspond to instances of reduction rules implemented in a tool called Reduce, which can automatically find occurrences of reductions in Petri nets and apply them recursively. We give more information about this tool and its relation to our approach in Sect. 3.4.3. This section also contains an example showing how to apply our reduction rules to derive the equivalence stated in Fig. 3.1.

We consider four general families of reductions: first rules for agglomerating places (like [CONCAT] and [AGG]); then rules based on a "place invariant" over the initial net (what we call a *redundancy rule* like [RED] and [SHORTCUT]); rules for garbage collecting dead places or transitions (like [DEADT] and [REDT]); and finally rules that can be used to abstract constant or "closed" places (like [CONSTANT] and [SOURCE]).

We give a detailed proof of correctness for our first "reduction axiom", rule [CONCAT] since it is representative of the complexity of checking simple instances of E-abstraction equivalence. We do not prove similar results for all the rules defined in this section but will only give one other example for the redundancy rule [RED]. All the correctness proofs for the reduction rules in this section are similar to one of these examples.

Rule [CONCAT]

Our first example is the prototypical example of net reduction, as defined in [Ber87]. It also corresponds to the simplest example of the agglomeration rule.

Rule [CONCAT], Fig. 3.4, can be used to fuse together two places "connected only through a deterministic transition" (modeled as a silent transition in our approach). The constraint for applying this rule is that place y_2, in the initial net, must be empty. We also have the condition that no transition except the one we agglomerate (hence no transition that can potentially be merged with an outside context in a synchronous composition, see later in Sect. 3.4.2) can add a token directly to place y_2. This condition is necessary to ensure the correctness of this rule; see Proposition 3.4.

Note that nets N_1 and N_2 are not bounded since transition a can always be fired to increase the marking of places y_1 and x. This means that we need to consider an unbounded number of firing sequences.

Proposition 3.4 (Correctness of Rule [CONCAT])**.** *We have* $(N_1, m_1) \equiv_E (N_2, m_2)$*, with* E *the system containing the single equation* $x = y_1 + y_2$*, and* N_1, N_2 *the nets depicted in Fig. 3.4.*

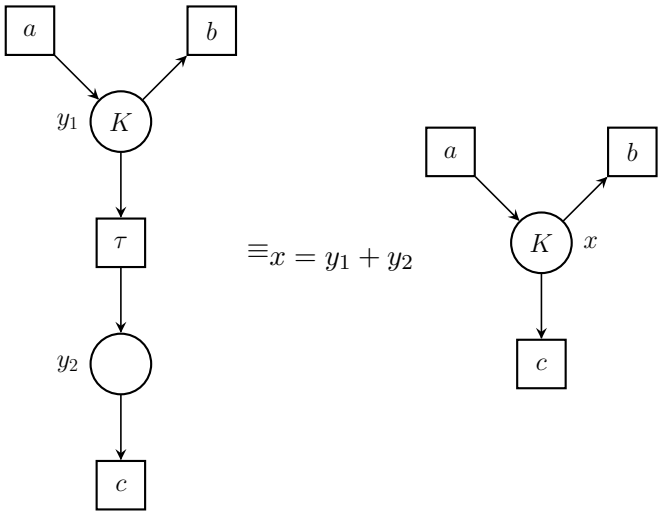


Fig. 3.4 Rule [CONCAT].

Proof. The constraints on the initial marking of the nets are such that $m_1(y_1) = m_2(x) = K \geqslant 0$ and $m_1(y_2) = 0$. To ease the presentation, we should use τ, a, b, c as the name of the transitions and not only as labels. Following Definition 3.2, $(N_1, m_1) \equiv_E (N_2, m_2)$ is shown by proving both $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$ and $(N_2, m_2) \sqsubseteq_E (N_1, m_1)$.

We start by proving condition (A1) and the first constraint of condition (A2) for $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$: by construction, we have $m_1 \equiv_E m_2$ and, for any marking m_1', $m_1' \equiv_E m_2'$ holds by taking $m_2'(x) = y_1 + y_2$.

We now prove the second constraint of condition (A2) for the relation $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$. Assume that $(N_1, m_1) \xrightarrow{\varrho_1} (N_1, m_1')$ and that $m_1' \equiv_E m_2'$. By definition of E, we must have $m_2'(x) = m_1'(y_1) + m_1'(y_2)$. Take ϱ_2 the unique firing sequence of N_2 such that $l_1(\varrho_1) = l_2(\varrho_2)$ (ϱ_2 is obtained from ϱ_1 by erasing all occurrences of the silent transition). We prove that it is the case that $(N_2, m_2) \xrightarrow{\varrho_2} (N_2, m_2')$, by induction on the length of ϱ_1:

- (Base Case) If $\varrho_1 = \epsilon$ then we choose $\varrho_2 = \epsilon$ and $m_2' = m_2$.
- (Induction Case) We have $\varrho_1 = \varsigma_1\, t$ where t is one of the transitions τ, a, b or c. Therefore, there is a marking m_1'' over N_1 such that $(N_1, m_1) \xrightarrow{\varsigma_1} (N_1, m_1'') \xrightarrow{t} (N_1, m_1')$. By induction hypothesis, there is a firing sequence ς_2 and a marking m_2'' over N_2 such that $m_2''(x) = m_1''(y_1) + m_1''(y_2)$ and $(N_2, m_2) \xrightarrow{\varsigma_2} (N_2, m_2'')$. The property follows a case analysis on the possible choice of t.

  Case $t = \tau$: in this case, the overall number of tokens is left unchanged, and we choose $\varrho_2 = \varsigma_2$ and $m_2' = m_2''$.

  Case $t = a$: transition a can always be fired, we choose $\varrho_2 = \varsigma_2\, a$ and m_2' the unique marking such that $m_2'' \xrightarrow{a} m_2'$.

Case $t = b$: since b can be fired from m_1'' it must be the case that $m_1''(y_1) \geqslant 1$. Hence, $m_2''(x) \geqslant 1$ and b can also fire from m_2'' in N_2. We choose $\sigma_2 = \varsigma_2\, b$ and m_2' the unique marking such that $m_2'' \xrightarrow{b} m_2'$. The proof is similar in the case where $t = c$.

Conversely, we show $(N_2, m_2) \sqsubseteq_E (N_1, m_1)$. Condition (A1) still holds by construction. The first constraint of condition (A2) holds by noticing that, if m_2' is given, it suffices to define $m_1'(y_1) = m_2'(x)$ and $m_1'(y_2) = 0$. Then, $m_1' \equiv_E m_2'$ holds as expected.

We are left to prove the second constraint of condition (A2). Assume that we have $(N_2, m_2) \xrightarrow{\varrho_2} (N_2, m_2')$. To begin with, we prove that there is a firing sequence ϱ_1 such that $(N_1, m_1) \xrightarrow{\varrho_1} (N_1, m_1')$ and $l_1(\varrho_1) = l_2(\varrho_2)$, where m_1' is the marking defined by $m_1'(y_1) = m_2'(x)$ and $m_1'(y_2) = 0$ (all the tokens are in y_1). We define ϱ_1 as the (unique) sequence obtained from ϱ_2 by adding one occurrence of the τ-transition before each occurrence of c in ϱ_2. Intuitively, we always keep all the tokens in place y_1 of N_1, except before firing a c; in this case, we add a token to place y_2. We prove, using induction on the size of ϱ_2, that ϱ_1 is a legitimate firing sequence of (N_1, m_1) and that $(N_1, m_1) \xrightarrow{\varrho_1} (N_1, m_1')$. As in the previous case, we proceed by induction on the length of the firing sequence and by case analysis on the last transition fired in N_2.

- (Base Case) If $\varrho_2 = \epsilon$ then we choose $\sigma_1 = \epsilon$ and $m_2' = m_2$.
- (Induction Case) We have $\varrho_2 = \varsigma_2\, t$ where t is one of the transitions a, b or c. Therefore, there is a marking m_2'' over N_2 such that $(N_2, m_2) \xrightarrow{\varsigma_2} (N_2, m_2'') \xrightarrow{t} (N_2, m_2')$. By induction hypothesis, there is a firing sequence ς_1 and a marking m_1'' such that $m_1''(y_1) = m_2''(x)$ and $(N_1, m_1) \xrightarrow{\varsigma_1} (N_1, m_1'')$.

  Case $t = a$: transition a can always be fired, we choose $\varrho_1 = \varsigma_1\, a$ and m_1' the unique marking such that $m_1'' \xrightarrow{a} m_1'$.

  Case $t = b$: since b can be fired from m_2'' it must be the case that $m_2''(x) \geqslant 1$. Hence, $m_1''(y_1) \geqslant 1$ and b can also fire from m_1'' in N_2. We choose $\varrho_1 = \varsigma_1\, b$ and m_2' the unique marking such that $m_1'' \xrightarrow{b} m_1'$.

  Case $t = c$: since c can be fired from m_2'' it must be the case that $m_2''(x) \geqslant 1$. Hence, $m_1''(y_1) \geqslant 1$ and it is possible to fire the sequence $\tau\, c$ from m_1''. So we have $\varrho_1 = \varsigma_1\, \tau\, c$.

Condition (A2) follows from the fact that, when marking m_2' is fixed, then all solutions m_1' to the constraint $m_1' \equiv_E m_2'$ can be reached by firing a sequence of τ transitions from $m_1^{0'}$ such that $m_1^{0'}(y_1) = m_2'(x)$ and $m_1^{0'}(y_2) = 0$. □

Rule [AGG]

Our second example of a rule is for the *agglomeration of places*, see Fig. 3.5, that can be used to simplify a "cluster of places", where tokens can move freely between y_1 and y_2. This is an instance of the general "loop agglomeration" rule given in Fig. 7 of [BLD19].

We could easily define a family of reduction rules similar to [AGG] and [CONCAT] but for longer "loops" or "chains" of places or with the addition of weights on the arcs. For the sake of brevity, we only list one archetypal instance of each rule in this section.

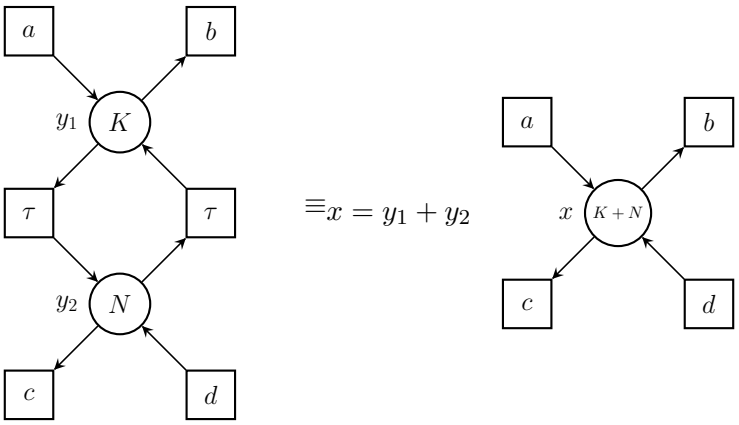


Fig. 3.5 Rule [AGG].

Rules [RED] and [SHORTCUT]

Our following two rules, Fig. 3.6, are reductions that can be used to eliminate *redundant places*, meaning places whose marking derives from a place invariant (and the knowledge of the marking of other places).

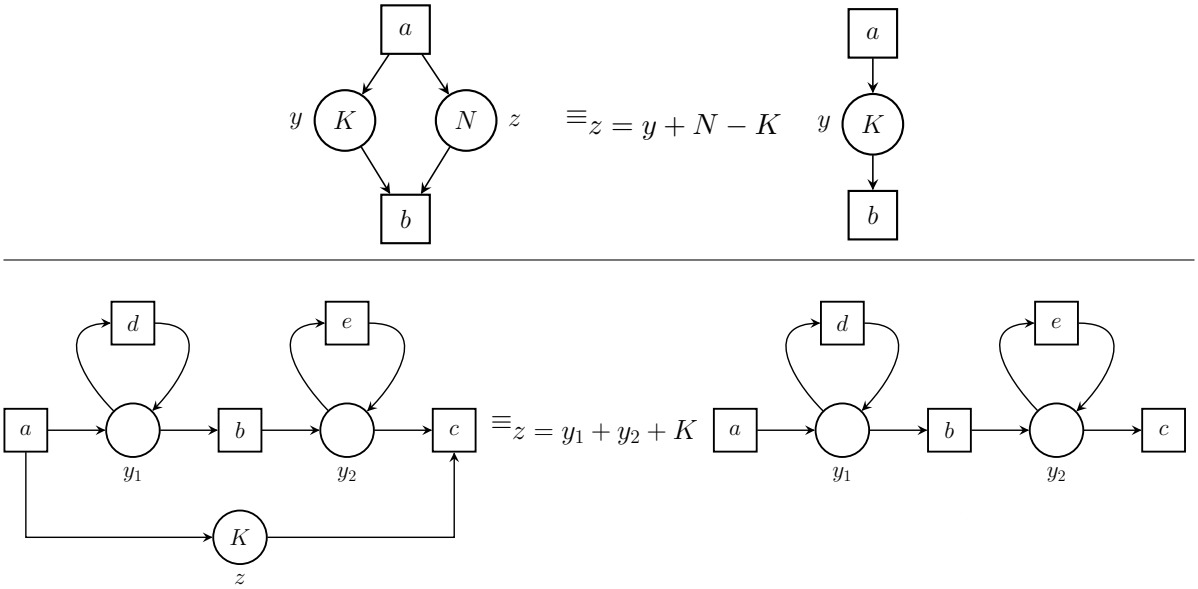


Fig. 3.6 Rule [RED] (above), assuming $K \leqslant N$, and rule [SHORTCUT] (below).

In rule [RED], for instance, with the assumption that we have more tokens in place z than in y initially, it is always the case that $m(z) - m(y)$ is a constant for all the reachable states m. Hence, we can safely eliminate z and keep the relevant information in our linear system E.

Rule [SHORTCUT] gives a more involved example that relies on a condition involving more than two places, an invariant of the form $z = y_1 + y_2 + K$.

We give the proof of correctness for the equivalence corresponding to rule [RED]. The proofs for other redundant place elimination rules are all similar.

Proposition 3.5 (Correctness of Rule [RED])**.** *Assuming $K \leqslant N$, we have $(N_1, m_1) \equiv_E (N_2, m_2)$, with E the system containing the single equation $x = y + N - K$, and N_1, N_2 the nets depicted in Fig. 3.6.*

Proof. Condition $K \leqslant N$ is necessary to have that $N - K \geqslant 0$, and therefore that the marking of y in N_2 is indeed non-negative.

By construction, we have $m_1 \equiv_E m_2$, satisfying condition (A1). The first constraint of condition (A2) follows from the fact that $z = y + N - K$ is an invariant on (N_1, m_1), meaning that for all firing sequences ϱ such that $(N_1, m_1) \xrightarrow{\varrho} (N_1, m_1')$ we have $m_1'(z) = m_1'(y) + N - K$. This can be proved by a simple induction on the length of ϱ. Hence, E is satisfied for every reachable marking in (N_1, m_1), and so E is solvable.

We now prove the second constraint of condition (A2) for the relation $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$. Assume that $(N_1, m_1) \xrightarrow{\varrho} (N_1, m_1')$. We have that ϱ is also a firing sequence of (N_2, m_2) and, moreover, $(N_2, m_2) \xrightarrow{\varrho} (N_2, m_2')$ such that $m_2'(y) = m_1'(y)$. The proof is similar in the other direction. □

Rules [REDT] and [DEADT]

We can use the same approach to simplify transitions in a net rather than places. One such example is rule [REDT], to remove redundant transitions. Such rules are interesting because, when applied in collaboration with others, they can create new opportunities to apply reductions. We give an example of such a mechanism in the example of Sect. 3.4.3.

Another example is the elimination of dead transitions, rule [DEADT] (Figure 3.7), that can eliminate transitions that are "structurally dead". In this example, we know that place x will always stay empty since no transition can increase its marking. Hence, the τ transition is dead, and we can remove it without modifying the set of reachable markings or the observable sequences.

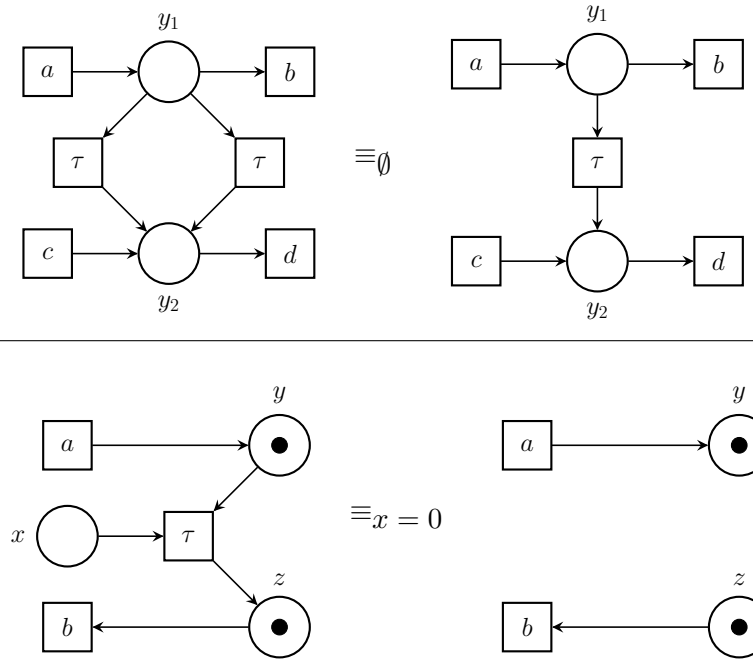


Fig. 3.7 Rules [REDT] (above), and [DEADT] (below).

Rules [CONSTANT] and [SOURCE]

Our last examples of rules illustrate the case of equivalences $(N_1, m_1) \equiv_E (N_2, m_2)$ where the final Petri net is "empty" (denoted $\emptyset$). A Petri net with an empty set of places has only one marking; the empty mapping (the only function in $\emptyset \to \mathbb{N}$).

In this case, the reachable markings of (N_1, m_1) are exactly defined by the non-negative solutions of the predicate E.

Such cases may occur in practice when we can apply several reductions in a row. We say that the initial net is "fully reducible". In example [SOURCE], we can abstract the state space of the initial net with the single constraint $x \leqslant K$.

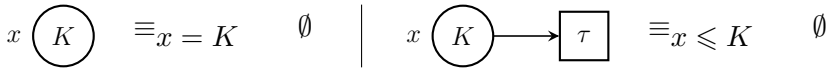


Fig. 3.8 Rules [CONSTANT] (left) and [SOURCE] (right).

We have other rules that allow us to fully reduce a net. For instance, specific structural or behavioral restrictions, such as nets that are marked graphs or other cases where the set of reachable markings is exactly defined by the solutions of the state equation [Huj+20a; Huj+20b].

3.4.2 Composition Laws

We prove that polyhedral equivalence is a transitive relation (Theorem 3.6) that is also closed by synchronous composition (Theorem 3.8) and relabeling (Theorem 3.9). These results can be

used as a set of "algebraic laws" allowing us to derive complex equivalence assertions from much simpler instances, or *axioms*, inside arbitrary contexts. We give an example of such reasoning in Sect. 3.4.3.

Before defining our composition laws, we describe sufficient conditions to compose equivalence relations safely. The goal is to avoid inconsistencies that could emerge if we inadvertently reuse the same variable in different reduction constraints.

Let EQ be the statement $(N_1, m_1) \equiv_E (N_2, m_2)$. We say that the equivalence statement $\mathrm{EQ}' : (N_2, m_2) \equiv_{E'} (N_3, m_3)$ is *compatible* with EQ when $P_1 \cap P_3 \subseteq P_2$. We also say that a net N_3 is *disjoint* from EQ when $(P_1 \cup P_2) \cap P_3 = \emptyset$.

The composition laws stated in the following theorems help build larger equivalences from simpler axioms (reduction rules). In the next section, we show some examples of reductions and how they occur in the example of Fig. 3.1.

Preservation by Chaining

We prove we can chain equivalences together to derive more general reduction rules. When doing so, we need to combine constraints together. Since the equivalence is symmetric, it is enough to prove the results on E-abstraction.

Theorem 3.6. *Assume that* $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$ *and* $(N_2, m_2) \sqsubseteq_{E'} (N_3, m_3)$ *are two compatible statements, then* $(N_1, m_1) \sqsubseteq_{\exists P_2 \setminus (P_1 \cup P_3).E \wedge E'} (N_3, m_3)$.

Proof. We show both conditions (A1) and (A2) of Definition 3.2 applied to the relation $(N_1, m_1) \sqsubseteq_{\exists P_2 \setminus (P_1 \cup P_3).E \wedge E'} (N_3, m_3)$. We start by proving (A1) and the first part of (A2).

Assume that S is the set of places "freshly" introduced in P_2 but not in P_3, i.e., $S \triangleq P_2 \setminus (P_1 \cup P_3)$. First, we prove that predicate $\exists S \,.\, E \wedge E'$ is solvable for N_1, N_3. Take a marking m'_1 in $R(N_1, m_1)$. From condition (A2) of the statement $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$ there is a marking $m'_2 \in R(N_2, m_2)$ such that $m'_1 \equiv_E m'_2$. Again, from the statement $(N_2, m_2) \sqsubseteq_E (N_3, m_3)$, there is also a marking $m'_3 \in R(N_3, m_3)$ such that $m'_2 \equiv_{E'} m'_3$. By assumption, both equivalence statements are compatible, i.e., $P_1 \cap P_3 \subseteq P_2$, then $\underline{m'_1} \wedge \underline{m'_2} \wedge \underline{m'_3}$ is satisfiable. Since $\mathrm{FV}(E) \subseteq P_1 \cup P_2$ and $\mathrm{FV}(E') \subseteq P_2 \cup P_3$, we have $\mathrm{FV}(E) \cap \mathrm{FV}(E') \subseteq P_2$, then $\underline{m'_1} \wedge \underline{m'_2} \wedge \underline{m'_3} \wedge E \wedge E'$ is satisfiable. Hence, $\models \exists S \,.\, E \wedge E' \wedge \underline{m'_1} \wedge \underline{m'_3}$, equivalent to $m'_1 \equiv_{\exists S.E \wedge E'} m'_3$.

For similar reasons, we have $m_1 \equiv_E m_2$ and $m_2 \equiv_{E'} m_3$ entails $m_1 \equiv_{\exists S.E \wedge E'} m_3$. Indeed, we still have the stronger property that $\exists S \,.\, \underline{m_1} \wedge \underline{m_2} \wedge \underline{m_3} \wedge E \wedge E'$ is satisfiable. From this, we obtain condition (A1) and the first constraint of condition (A2).

For the second constraint of condition (A2), we assume that σ is an observable sequence such that $(N_1, m_1) \stackrel{\sigma}{\Rightarrow} (N_1, m'_1)$. Hence, using the fact that $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$, we have $(N_2, m_2) \stackrel{\sigma}{\Rightarrow} (N_2, m'_2)$ for every marking m'_2 of N_2 such that $m'_1 \equiv_E m'_2$. Using a similar

property from $(N_2, m_2) \sqsubseteq_{E'} (N_3, m_3)$, we have $(N_3, m_3) \stackrel{\sigma}{\Rightarrow} (N_3, m'_3)$ for every marking m'_3 of N_3 such that $m'_2 \equiv_E m'_3$. The result follows from the observation that, since E and E' are both solvable and the nets are compatible, for all markings m''_1 of N_1, if a marking m''_3 of N_3 satisfies $m''_1 \equiv_{\exists S.E \wedge E'} m''_3$ then there must be a marking m''_2 of N_2 such that both $m''_1 \equiv_E m''_2$ and $m''_2 \equiv_{E'} m''_3$. □

Preservation by Synchronous Composition

Our next result relies on the classical synchronous product operation between labeled Petri nets [LAV91]. Assume that $N_1 \triangleq (P_1, T_1, \mathrm{Pre}_1, \mathrm{Post}_1)$ and $N_2 \triangleq (P_2, T_2, \mathrm{Pre}_2, \mathrm{Post}_2)$ are two labeled Petri nets with respective labeling functions l_1 and l_2 on the respective alphabets Σ_1 and Σ_2. Without loss of generality, we can assume that the sets P_1 and P_2 are disjoint.

We introduce a new symbol, $\circ$, used to build (structured) names for transitions that are not synchronized. The *synchronous product* between N_1 and N_2, denoted as $N_1 \| N_2$, is the net $(P_1 \cup P_2, T, \mathrm{Pre}, \mathrm{Post})$ with labeling function l where T is the smallest set containing:

- transition $(t, \circ)$ if $l_1(t) \notin \Sigma_2$, such that $l((t, \circ)) \triangleq l_1(t)$;
- transition $(\circ, t)$ if $l_2(t) \notin \Sigma_1$, such that $l((\circ, t)) \triangleq l_2(t)$;
- and transition (t_1, t_2) if $l_1(t_1) = l_2(t_2) \neq \tau$, such that $l((t_1, t_2)) \triangleq l_1(t_1)$.

The flow functions of $N_1 \| N_2$ are such that $\mathrm{Pre}((t_1, t_2), p) \triangleq \mathrm{Pre}_1(t_1, p)$ if $p \in P_1$ and $t_1 \neq \circ$, or $\mathrm{Pre}_2(t_2, p)$ if $p \in P_2$ and $t_2 \neq \circ$ (and 0 in all the other cases). Similarly, for Post.

To simplify our proofs, we define a notion of projection over firing sequences of $N_1 \| N_2$, that is two functions $\varrho \cdot 1$ and $\varrho \cdot 2$ such that $\epsilon \cdot i = \epsilon$ and $(\varrho\, t) \cdot i = (\varrho \cdot i)\ (t \cdot i)$ for all $i \in 1..2$, where $(t_1, \circ) \cdot 1 = t$, and $(\circ, t_2) \cdot 1 = \epsilon$, and $(t_1, t_2) \cdot 1 = t_1$ (and symmetrically with $\cdot 2$ on the second component of each transition pair).

Projections can be used to extract from a firing sequence of $N_1 \| N_2$ the transitions that were fired from the left ($\cdot 1$) and right ($\cdot 2$) components of the synchronous product.

We also need to define a dual relation, denoted $\varrho_1 \| \varrho_2$, that defines the (potential) "zip merge" of firing sequences in $T_1^\star \times T_2^\star$ into firing sequences of $N_1 \| N_2$, when the two sequences can synchronize. When defined, $\varrho_1 \| \varrho_2$ is the smallest set of sequences of $N_1 \| N_2$ satisfying the following inductive rules. In particular, we say that ϱ_1 and ϱ_2 can be synchronized when $\varrho_1 \| \varrho_2 \neq \emptyset$.

- $\epsilon \| \epsilon \triangleq \{\epsilon\}$
- $(t_1\ \varrho_1) \| \epsilon \triangleq \begin{cases} \{(t_1, \circ)\ \varrho \mid \varrho \in (\varrho_1 \| \epsilon)\} & \text{if } l_1(t_1) \notin \Sigma_2, \\ \emptyset & \text{otherwise.} \end{cases}$
- $\epsilon \| (t_2\ \varrho_2) \triangleq \begin{cases} \{(\circ, t_2)\ \varrho \mid \varrho \in (\epsilon \| \varrho_2)\} & \text{if } l_2(t_2) \notin \Sigma_1, \\ \emptyset & \text{otherwise.} \end{cases}$

- $(t_1\ \varrho_1)\|(t_2\ \varrho_2) \triangleq \begin{cases} \{(t_1,t_2)\ \varrho \mid \varrho \in (\varrho_1\|\varrho_2)\} & \text{if } l_1(t_1) = l_2(t_2) \neq \tau\ , \\ \{(t_1,\circ)\ \varrho \mid \varrho \in \varrho_1\|(t_2\ \varrho_2)\} & \text{if } l_1(t_1) \notin \Sigma_2, \\ \{(\circ,t_2)\ \varrho \mid \varrho \in (t_1\ \varrho_1)\|\varrho_2\} & \text{if } l_2(t_2) \notin \Sigma_1, \\ \emptyset & \text{otherwise.} \end{cases}$

We can also project the reachable markings of a synchronous product over the reachable markings of each component. Since the places in N_1 and N_2 are disjoint, we can always see a marking m in $N_1\|N_2$ as the disjoint union of two (necessarily compatible) markings m_1, m_2 from N_1, N_2. In this case, we simply write $m = m_1\|m_2$.

More generally, we extend this product operation to marked nets and write $(N_1, m_1)\|(N_2, m_2)$ for the marked net $(N_1\|N_2, m_1\|m_2)$. The following result underscores the equivalence between the semantics (the Labeled Transition System) of $N_1\|N_2$ and the product of the LTS of its components.

Lemma 3.7 (Projection and Product of Sequences)**.** *Assume that there is a firing sequence $(N_1\|N_2, m_1\|m_2) \xrightarrow{\varrho} (N_1\|N_2, m_1'\|m_2')$ on the synchronous product $N_1\|N_2$. Then the projections $\varrho \cdot 1$ and $\varrho \cdot 2$ are firing sequences of their respective components, $(N_i, m_i) \xrightarrow{\varrho\cdot i} (N_i, m_i')$ for all $i \in 1..2$, such that $\varrho \cdot 1$ and $\varrho \cdot 2$ can be synchronized: $\varrho \cdot 1\|\varrho \cdot 2 \neq \emptyset$. Conversely, if $(N_i, m_i) \xrightarrow{\varrho_i} (N_i, m_i')$ for all $i \in 1..2$ and $\varrho \in (\varrho_1\|\varrho_2)$ then $(N_1\|N_2, m_1\|m_2) \xrightarrow{\varrho} (N_1\|N_2, m_1'\|m_2')$.*

Proof. See, for instance, Proposition 2.1 in [LAV91]. □

We can now prove that the E-abstraction equivalence is stable by synchronous composition.

Theorem 3.8 (Composability)**.** *Assume that $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$ and that (M, m) is disjoint from this equivalence then $(N_1, m_1)\|(M, m) \sqsubseteq_E (N_2, m_2)\|(M, m)$.*

Proof. By hypothesis, predicate E is solvable for N_1, N_2. Hence, since M is disjoint, no place in the net M can occur in one of the constraints of E. Therefore, E is also solvable for the pair of nets $(N_1\|M)$ and $(N_2\|M)$. Likewise, the initial markings $(m_1\|m)$ and $(m_2\|m)$ are compatible together and $(m_1\|m) \equiv_E (m_2\|m)$ (the constraints in m have no effect on the constraints of E). Therefore, condition (A1) is valid for the marked nets $(N_1, m_1)\|(M, m)$ and $(N_2, m_2)\|(M, m)$, and we obtain the first constraint of condition (A2).

We are left with proving the second constraint of condition (A2). Assume that ϱ is a firing sequence in $N_1\|M$. By our projection property (Lemma 3.7) it must be the case that $(N_1\|M, m_1\|m) \xrightarrow{\varrho} (N_1\|M, m_1'\|m')$ with $(N_1, m_1) \xrightarrow{\varrho\cdot 1} (N_1, m_1')$. We also have that $(M, m) \xrightarrow{\varrho\cdot 2} (M, m')$ such that $(\varrho \cdot 1)\|(\varrho \cdot 2) \neq \emptyset$.

By condition (A2) on the abstraction between N_1 and N_2, it must be the case that $(N_2, m_2) \xrightarrow{\varrho_2} (N_2, m_2')$, for some firing sequence ϱ_2 of N_2, for all markings m_2' of N_2 such

that $m'_1 \equiv_E m'_2$. Moreover, the observable sequence obtained from ϱ_2 and $\varrho \cdot 1$ are the same: $l_1(\varrho \cdot 1) = l_2(\varrho_2)$ $(\star)$, which means also that $(\varrho_2)\|(\varrho \cdot 2) \neq \emptyset$. Hence, using the second direction in Lemma 3.7, we can find a firing sequence in $\varrho_2\|(\varrho \cdot 2)$, say ϱ', such that $(N_2\|N_3, m_2\|m_3) \xrightarrow{\varrho'} (N_2\|M, m'_2\|m')$. Like in the proof of condition (A1), we obtain that $(m'_1\|m') \equiv_E (m'_2\|m')$ from the fact that $m'_1 \equiv_E m'_2$, and E is solvable, and M is disjoint.

We are left to prove that ϱ and ϱ' have the same observable sequences. This is a consequence of the fact that $l_1(\varrho \cdot 1) = l_2(\varrho_2)$ (property $(\star)$ above); and the fact that, by construction of ϱ', we have $\varrho' \cdot 1 = \varrho_2$ and $\varrho' \cdot 2 = \varrho \cdot 2$. □

Preservation by Relabeling

Another standard operation on labeled Petri nets is *relabeling*, denoted as $N[a/b]$, which applies a substitution to the labeling function of a net. Assume that l is the labeling function over the alphabet Σ. We denote $l[a/b]$ the labeling function on $(\Sigma \setminus \{a\}) \cup \{b\}$ such that $l[a/b](t) \triangleq b$ when $l(t) = a$ and $l[a/b](t) \triangleq l(t)$ otherwise. Then $N[a/b]$ is the same as net N but equipped with labeling function $l[a/b]$. Relabeling does not affect the marking of a net. The relabeling law is true even when b is the silent action τ. In this case, we say that we *hide* action a from the net.

We prove that E-abstraction equivalence is also preserved by relabeling and hiding.

Theorem 3.9. *If* $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$ *then* $(N_1[a/b], m_1) \sqsubseteq_E (N_2[a/b], m_2)$.

Proof. Assume that $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$. Condition (A1) does not depend on the labels, and therefore it is also true between $N_1[a/b], E$ and $N_2[a/b]$. For condition (A2), we use the fact that for any firing sequences ϱ_1 and ϱ_2, $l_1(\varrho_1) = l_2(\varrho_2)$ implies $l_1[a/b](\varrho_1) = l_2[a/b](\varrho_2)$. □

3.4.3 Running Examples

We can compute net reductions by reusing a tool called Reduce, which was developed in a previous work [BLD19]. The tool takes a marked Petri net as input and returns a reduced net and a sequence of linear constraints. For example, given the net M_1 of Fig. 3.1, Reduce returns net M_2 and equations $(p_5 = p_4), (a_1 = p_1 + p_2), (a_2 = p_3 + p_4)$, and $(a_1 = a_2)$, that correspond to formula E_M in Fig. 3.1 (if we forget about existential quantifiers).

The tool works by applying successive reduction rules in a compositional way. We give an example of this mechanism in Fig. 3.9, showing the four reduction steps involved in this running example.

The first step is a direct application of rule [RED] inside a larger context; in each case, we use colors to emphasize the subnet where the rule is applied. The two following ones are variations of rule [CONCAT]. Each rule introduces a fresh "place variable", a_1, and a_2. Finally, after simplification, we obtain a net with a new opportunity to apply a redundancy rule.

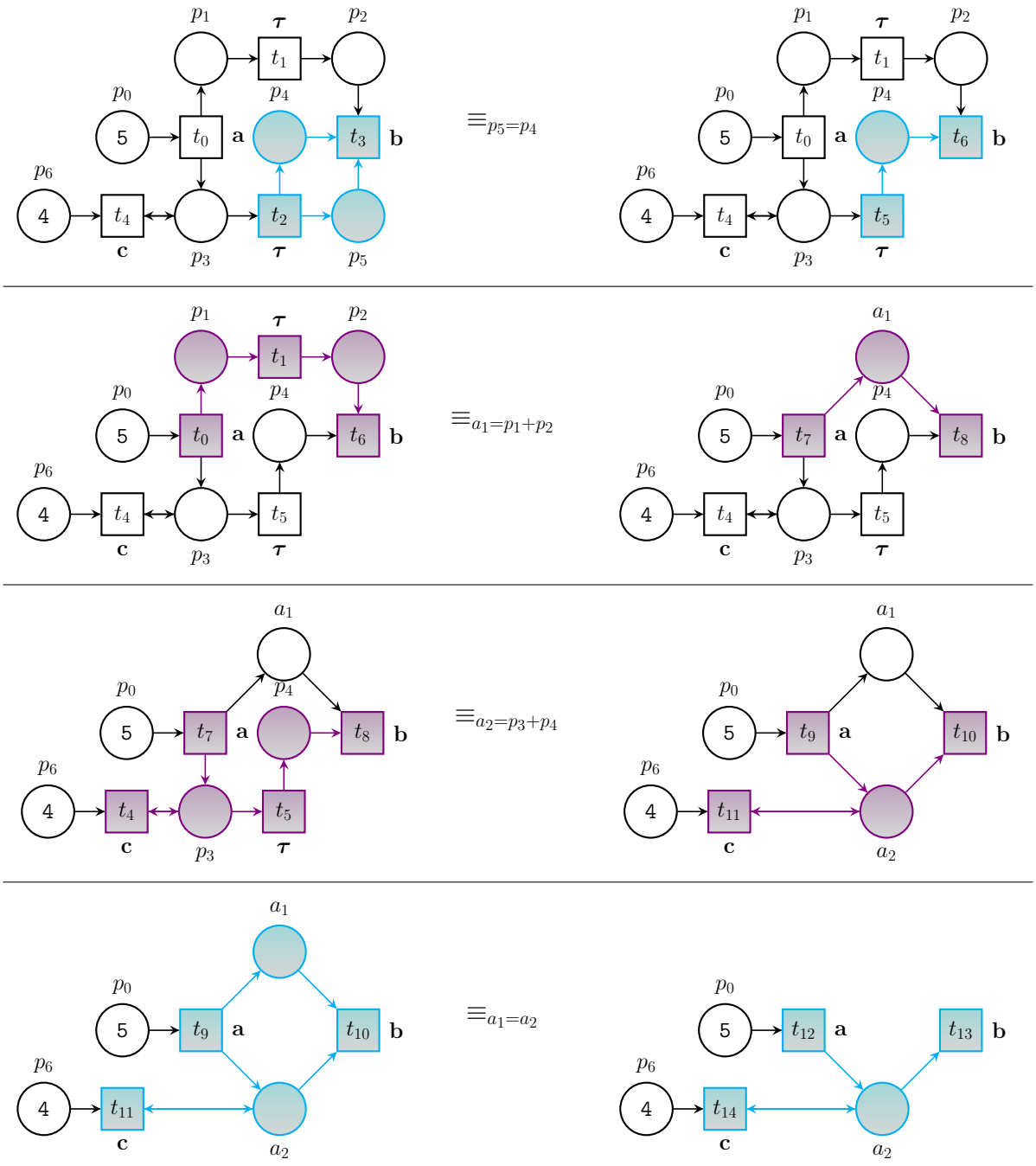


Fig. 3.9 Example of sequence of four reductions leading from the net M_1 to M_2 from Fig. 3.1.

It is possible to prove that each reduction step computed by Reduce, from a net (M_i, m_i) to (M_{i+1}, m_{i+1}) with constraints E_i, is such that $(M_i, m_i) \equiv_{E_i} (M_{i+1}, m_{i+1})$. From Theorem 3.6, we have $(M_0, m_0) \equiv_{\exists P_2 \setminus (P_1 \cup P_2).E} (M_n, m_n)$, i.e., the results computed by Reduce always translate into valid polyhedral equivalences.

Reduce can also perform some reduction steps that differ from structural reduction rules (such as the ones described in Sect 3.4.1). This still entails a correct polyhedral equivalence. For instance, the net depicted in Fig. 3.10 (left), from the MCC benchmark [Kor15], abstracts the lifecycle of a task in a simplified operating system handling the execution of tasks on a machine with several memory segments, disk controller units, and cores. The initial marking of the net gives the number of resources available (e.g., there are 8192 available memory segments in our example). A possible polyhedral reduction is depicted in Fig. 3.10 (right), where:

$$E \triangleq \exists a_1, a_2, a_3 \ . \begin{cases} \mathit{TaskOnDisk} = \mathit{DiskControllerUnit} + 4096 \\ a_1 = \mathit{CPUUnit} + \mathit{ExecutingTask} \\ a_2 = \mathit{TaskSuspended} + \mathit{ExecutingTask} \\ a_1 = 8192 \\ a_3 = a_2 + \mathit{TaskReady} \\ a_4 = \mathit{DiskControllerUnit} + \mathit{TransferToDisk} \\ a_5 = a_3 + \mathit{TransferToDisk} \end{cases}$$

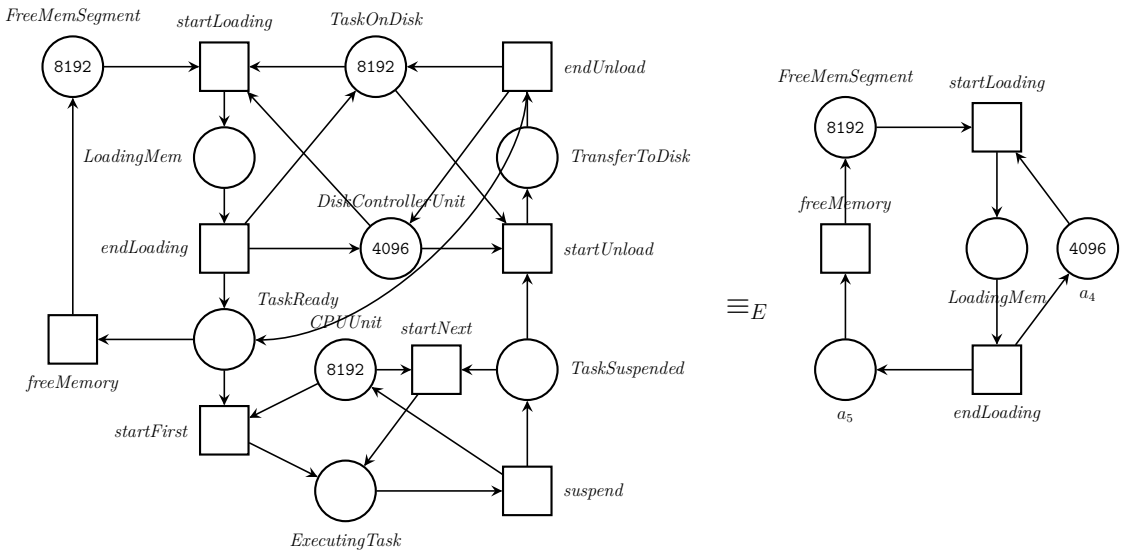


Fig. 3.10 The SmallOperatingSystem net (left) and an equivalent polyhedral reduction (right).

This reduction is obtained, as previously, by applying successive rules. However, the residual net is a live marked graph that is part of the PR-R class (see Sect. 1.5.4), i.e., the potentially reachable markings (solutions of the state equation) are indeed reachable. Hence, the residual net can be fully reduced, and the reachability set of the initial net corresponds to the solutions

of:

$$\exists a_4, a_5 \;.\; E \wedge \begin{cases} 4096 = a_4 + \mathit{LoadingMem} \\ 8192 = \mathit{FreeMemSegment} + \mathit{LoadingMem} + a_5 \end{cases}$$

In conclusion, we can use Reduce to compute polyhedral reductions automatically. In the other direction, we can use our notion of equivalence to prove the correctness of new reduction patterns that could be added to the tool. While it is not always possible to reduce the complexity of a net using this approach, we observed in our experiments (Sect. 3.7) that, on a benchmark suite that includes almost 1 400 instances of nets, about half of them can be reduced by a factor of more than 30%.

3.5 SMT-Based Model Checking Using Reductions

We introduce a general method for combining polyhedral reductions with SMT-based procedures. Assume that we have $(N_1, m_1) \equiv_E (N_2, m_2)$, where the nets N_1, N_2 have sets of places P_1, P_2 respectively. In the following, we use $\boldsymbol{p_1} \triangleq (p_1^1, \ldots, p_k^1)$ and $\boldsymbol{p_2} \triangleq (p_1^2, \ldots, p_l^2)$ for the places in P_1 and P_2. We also consider (disjoint) sequences of variables, $\boldsymbol{x}$ and $\boldsymbol{y}$, ranging over (the places of) N_1 and N_2. With these notations, we denote $\tilde{E}(\boldsymbol{x}, \boldsymbol{y})$ the formula obtained from E where place names in N_1 are replaced with variables in $\boldsymbol{x}$, and place names in N_2 are replaced with variables in $\boldsymbol{y}$. When we have the same place in both nets, say $p_i^1 = p_j^2$, we also add the constraint $(x_i = y_j)$ to $\tilde{E}$ in order to avoid shadowing variables. (Remark that $\tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2})$ is equivalent to E, since equalities $x_i = y_j$ become tautologies in this case.)

$$\tilde{E}(\boldsymbol{x}, \boldsymbol{y}) \triangleq E\{\boldsymbol{p_1} \leftarrow \boldsymbol{x}\}\{\boldsymbol{p_2} \leftarrow \boldsymbol{y}\} \wedge \bigwedge_{\{(i,j) | p_i^1 = p_j^2\}} (x_i = y_j) \tag{3.2}$$

Assume that F_1 is a property that we want to study on N_1, such that $\mathrm{FV}(F_1) \subseteq P_1$. We construct an equivalent formula F_2, to study on N_2, which we call the E-transform formula of F_1.

Definition 3.3 (E-Transform Formula)**.** *Assume that $(N_1, m_1) \equiv_E (N_2, m_2)$ and take F_1 a property with variables in P_1, i.e., $\mathrm{FV}(F_1) \subseteq P_1$. Formula $F_2(\boldsymbol{y}) \triangleq \exists \boldsymbol{x} \;.\; \tilde{E}(\boldsymbol{x}, \boldsymbol{y}) \wedge F_1(\boldsymbol{x})$ is the E-transform of F_1.*

The following property states that, to check F_1 reachable in N_1, it is enough to check the corresponding E-transform formula F_2 on N_2.

Theorem 3.10 (Reachability Conservation)**.** *Assume that $(N_1, m_1) \equiv_E (N_2, m_2)$ and that F_2 is the E-transform of formula F_1 on N_1. Then, formula F_1 is reachable in N_1 if and only if F_2 is reachable in N_2.*

Proof. Assume that $(N_1, m_1) \equiv_E (N_2, m_2)$ and that property F_1 is reachable in N_1. Hence, a reachable marking m'_1 in N_1 exists such that $m'_1 \models F_1$. By definition of E-abstraction, we have at least one reachable marking m'_2 in N_2 such that $m'_1 \equiv_E m'_2$. The condition $m'_1 \equiv_E m'_2$ is equivalent to $\underline{m'_1} \wedge \underline{m'_2} \wedge E$ satisfiable. By definition, we have $\tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \equiv E$, which implies $\underline{m'_1}(\boldsymbol{p_1}) \wedge \underline{m'_2}(\boldsymbol{p_2}) \wedge \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge F_1(\boldsymbol{p_1})$ satisfiable, since the only variables that are both in F_1 and E must also be places of N_1. Hence, m'_2 satisfies the E-transform formula of F_1.

Now assume that F_2 is reachable in N_2, i.e., there exists a reachable marking m'_2 in N_2 such that $m'_2 \models F_2$. Since $m'_2 \models \exists \boldsymbol{x} \,.\, \tilde{E}(\boldsymbol{x}, \boldsymbol{y}) \wedge F_1(\boldsymbol{x})$, we can exhibit a marking m'_1 such that $\models \tilde{E}(m'_1, m'_2)$—that entails $m'_1 \equiv_E m'_2$—, and $\models F_1(m'_1)$. By definition of the E-abstraction (condition (A2)), we have m'_1 reachable.

Since the relation $\equiv_E$ is symmetric, the proof is similar in the other direction. □

Since F_1 invariant on N_1 is equivalent to $\neg F_1$ not reachable, we can directly infer an equivalent conservation theorem for invariance:

Corollary 3.11 (Invariant Conservation)**.** *Assume that* $(N_1, m_1) \equiv_E (N_2, m_2)$ *and that* F_2 *is the* E*-transform of formula* $\neg F_1$ *on* N_1*. Then* F_1 *is an invariant on* N_1 *if and only if* $\neg F_2$ *is an invariant on* N_2*.*

Note that $\neg F_2$ *is actually* $\forall \boldsymbol{x} \,.\, \tilde{E}(\boldsymbol{x}, \boldsymbol{y}) \implies F_1(\boldsymbol{x})$*.*

Negating the E-transform formula, as done in Corollary 3.11, introduces universally quantified variables that may impact the solver performance since we require the "full" LIA theory instead of only the quantifier-free fragment. In Chapter 5 we show how to get around this problem using by introducing a quantifier elimination procedure.

Note also that given a formula F_1, the E-transform of $\neg F_1$ is usually not equal to the negation of the E-transform of F_1. We always compute the E-transform formula of the formula we want to show reachable (F in EF F or AG $\neg F$).

3.6 Combining Polyhedral Reduction with BMC

We first developed the model checker SMPT for taking advantage of polyhedral reduction. In addition to PDR, the tool includes the verification procedures presented in Sect. 1.5 (except CEGAR) developed for generalized Petri nets. (No specific optimizations are applied when we know the net is safe, like, for instance, using Boolean formulas instead of QF-LIA.) We describe here our adaptation of BMC.

Bounded Model Checking (BMC)

We already described the BMC method in Sect. 1.5.2, an iterative method for exploring the state space of systems by unrolling their transitions. In BMC, we compute formulas ϕ_i that

represent the set of markings reachable using firing sequence of length at most i until $\phi_i \wedge F$ is satisfiable, e.g., we found a witness (or counter-example).

We aim not to develop a state-of-the-art BMC model checker for generalized Petri nets. Instead, we develop a textbook implementation that is enough to test the impact on performance when using reductions. BMC can rely on optimization techniques, such as compositional reasoning; acceleration methods; or invariants on the underlying model to add extra constraints. We do not consider such optimizations here, on purpose, since our motivation is to study the impact of polyhedral reduction.

We believe that our use of reductions is orthogonal and does not overlap with many of these optimizations because we do not preclude them. We show that this conjecture holds in Chapter 5 where we combine polyhedral reductions with the best-performing tools at the MCC.

Combination with Polyhedral Reduction

Assume that we have $(N_1, m_1) \equiv_E (N_2, m_2)$. We denote $\mathrm{T}_1, \mathrm{T}_2$ the equivalent of the transition relation predicate T, Equation (1.10) from Sect. 1.2.3, for the nets N_1, N_2 respectively. We also use $\boldsymbol{x}$, $\boldsymbol{y}$ for sequences of variables ranging over (the places of) N_1 and N_2, respectively. We shall use $\phi(N_1, m_1)$ for the family of formulas built using operator T_1 and variables $\boldsymbol{x_0}, \boldsymbol{x_1}, \ldots$ and similarly for $\phi(N_2, m_2)$, where we use T_2 and variables of the form $\boldsymbol{y_i}$.

The following property states that, to find a model of F in the reachable markings of N_1 (meaning EF F true), it is enough to find a model for its E-transform in N_2.

Theorem 3.12 (BMC with E-transform)**.** *Assume that* $(N_1, m_1) \equiv_E (N_2, m_2)$ *and that* F_2 *is the* E*-transform of* F_1*. Formula* F_1 *is reachable in* N_1 *if and only if there exists* $j \geqslant 0$ *such that* $F_2(\boldsymbol{y}_j) \wedge \phi_j(N_2, m_2)$ *is satisfiable.*

Proof. Our proof relies on the property that BMC is sound and complete for finding a finite witness (see, e.g., [Cim+16]): there is a firing sequence ϱ, of size less than i, such that $m_1 \xrightarrow{\varrho} m_1'$ and $m_1' \models F_1$—meaning property F_1 is reachable in N_1— if and only if $F_1 \wedge \phi_i(N_1, m_1)$. We can prove this property by induction on the value of i and use the fact that $m \to m'$ or $m = m'$ in N_1 entails $\mathrm{T}_1(m, m')$.

By our *reachability conservation* theorem (Theorem 3.10), property F_1 is reachable in N_1 (say with a witness sequence of size i) if and only if property F_2 is reachable in N_2 (say with a witness sequence of size j). Therefore, there exists i such that $F_1(\boldsymbol{x_i}) \wedge \phi_i(N_1, m_1)$ is satisfiable if and only if there exists j such that $F_2(\boldsymbol{y_j}) \wedge \phi_j(N_2, m_2)$ is satisfiable. □

We can give a stronger result, comparing the value of i and j, when the reductions used in computing the E-abstraction equivalence never introduce new transitions. This is the case, for example, with the reductions computed using the Reduce tool. Indeed, in this case, we can show that we may find a witness of length i in N_1 (a firing sequence of length i showing that

F_1 is reachable in N_1) when we find a witness of length $j \leqslant i$ in N_2. This is because, in this case, reductions may compact a sequence of several transitions into a single one or, at worst, not change it. Take the example of the [CONCAT] rule in Fig. 3.9. Therefore, BMC benefits from reductions in two ways. First, we can reduce the size of formulas ϕ (proportional to the net's size), and second, we can accelerate transition unrolling in the reduced net.

3.7 Experimental Results

We have implemented the approach described in Sect. 3.6 into our tool SMPT. In this section, we report on some experimental results obtained with SMPT on the extensive benchmark of models and formulas provided by the 2023 edition of the Model Checking Contest (MCC).

SMPT does not compute net reductions directly but relies on the tool Reduce, distributed with the standard distribution of the Tina toolbox [BRV04; LAA23]. We also provide an open-source, feature-complete version of an equivalent tool, called Shrink [Cha22], which provides several Rust libraries for manipulating Petri nets and performing structural reductions.

A complete description of the benchmark and toolchain can be found in Chapter 8.

3.7.1 Distribution of Reduction Ratios

Since our approach relies on net reductions, it is natural to wonder if reductions occur in practice. To answer this question, we computed the reduction ratio (r), obtained using Reduce, as a quotient between the number of places deleted ($p_{\text{init}} - p_{\text{red}}$) and the initial number (p_{init}): $r \triangleq (p_{\text{init}} - p_{\text{red}})/p_{\text{init}}$. We display in Fig. 3.11 the results for the whole collection of instances in the MCC, sorted in descending order.

A ratio of 100% ($r = 1$) means that the net is *fully reduced*; the resulting net has only one (empty) marking. We see a surprisingly high number of models that are fully reducible with our approach (about 14% of the total number), with approximately half of the instances that can be reduced by a ratio of 30% or more.

We evaluated the performance of SMPT using the formulas of the MCC'2023 on a selection of 1 145 Petri nets (80% of the benchmark) taken from instances with a reduction ratio greater than 1%.

A pair of an instance and a formula is called a *test case*. For each test case, we check the formulas with and without the help of reductions using BMC and with a fixed timeout of 180 s. We selected queries that can be computed with BMC, that is, queries EF F true or AG F false. This adds up to a total of 9 989 *test cases*, which required the equivalent of 565 hours of CPU time.

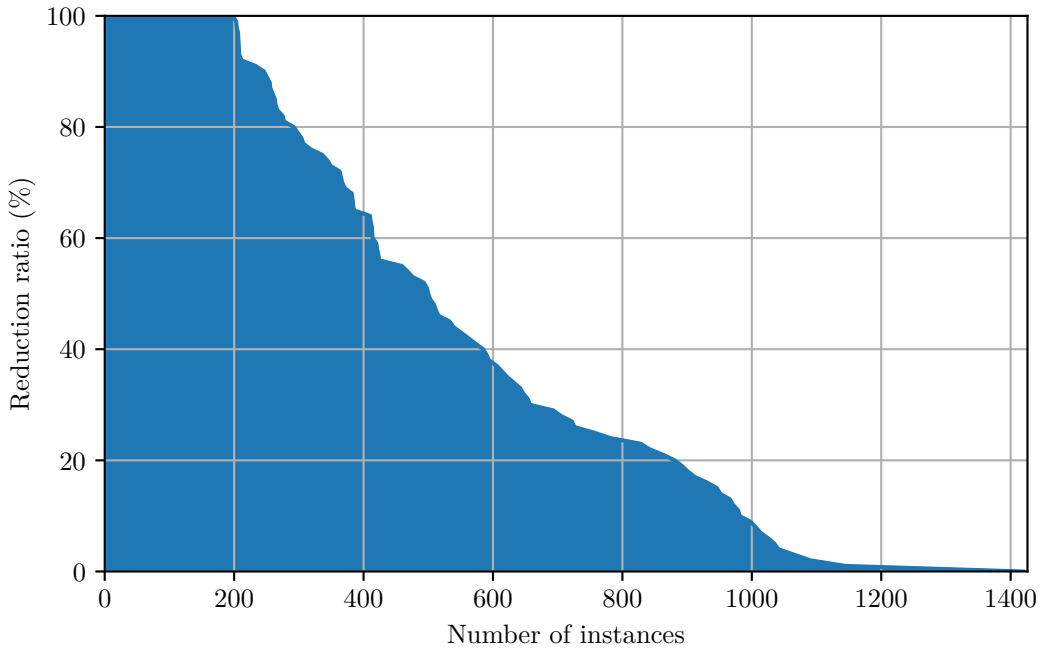


Fig. 3.11 Distribution of reduction ratios over the instances in the MCC.

3.7.2 Impact on the Number of Solvable Queries

We report our results in Table 3.1. Out of the almost 10 000 test cases, we could compute 6 423 results using reductions and only 2 903 without reductions (approximately twice more).

We compared our results with the ones provided by an *oracle*, which gives the expected answer (as computed by a majority of tools, using different techniques, during the MCC competition). We achieve 100% reliability on the benchmark, meaning we always give the answer predicted by the oracle.

We give the number of computed results for four different categories of test cases: *Full* contains only the fully reducible instances (the best possible case with our approach), while *Low*/*Good*/*High* correspond to instances with a low/moderate/high level of reduction. We chose the limits for these categories to obtain samples with comparable sizes. We also have a general category, *All*, for the complete set of benchmarks.

We observe that we can compute almost twice as many results when we use reductions than without. This gain is more significant on the *High* (×2.6) than on the *Good* (×1.57) instances. Nonetheless, the fact that the number of additional queries solved using reductions is still substantial, even for a reduction ratio under 50%, indicates that our approach can benefit from all the reductions we can find in a model (and that numerous fully reducible instances do not skew our results).

In the case of *fully reducible* nets, checking a query amounts to solving a linear predicate on the initial marking of the reduced net. There are no iterations. For this category, we can

Reduction Ratio (r)		# Test Cases	# Computed Queries		
			With reductions	Without	
All	$r \in [0.01, 1]$	9 989	6 423	2 903	×2.21
Low	$r \in [0.01, 0.25[$	3 111	1 569	1 277	×1.22
Good	$r \in [0.25, 0.5[$	2 302	1 198	762	×1.57
High	$r \in [0.5, 1[$	4 748	2 724	1046	×2.6
Full	$r = 1$	2 130	2 130	580	×3.67

Table 3.1 Impact of the reduction ratio on the number of solved instances.

compute a result for all the queries, and most of these queries can be solved in less than a few seconds (99% in less than 1 s).

3.7.3 Impact on Computation Time

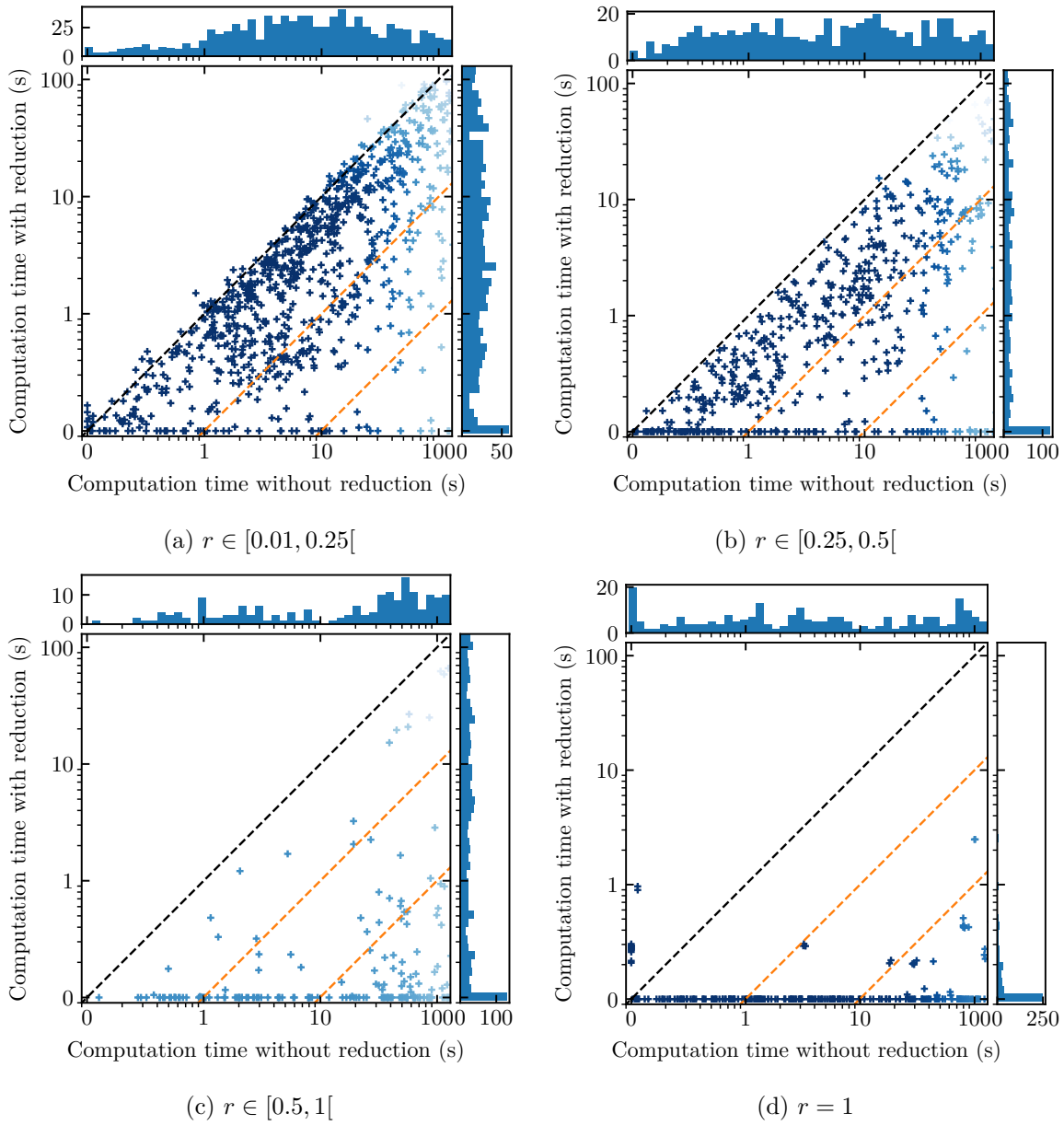
To better understand the impact of reductions on the computation time, we compare the computation time, with or without reductions, for each test case. These results do not consider the time spent for reducing each instance. This time is negligible compared to each test, usually in the order of 1 s. Also, we only need to reduce the net once when checking different properties for the same instance (16 during the MCC).

We display our results in Fig. 3.12, where we give four scatter plots comparing the computation time "with" (y-axis) and "without" reductions (x-axis), for the *Low*, *Good*, *High* and *Full* categories of instances. Each chart uses a logarithmic scale. We also display a histogram for each axis on the charts that gives the density of points for a given duration. To avoid overplotting, we removed all the "trivial" properties (the bottom left part of the chart) that can be computed with and without reduction in less than 10 ms. These "trivial" queries (994) correspond to instances with a small state space or situations where a counter-example can be found quickly.

We observe that almost all the data points are below the diagonal, meaning reductions accelerate the computation, with many test cases exhibiting speed-ups larger than ×100. We have added two light-colored, dashed lines to materialize data points with speed-ups larger than ×10 and ×100, respectively.

On our 9 989 test cases, we timeout with reductions but compute a result without on only 2 cases. These exceptions can be explained by border cases where the order in which transitions are processed has a sizeable impact.

Another interesting point is the ratio of properties that can be computed only using reductions. This is best viewed when looking at the histogram values. A vast majority of the points in the charts are either on the right border (computation without reductions timeout) or on the x-axis (they can be computed in less than 10 ms using reductions).



(a) $r \in [0.01, 0.25[$

(b) $r \in [0.25, 0.5[$

(c) $r \in [0.5, 1[$

(d) $r = 1$

Fig. 3.12 Comparing computation time, "with" (y-axis) and "without" (x-axis) reductions for categories *Low* (a), *Good* (b), *High* (c) and *Full* (d).

3.8 Discussion

We propose a new method to combine structural reductions with SMT solving to check properties on arbitrary Petri nets. While this idea is not original, the framework we developed is new. Our main innovation resides in the use of a principled approach, where we can trace back reachable markings (between an initial net and its residual) through a conjunction of linear equalities (the formula $\tilde{E}$). Basically, we show that we can adapt an SMT-based procedure for checking a property on a net (that relies on computing a family of formulas of the form $(\phi_i)_{i\in I}$) into a procedure that relies on a reduced version of the net and formulas of the form $(\phi_i \wedge \tilde{E})_{i\in J}$.

As a proof of concept, we apply our approach to a basic implementation of the BMC procedure. Our empirical evaluation shows promising results. For example, we can compute twice as many results using reductions than without. In Chapter 5 we show that our approach can be adapted to any decision procedure or tool by eliminating variables in $E \wedge F$ that are not places of the reduced net N_2, and therefore act as existentially quantified variables in the E-transform formula.

Our main theoretical results (the conservation theorems of Sect. 3.5) can be interpreted as examples of *reduction theorems* [Lip75; CL98], that allow to deduce properties of an initial model (N) from properties of a simpler, coarser-grained version (N^R). While these works are related, they mainly focus on reductions where one can group a sequence of transitions into a single atomic action. Hence, in our context, they correspond to a restricted class of reductions, similar to a subset of the agglomeration rules used in [BLD19].

We can also mention approaches where the system is simplified with respect to a given property, for instance, by eliminating parts that cannot contribute to its truth value, like with the *slicing* [Wei84] or *cone of influence* [CGP99] abstractions used in some model checkers. Finding such "parts" (places and transitions) in a Petri net is not always easy, especially when the formula involves many places. This is not a problem with our approach since we can always abstract away a place as long as its effect is preserved in the E-transform formula.

In [Sch03], the author uses net invariants to compress the representation of markings. This approach is based on the fact that place invariants provide linear constraints between the markings of several places, like in our use of redundancy rules. However, the goal is to reduce the memory footprint when computing the explicit state space while verification is still performed on "uncompressed" markings. On the contrary, our approach can be used with symbolic methods—working on a reduced version of the net—and can use more general rules. For instance, it cannot benefit from rules that agglomerate places.

In practice, we derive polyhedral equivalences using *structural reductions*, a concept introduced by Berthelot in [Ber87]. In our work, we are interested in reductions preserving reachable states. This contrasts with most works about reductions, where more powerful transformations can be applied when focusing on specific properties, such as the absence of deadlocks.

Several tools use reductions for checking reachability properties. TAPAAL [Dav+12], for instance, is an explicit-state model checker that combines partial order reduction techniques and structural reductions and can check properties on Petri nets with weighted arcs and inhibitor arcs [Bøn+19]. A more relevant example is ITS-Tools [Thi15], which combines several techniques, including structural reductions and using SAT and SMT solvers [Thi20; Thi21]. But, it has to be kept in mind, though, that our goal is to study the impact of polyhedral reduction in isolation from other techniques. A combination of polyhedral reduction with ITS-Tools, TAPAAL and LoLA [Wol18] is proposed in Chapter 5.

In the following chapters, we extend our polyhedral framework. In particular, we apply our approach to the verification of properties more complex than reachability, like the concurrent places problem (Chapter 6). The problem, in this case, is to enumerate all pairs of places that can be marked together for some reachable states. For this work, we rely on a new data structure that precisely captures the structure of reduction constraints, what we call the *Token Flow Graph* (Chapter 4).

On a more theoretical side, we also identified a need to develop an automated method to prove the correctness of new reduction rules (see Chapter 7). This procedure permits understanding the central philosophy of polyhedral reduction: to capture "flat" sub-parts of nets, i.e., subnets with Presburger-definable reachability sets.

This work has been published in:

- N. Amat, B. Berthomieu, and S. Dal Zilio. "On the Combination of Polyhedral Abstraction and SMT-Based Model Checking for Petri Nets". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. vol. 12734. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-76983-3_9
- N. Amat, B. Berthomieu, and S. Dal Zilio. "A Polyhedral Abstraction for Petri Nets and its Application to SMT-Based Model Checking". In: *Fundamenta Informaticae* 187.2-4 (2022), pp. 103–138. DOI: 10.3233/FI-222134

The tool related to this chapter is:

- SMPT https://github.com/nicolasAmat/SMPT

Chapter 4

Token Flow Graphs

Definition and Application for Marking Reachability

> The first law of computer science: Every problem is solved by yet another indirection.
>
> Bjarne Stroustrup

In this chapter, we propose a data structure called Token Flow Graph (TFG) that captures the particular structure of constraints stemming from polyhedral reductions. To illustrate the use of this new data structure, we propose to accelerate the reachability check of a given marking.

Our data structure and algorithm are implemented in a tool, called Kong, that we evaluate on the collection of models used in the Model Checking Contest. As when combined with BMC, our experiments show that the approach works well, even when a moderate amount of reductions applies.

4.1 Introduction

This chapter follows on directly from Chapter 3 by introducing Token Flow Graphs. We reuse this new data structure in Chapter 5 to accelerate our main problem of interest, the generalized reachability problem, by eliminating existentially quantified variables in the E-transform formula $F_2(\boldsymbol{p_2}) \triangleq \exists \boldsymbol{p_1} \ . \ \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge F_1(\boldsymbol{p_1})$, including those that may appear in $\tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2})$; and in Chapter 6, by accelerating a more complex reachability problem, the concurrent places problem.

Context. In Chapter 3, we introduced polyhedral reduction and used it on symbolic model checking to accelerate the verification of "generalized" reachability properties, in the sense that

we check whether it is possible to reach a marking that satisfies a property F expressed as a Boolean combination of linear constraints between places, such as $(p_0 + p_1 = p_2 + 1) \wedge (p_0 \leqslant p_2)$ for example.

This resulted in a new relation between nets, $(N_1, m_1) \equiv_E (N_2, m_2)$, called polyhedral equivalence. One of the main results is that, given a reachability formula F_1 with support on the initial net N_1 it is possible to build a formula F_2, called the E-transform formula of F_1, such that F_1 is reachable in N_1 if and only if F_2 is reachable in N_2.

Challenge. The challenge here is to capture the particular structure of the constraints in E using a graph structure. To demonstrate the versatility of this approach, we want to apply it to two specific reachability problems: first, to check the *reachability* of a given marking (as an illustration in this chapter); second, to compute the *concurrency relation* of a net (Chapter 6), that is all pairs of places that can be marked together in some reachable marking.

Regarding marking reachability, we consider the simple problem of checking whether a given marking m'_1 is reachable by firing a sequence of transitions in a net N_1, starting from an initial marking m_1. We want to use TFGs to prove a stronger property than in Chapter 3 for the marking reachability problem, namely that, given a target marking m'_1 for N_1, we want to effectually compute a marking m'_2 of N_2 such that m'_1 is reachable in N_1 if and only if m'_2 is reachable in N_2.

This can be more efficient than our previous method since the E-transform formula $F_2(\boldsymbol{p_2}) \triangleq \exists \boldsymbol{p_1} \,.\, \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge F_2(\boldsymbol{p_1})$ can be quite complex in practice, even though the property for marking reachability is a simple conjunction of equality constraints. For instance, we cannot perform an explicit-state model checker since we need to solve an integer linear problem for each new state instead of just evaluating a closed formula. Also, given a marking m'_1 to be checked on N_1, as things stand, there is nothing to prevent the existence of different markings m'_2 related to m'_1 up-to E.

Proposal. Our algorithm relies on a new data structure, called a *Token Flow Graph* (TFG), that captures the particular structure of constraints occurring in the predicate E. We describe TFGs and show how to leverage this data structure in order to accelerate the computation of solutions for the two reachability problems we mentioned: (1) marking reachability and (2) concurrency relation (Chapter 6). We use the term acceleration to stress the "multiplicative effect" of TFGs. Indeed, we propose a framework that, starting from a tool for solving the problem (1) or (2), provides an augmented version of this tool that takes advantage of reductions. The augmented tool can compute the solution for an initial instance, say on some net N_1, by solving it on a reduced version of N_2 and then reconstructing a correct solution for the initial instance. In each case, our approach takes the form of an "inverse transform" that relies only on E and does not involve expensive preprocessing on the reduced net.

For the marking reachability problem, we illustrate our approach by augmenting the tool Sift, an explicit-state model checker for Petri nets part of the Tina toolbox [BRV04; LAA23] that can check reachability properties on the fly.

Outline and Contributions Sections 4.3 and 4.4 contain our main contributions. We describe Token Flow Graphs (TFGs) in Sect. 4.3 and prove several results about them in Sect. 4.4. These results allow us to reason about the reachable markings of a net by playing a "token game" on the nodes of a TFG. In Sect. 4.5, we use TFGs to define a decision procedure for the reachability problem.

Our approach has been implemented, and computing experiments show that reductions are effective on a large set of models (Sect. 4.6). We observe that even with a moderate amount of reductions, we can check the marking reachability much faster with reductions than without, often by several orders of magnitude.

4.2 Polyhedral Equivalence Relaxation

In Definition 3.2, we defined the notion of *polyhedral equivalence*, $(N_1, m_1) \equiv_E (N_2, m_2)$, which uses the observable sequences of nets (N_1, m_1) and (N_2, m_2), and E a Presburger predicate with support on the places of N_1 and N_2 (e.g., $\mathrm{FV}(E) \subseteq P_1 \cup P_2$). For this current work, we do not need to exhibit these sequences. This motivates the use of a simplified, relaxed version of equivalence, which entails an equivalence between the state space of two nets, (N_1, m_1) and (N_2, m_2), "up-to" a predicate E; without conditions on the observable sequences.

Definition 4.1 (Relaxed E-Equivalence)**.** *Assume that* $N_1 \triangleq (P_1, T_1, \mathrm{Pre}_1, \mathrm{Post}_1)$ *and* $N_2 \triangleq (P_2, T_2, \mathrm{Pre}_2, \mathrm{Post}_2)$ *are two Petri nets and that* E *is a Presburger predicate whose free variables are in* $P_1 \cup P_2$*. We say that* (N_2, m_2) *is* E*-equivalent to* (N_1, m_1) *with the relaxed relation, denoted* $(N_1, m_1) \doteq_E (N_2, m_2)$*, if and only if:*

(A1) *initial markings are related up-to* E*, meaning* $m_1 \equiv_E m_2$*;*
(A2a) $E \wedge \underline{m}$ *is satisfiable for all markings* m *in* $R(N_1, m_1)$ *or* $R(N_2, m_2)$*;*
(A2b) *assume that* m'_1, m'_2 *are markings of* N_1, N_2 *such that* $m'_1 \equiv_E m'_2$*, then* m'_1 *is reachable if and only if* m'_2 *is reachable:* $m'_1 \in R(N_1, m_1) \Leftrightarrow m'_2 \in R(N_2, m_2)$*.*

In practice, given a relation $(N_1, m_1) \doteq_E (N_2, m_2)$, if E satisfies some well-formedness condition then each marking m'_2 reachable in N_2 can be associated with a unique subset of markings reachable in N_1, defined from the solutions to $E \wedge \underline{m'_2}$ (by conditions (A2a) and (A2b)). We show in Theorem 4.8 that this gives a partition of the reachable markings of (N_1, m_1) into "convex sets", each associated to a reachable marking in N_2. By construction, the relaxed relation $\doteq_E$ is directly implied by $\equiv_E$. In the following chapters, in order to simplify the notation, the symbol $\equiv_E$ stands for $\doteq_E$.

Proposition 4.1. *If* $(N_1, m_1) \equiv_E (N_2, m_2)$ *then* $(N_1, m_1) \doteq_E (N_2, m_2)$.

Proof. Condition (A1) for both relations is identical. Conditions (A2a) and (A2b) of the relaxed E-equivalence (Definition 4.1) are a split of condition (A2) of the E-equivalence (Definition 3.2) by only considering the reachability sets and omitting the observable sequences. □

The example in Fig. 4.1 (used as a running example in the previous chapter) is representative of the "shape" of reduction predicates: it mainly contains equalities of the form $x = \sum x_i$ over a sparse set of variables but may also include some inequalities. It can have a very large number of literals (often proportional to the size of the initial net). Another interesting feature is the absence of cyclic dependencies, which underlines a hierarchical relation between variables.

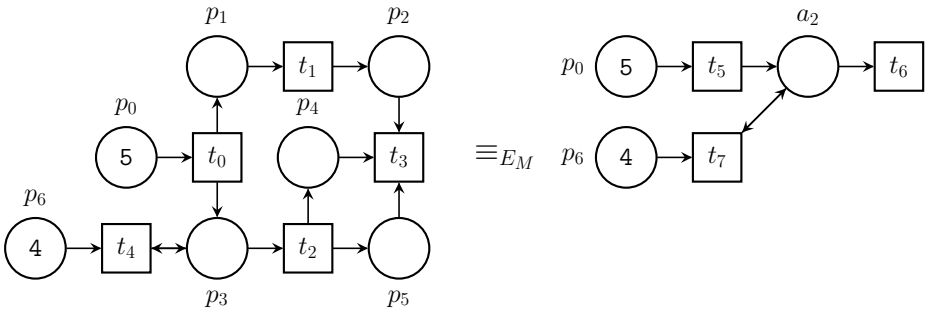


Fig. 4.1 An example of Petri net (Fig. 1.1), (M_1, m_1) (left), and one of its polyhedral reduction, (M_2, m_2) (right), with $E_M \triangleq \exists a_1 \,.\, (p_5 = p_4) \wedge (a_1 = p_1 + p_2) \wedge (a_2 = p_3 + p_4) \wedge (a_1 = a_2)$.

In this chapter, we restrict the predicate E to the equations we obtain with our reduction system (as the one implement in the tool Reduce), which we call *linear system form* in Definition 4.2. Applying successive reductions steps (by Theorem 3.6) always leads to a formula of the form $E = \exists Q \,.\, \phi_1 \wedge \cdots \wedge \phi_n$ where ϕ_i's are equalities. Existential variables in Q correspond to the freshly introduced places not in N_1 that have been removed afterward (also from Theorem 3.6); or to slack variables introduced for encoding inequalities. For instance, it is possible to encode $p \leqslant K$ from the [SOURCE] rule (Fig. 3.8) into $K = p + s$ with s an additional variable.

Definition 4.2 (Linear System Form)**.** *Given an equivalence statement* $(N_1, m_1) \equiv_E (N_2, m_2)$ *we say that the Presburger predicate* E *is in linear system form if and only if we have* $E = \exists Q \,.\, E'$*, with* Q *a set of variables such that* $Q \cap (P_1 \cup P_2) = \emptyset$*,* $E' \triangleq \bigwedge_{i \in 1..n} \phi_i$*, and each equality* ϕ_i *is of the form* $\alpha_i = \Sigma_{j \in I_i} \beta_j$ *where* α_i, β_j *in* $\mathbb{N} \cup Q \cup P_1 \cup P_2$.

In this chapter, as well as in Chapters 5 and 6, E is in linear system form by construction. Thus, we simply write $E \triangleq \exists Q.E'$ to indicate that Q and E' are as defined in Definition 4.2

above. One of the goals of Token Flow Graphs is to abstract away the variables introduced by the quantification $\exists Q$.

4.3 Token Flow Graphs

We introduce a set of structural constraints on the equations occurring in an equivalence statement $(N_1, m_1) \equiv_E (N_2, m_2)$. The goal here is to define a data structure that permits to answer to reachability problems on N_1, given a result on N_2, by taking advantage of the structure of the equations in E. Our method is tailored to the specific kind of constraints that occur in polyhedral reductions. As stated in Definition 4.2, all our equations will be of the form $x = y_1 + \cdots + y_l$ or $y_1 + \cdots + y_l = k$ (with k a constant in $\mathbb{N}$).

We define the *Token Flow Graph* (TFG) of a system $E \triangleq \exists Q.E'$ in linear system form as a Directed Acyclic Graph (DAG) with one vertex for each variable occurring in E'. Arcs in the TFG are used to depict the relation induced by equations in E'. We consider two kinds of arcs, redundancy ($\rightarrow\!\!\bullet$) and agglomeration ($\circ\!\!\rightarrow$), corresponding to two main reduction rule classes.

Arcs for *redundancy equations*, $q \rightarrow\!\!\bullet p$, represent equations of the form $p = q$ (or $p = q + r + \ldots$), expressing that place p can be removed and that its marking can be reconstructed from the marking of $q, r, \ldots$. Figure 4.2 illustrates such reduction rules on a subpart of the net M_1 given in Fig. 4.1. In this case, place p_4 has the same Pre and Post relations than p_5; thus, both places are redundant. And so, by removing place p_5, we obtain the TFG on the right, corresponding to the equation $p_5 = p_4$ (modeled by a "black dot" arc).

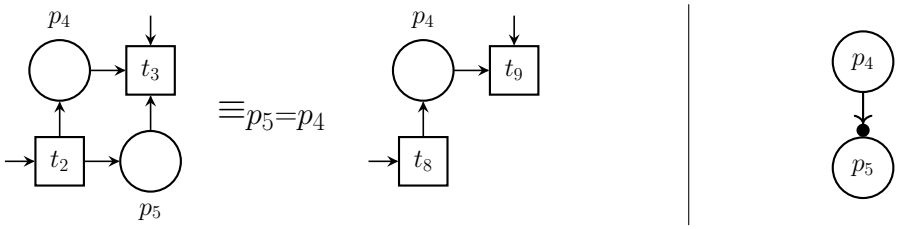


Fig. 4.2 Redundancy reduction applied on a subpart of the net M_1 from Fig. 4.1 (left) and its corresponding TFG (right).

The second kind of arc, $a \circ\!\!\rightarrow p$, is for *agglomeration equations*. It represents equations of the form $a = p + q + \ldots$, generated when we agglomerate several places into a new one. In this case, we expect that if we can reach a marking with k tokens in a, then we can certainly reach a marking with k_1 tokens in p and k_2 tokens in q, ... such that $k = k_1 + k_2 + \ldots$ (see property Agglomeration in Lemma 4.3). Hence, the marking of p and q can be reconstructed from the marking of a. In this case, places $p, q, \ldots$ are removed. We also say that node a is *inserted*; it does not exist in N_1 but may appear as a new place in N_2 unless a subsequent reduction removes it. We can have more than two places in an agglomeration. Figure 4.3 illustrates an example of such reduction obtained by agglomerating places p_1, p_2 together, in net M_1 of

Fig. 4.1, into a new place a_1. Thus, the TFG in Fig 4.3 (right) depicts the obtained equation $a_1 = p_1 + p_2$ (modeled by "white dot" arcs).

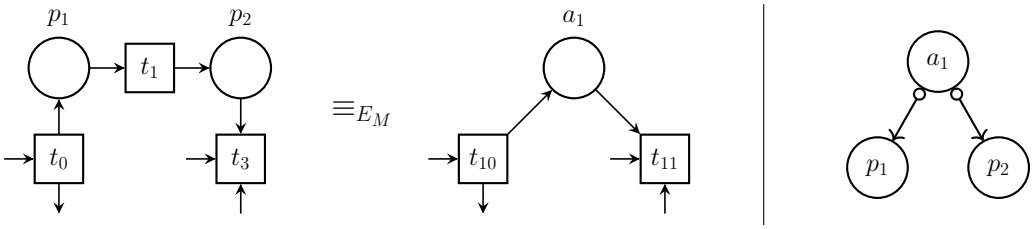


Fig. 4.3 Agglomeration reduction applied on a subpart of the net M_1 from Fig. 4.1 (left) and its corresponding TFG (right).

A TFG can also include nodes for *constants*, used to express invariant statements on the markings of the form $p + q = k$. To this end, we assume that we have a family of disjoint sets $K(n)$ (also disjoint from place and variable names), for each n in $\mathbb{N}$, such that the "valuation" of a node $v \in K(n)$ will always be n. We use K to denote the set of all constants. We may write v^n (instead of just v) for a constant node whose value is n. Note that we can have more than one constant node in $K(n)$ with the same valuation n.

Definition 4.3 (Token Flow Graph)**.** *A Token Flow Graph (TFG) with set of places P is a directed graph $(P, S, R^\bullet, A^\circ)$ such that:*

- *$V \triangleq P \cup S$ is a set of vertices (or nodes) with $S \subset K$ a finite set of constants;*
- *$R^\bullet \in V \times V$ is a set of* redundancy arcs*, $v \to\!\!\!\!\bullet\ v'$;*
- *$A^\circ \in V \times V$ is a set of* agglomeration arcs*, $v \circ\!\!\to v'$, disjoint from $R^\bullet$.*

The primary source of complexity in our approach arises from the need to manage interdependencies between A° and $R^\bullet$ arcs, in situations where redundancies and agglomerations are combined. This is not something that can be easily achieved by looking only at the equations in E and thus motivates the need for a specific data structure.

We define several notations that will be useful in the following. We use the notation $v \to v'$ when we have $(v \to\!\!\!\!\bullet\ v')$ in $R^\bullet$ or $(v \circ\!\!\to v')$ in A°. We say that a node v is a *root* if it is not the target of an arc, and a *leaf* denoted $v \not\to$ when it has no output arc. It is a *$\circ$-leaf* when it has no output arc of the form $(v \circ\!\!\to v')$. A sequence of nodes $(v_1, \ldots, v_n)$ in V^n is a *path* if for all $1 \leqslant i < n$ we have $v_i \to v_{i+1}$. We use the notation $v \to^\star v'$ when there is a path from v to v' in the graph or when $v = v'$. We write $v \circ\!\!\to X$ when X is the largest subset $\{v_1, \ldots, v_k\}$ of V such that $X \neq \emptyset$ and $v \circ\!\!\to v_i \in A^\circ$ for all $i \in 1..k$. And similarly, we write $X \to\!\!\!\!\bullet\ v$ when X is the largest, non-empty set of nodes $\{v_1, \ldots, v_k\}$ such that $v_i \to\!\!\!\!\bullet\ v \in R^\bullet$ for all $i \in 1..k$. Finally, the notation $\downarrow v$ denotes the set of successors of v, that is: $\downarrow v \triangleq \{v' \in V \mid v \to^\star v'\}$. We extend it to a set of variables X with $\downarrow X \triangleq \bigcup_{x \in X} \downarrow x$.

We display an example of Token Flow Graph in Fig. 4.4, which corresponds to reduction equations in our running example, and where "black dot" arcs model edges in $R^\bullet$ and "white dot" arcs model edges in A°. The idea is that each relation $X \rightarrow\!\!\bullet\, v$ or $v \circ\!\!\rightarrow X$ corresponds to one equation $v = \sum_{v_i \in X} v_i$ in E, and that all the equations in E should be reflected in the TFG. As mentioned, we deal with inequalities by adding slack variables, such as with the TFG depicted in Fig. 5.2 corresponding to the polyhedral reduction of Fig. 5.1. We want to avoid situations where the same place is removed more than once or where some place occurs in the TFG but is never mentioned in N_1, N_2 or E. Furthermore, we also have the roots (if we forget about constant nodes) that match the places in N_2, and the $\circ\!\!\rightarrow$-leaves to the places in N_1 (if we forget about slack variables). Finally, to ensure the state space partition, slack variables may only be used on constant nodes (otherwise, one marking of the initial net N_1 may be related to two different markings in the reduced net N_2). To simplify the presentation, we also require at most one slack variable per constant (more does not improve the expressiveness). All these constraints can be expressed using a suitable notion of a well-formed graph built from E.

Definition 4.4 (Well-formed TFG)**.** *Assume the equivalent statement* $(N_1, m_1) \equiv_E (N_2, m_2)$ *such that* $E \triangleq \exists Q.E'$ *is in linear system form. A TFG* $G \triangleq (P, S, R^\bullet, A^\circ)$ *for this equivalence statement is well-formed when the following constraints are met, with* P_1, P_2 *the set of places in* N_1, N_2*:*

- **(T1)** no unused names*:* $(P \cup S) \setminus K = P_1 \cup P_2 \cup Q$*;*
- **(T2)** nodes in S are roots*: if* $v \in S$ *then* v *is a root of* G*;*
- **(T3)** nodes can be removed only once*: it is not possible to have* $v \circ\!\!\rightarrow w$ *and* $v' \rightarrow w$ *with* $v \neq v'$*, or to have both* $v \rightarrow\!\!\bullet\, w$ *and* $v \circ\!\!\rightarrow w$*;*
- **(T4)** G contains all and only the equations in E'*: we have* $v \circ\!\!\rightarrow X$ *or* $X \rightarrow\!\!\bullet\, v$ *if and only if the equation* $v = \sum_{v_i \in X} v_i$ *is in* E' *;*
- **(T5)** G is acyclic*;*
- **(T6)** nodes in G match nets*: roots in* $P \setminus S$ *are exactly the set of places* P_2*; for each o-leaf* v *either* $v \in P_1$ *or there is some constant root* $r \in S$ *such that* $r \circ\!\!\rightarrow v \not\rightarrow$*;*
- **(T7)** at most one slack variable per constant node*: for each constant node* $k \in S$ *there is at most one successor* $v \notin P_1$ *such that* $k \circ\!\!\rightarrow v \not\rightarrow$*.*

Given a relation $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$, the well-formedness conditions are enough to ensure the unicity of a TFG (up-to the choice of constant nodes) when we set each equation to be either in A° or in $R^\bullet$. In this case, we denote the graph $[\![E']\!]$. In practice, we use the tool Reduce to generate the E-equivalence from the initial net (N_1, m_1). This tool outputs a sequence of equations suitable to build a TFG, and for each equation, it adds a tag indicating if it is a Redundancy or an Agglomeration. We display in Fig. 4.4 the equations generated by Reduce for the net M_1 given in Fig. 4.1, where annotations `R` and `A` indicate if an equation is a redundancy or an agglomeration.

```
# R |- p5 = p4
# A |- a1 = p2 + p1
# A |- a2 = p4 + p3
# R |- a1 = a2
```

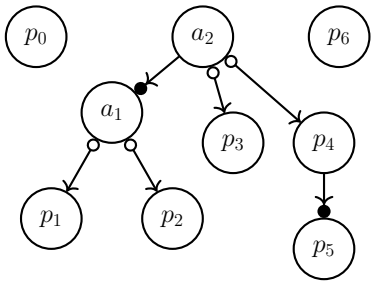


Fig. 4.4 Equations generated from net M_1, in Fig. 4.1, and their associated TFG.

The constraints (T1)–(T7) are not artificial or arbitrary. In practice, we compute E-equivalences using multiple steps of structural reductions, and a TFG exactly records the constraints and information generated during these reductions. In some sense, equations E abstract a relation between the semantics of two nets, whereas a TFG records the structure of reductions between places.

4.3.1 Example of a Non-TFGizable Polyhedral Reduction

Even if every polyhedral reduction computed by our tool Reduce provides a system of equations, with only top-level existential quantifiers, there are some reduction rules that are *non-TFGizable* (i.e., no well-formed TFG can match E). An example is the [GENERAL LOOP AGG] rule in Fig. 4.5. We obtain $E \triangleq (a_1 = p_0 + p_1) \wedge (a_2 = p_0 + p_2)$, and place p_0 is removed twice in two different agglomerations, a_1 and a_2. Hence, condition (T3) cannot be satisfied.

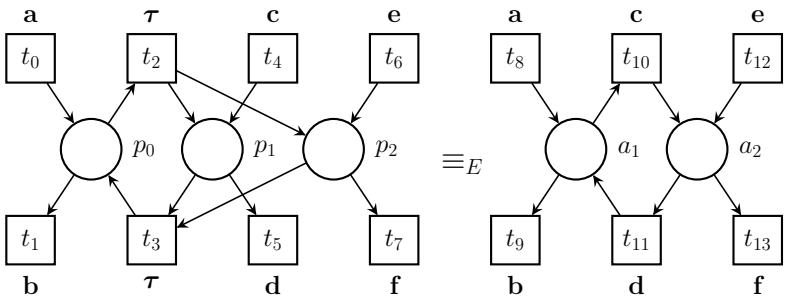


Fig. 4.5 Example of a non-TFGizable reduction rule [GENERAL LOOP AGG] with $E \triangleq (a_1 = p_0 + p_1) \wedge (a_2 = p_0 + p_2)$.

This particular reduction rule can be applied to reduce the SmallOperatingSystem net depicted in Fig. 3.10 by obtaining the additional reduction predicate $(a_1 = \mathit{CPUUnit} + \mathit{ExecutingTask}) \wedge (a_2 = \mathit{TaskSuspended} + \mathit{ExecutingTask})$.

We give in Sect. 4.6.2 a complete study on the few opportunities we lost to reduce a net from the MCC benchmark due to our well-formedness constraint.

4.3.2 Example of a TFG Not Generated by Structural Reductions

It is important to mention that not only "structural reduction rules" can lead to TFGizable polyhedral reduction. For example, the reduced version of the SmallOperatingSystem net in Fig. 3.10 (right) is part of the PR-R class since it is a live marked graph, and its generated equations form a well-formed TFG:

$$\begin{cases} 4096 \circ\!\!\rightarrow \{a_4, \mathit{LoadingMem}\} \\ 8192 \circ\!\!\rightarrow \{\mathit{FreeMemSegment}, \mathit{LoadingMem}, a_5\} \end{cases}$$

4.4 Semantics

By construction, a strong connection exists between "systems of reduction equations", $\exists Q.E'$, and their associated graph, $[\![E']\!]$. We show that a similar relation exists between solutions of $\exists Q.E'$ and the "valuations" of the graph (that we then call *configurations*).

A *configuration* c of a TFG $(P, S, R^\bullet, A^\circ)$ is a partial function from V to $\mathbb{N}$. We use the notation $c(v) = \bot$ when c is not defined on v, and we always assume that $c(v) = n$ when v is a constant node in $K(n)$.

Configuration c is *total* when $c(v)$ is defined for all nodes v in V; otherwise it is said *partial*. We use the notation $c_{|N}$ for the configuration obtained from c by restricting its domain to the set of places in the net N. We remark that when c is defined over all places of N, then $c_{|N}$ can be viewed as a marking. As for markings, we say that two configurations c and c' are *compatible*, denoted $c \equiv c'$, if they have the same value on the nodes where they are both defined: $c(p) = c'(p)$ when $c(v) \neq \bot$ and $c'(v) \neq \bot$. (Same holds when comparing a configuration to a marking.) We also use $\underline{c}$ to represent the system $v_1 = c(v_1) \wedge \cdots \wedge v_k = c(v_k)$ where the $(v_i)_{i\in 1..k}$ are the nodes such that $c(v_i) \neq \bot$. We say that a configuration c is *well-defined* when the valuation of the nodes agrees with the equations of $[\![E']\!]$.

Definition 4.5 (Well-Defined Configuration)**.** *Configuration c is well-defined when, for all nodes p, the following two conditions hold:*

(CBot) *if $v \rightarrow w$ then $c(v) = \bot$ if and only if $c(w) = \bot$;*
(CEq) *if $c(v) \neq \bot$ and $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ then $c(v) = \sum_{v_i \in X} c(v_i)$.*

We prove that the well-defined configurations of a TFG $[\![E']\!]$ are partial solutions of E' and reciprocally. Therefore, because all the variables in E' are nodes in the TFG (condition (T1)), we have an equivalence between solutions of E' and total, well-defined configurations of $[\![E']\!]$.

Lemma 4.2 (Well-Defined Configurations are Solutions)**.** *Assume that $[\![E']\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$. If c is a well-defined configuration of $[\![E']\!]$, then $E' \wedge \underline{c}$ is satisfiable. Conversely, if c is a total configuration of $[\![E']\!]$ such that $E' \wedge \underline{c}$ is satisfiable then c is also well-defined.*

Proof. We prove each property separately.

Assume that c is a well-defined configuration of $[\![E']\!]$. Since E' is a system of reduction equations, it is a sequence of equalities $\phi_1, \ldots, \phi_k$ where each equation ϕ_i has the form $x_i = y_1 + \cdots + y_n$. Also, since $[\![E']\!]$ is well-formed we have $X_i \rightarrow\!\!\bullet\, x_i$ or $x_i \circ\!\!\rightarrow X_i$ (only one case is possible) with $X_i \triangleq \{y_1, \ldots, y_n\}$ for all indices $i \in 1..k$. We define I as the subset of indices in $1..k$ such that $c(x_i)$ is defined. By condition (CBot), we have $c(x_i) \neq \bot$ if and only if $c(v) \neq \bot$ for all $v \in X_i$. Therefore, if $c(x_i) \neq \bot$, we have by condition (CEq) that $\phi_i \wedge \underline{c}$ is satisfiable. Moreover, the values of all the variables in ϕ_i are determined by $\underline{c}$ (these variables have the same value in every solution). As a consequence, the system combining $\underline{c}$ and the $(\phi_i)_{i \in I}$ has a unique solution. On the opposite, if $c(x_i) = \bot$ then no variables in ϕ_i are defined by $\underline{c}$. Nonetheless, we know that system E' is satisfiable. Indeed, by property of E-equivalence, we know that $E \wedge \underline{m_1}$ has solutions, which is also the case with only $E \triangleq \exists Q.E'$, hence E' also admits solutions. Therefore, the system combining the equations in $(\phi_i)_{i \notin I}$ has solutions. Since this system shares no variables with the equations in $(\phi_i)_{i \in I}$, we have $E' \wedge \underline{c}$ satisfiable.

For the second case, we assume that c is total and that $E' \wedge \underline{c}$ is satisfiable. Since c is total, condition (CBot) is true ($c(v) \neq \bot$ for all nodes in $[\![E']\!]$). Assume that we have $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$. For condition (CEq), we rely on the fact that $[\![E']\!]$ is well-formed (T4). Indeed, for all equations in E', we have a corresponding relation $X \rightarrow\!\!\bullet\, v$ or $v \circ\!\!\rightarrow X$. Hence, $E' \wedge \underline{c}$ satisfiable implies that $c(v) = \sum_{w \in X} c(w)$. □

We can prove several properties related to how the structure of a TFG constrains possible values in well-defined configurations. These results can be thought of as the equivalent of a "token game", which explains how tokens can propagate along the arcs of a TFG. This is useful in the context of Chapter 6 since we can assess that two nodes are concurrent when we can mark them in the same configuration. (A similar result holds for finding pairs of nonconcurrent nodes.)

Our first result shows that we can always propagate tokens from a node to its children, meaning that if a node has a token, we can find one in its successors (possibly in a different well-defined configuration). Property (Backward) states a dual result; if a child node is marked, then one of its parents must be marked.

Lemma 4.3 (Token Propagation)**.** *Assume that $[\![E']\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$ and that c is a well-defined configuration of $[\![E']\!]$.*

(Agglomeration) *If $p \circ\!\!\to \{q_1, \ldots, q_k\}$ and $c(p) \neq \bot$ then for every sequence $(l_i)_{i \in 1..k}$ of $\mathbb{N}^k$ such that $c(p) = \sum_{i \in 1..k} l_i$, we can find a well-defined configuration c' such that $c'(p) = c(p)$, and $c'(q_i) = l_i$ for all $i \in 1..k$, and $c'(v) = c(v)$ for every node v not in $\downarrow p$.*

(Forward) *If p, q are nodes such that $c(p) \neq \bot$ and $p \to^\star q$ then we can find a well-defined configuration c' such that $c'(q) \geqslant c'(p) = c(p)$ and $c'(v) = c(v)$ for every node v not in $\downarrow p$.*

(Backward) *If $c(p) > 0$ then there is a root v such that $v \to^\star p$ and $c(v) > 0$.*

Proof. We prove each property separately. Without loss of generality, we assume that an (arbitrary) total ordering on nodes exists.

Agglomeration: let p be a node such that $p \circ\!\!\to X$, with $X \triangleq \{q_1, \ldots, q_k\}$, and let $(l_1, \ldots, l_k) \in \mathbb{N}^k$ be a sequence such that $c(p) = \sum_{i \in 1..k} l_i$. We define configuration c' as a recursive function. The base cases are defined by: $c'(p) = c(p)$, for all $i \in 1..k$, $c'(q_i) = l_i$, and $c'(v) = c(v)$ for all the nodes v such that $v \notin \downarrow p$. The recursive cases concern only the nodes that are successors of nodes in X. Let w be such a node. It cannot be a root (since w is a successor of a node in X); hence it has at least one parent x. We consider two cases:

- Either $x \to\!\!\bullet w$ holds: then, let Y be the set of parents of w (as expected, $x \in Y$). Property (T3) of Definition 4.4 implies that $Y \to\!\!\bullet\, w$ holds, and we define $c'(w) = \sum_{y \in Y} c'(y)$.
- Or $x \circ\!\!\to w$ holds: then x is the only parent of w by property (T3). Let Y be the set of agglomeration children of x: we have $x \circ\!\!\to Y$, and $w \in Y$. We define $c'(w) = c'(x)$ if w is the smallest node of Y (according to the total ordering on nodes), or $c'(w) = 0$ otherwise. This entails $c'(x) = \sum_{y \in Y} c'(y)$ (where all terms are defined as zero except one defined as $c'(x)$).

Note that the recursion always involves the parents of a given node and is, therefore, well-founded since a TFG is a DAG. It is immediate to check that c' is well-defined: by construction, (CEq) is satisfied on all nodes where c' is defined.

Forward: take a well-defined configuration c of $[\![E']\!]$ and assume that we have two nodes p, q such that $c(p) \neq \bot$ and $p \to^\star q$. The proof is by induction on the path length from p to q. The initial case is when $p = q$, which is trivial. Otherwise, assume that $p \to r \to^\star q$. It is enough to find a well-defined configuration c' such that $c'(r) \geqslant c'(p) = c(p)$. Since the nodes not in $\downarrow p$ are not in the paths from p to q, we can ensure $c'(v) = c(v)$ for any node v not in $\downarrow p$. The proof proceeds by case analysis on $p \to r$:

- Either $X \rightarrow\!\!\bullet\, r$ with $p \in X$. Then by (CEq) we have $c(r) = c(p) + \sum_{v \in X, v \neq p} c(v) \geqslant c(p)$, and we can choose $c' = c$.
- Or we have $p \circ\!\!\rightarrow X$ with $r \in X$. By (Agglomeration) we can find a well-defined configuration c' such that $c'(r) = c'(p) = c(p)$ (and also $c'(v) = 0$ for all $v \in X \setminus \{r\}$).

Backward: let c be a well-defined configuration of $[\![E']\!]$ such that $c(p) > 0$. The proof is by reverse structural induction on the DAG. If p is a root, the result is immediate; otherwise, p has at least one parent q such that $q \to p$. As previously, we proceed by case analysis.

- Either $X \rightarrow\!\!\bullet\, p$ with $q \in X$. By (CEq), we have $c(p) = \sum_{v \in X} c(v) > 0$. Hence, there must be at least one node q' in X such that $c(q') > 0$, and we conclude by induction hypothesis on q', which is a parent of p.
- Or we have $q \circ\!\!\rightarrow X$ with $p \in X$. By (CEq), we have $c(q) = \sum_{v \in X} c(v) \geqslant c(p) > 0$, and we conclude by induction hypothesis on q, which is again a parent of p.

□

Until this point, none of our results rely on the properties of E-equivalence. We now prove an equivalence between the reachable markings—of N_1 and N_2—and configurations of $[\![E']\!]$. More precisely, we prove (Theorem 4.4) that every reachable marking in N_1 or N_2 can be extended into a well-defined configuration of $[\![E']\!]$. This entails that we can reconstruct all the reachable markings of N_1 by looking at well-defined configurations obtained from the reachable markings of N_2. Our algorithm for computing the concurrency relation in Chapter 6 will be a bit smarter since we do not need to enumerate exhaustively all the markings of N_2. Instead, we only need to know which roots can be marked together.

Theorem 4.4 (Configuration Reachability)**.** *Assume that $[\![E']\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$. If m is a marking in $R(N_1, m_1)$ or $R(N_2, m_2)$ then there exists a total, well-defined configuration c of $[\![E']\!]$ such that $c \equiv m$. Conversely, given a total, well-defined configuration c of $[\![E']\!]$, marking $c_{|N_1}$ is reachable in (N_1, m_1) if and only if $c_{|N_2}$ is reachable in (N_2, m_2).*

Proof. We prove each point separately.

First, we take a marking m in $R(N_1, m_1)$ (the case m in $R(N_2, m_2)$ is similar). By property (A2a) of Definition 4.1, $E \wedge \underline{m}$ is satisfiable. Hence, it admits a non-negative integer solution c, meaning a valuation for all the variables and places in Q (existential variables), N_1 and N_2 such that $E' \wedge \underline{c}$ is satisfiable and $c(p) = m(p)$ if $p \in N_1$. We may freely extend c to include constants in K (whose values are fixed), thus according to condition (T1) of Definition 4.4, this solution c is defined over all the nodes of $[\![E']\!]$. It is well-defined by virtue of Lemma 4.2.

For the converse property, we assume that c is a total, well-defined configuration of $[\![E']\!]$ and that $c_{|N_1}$ is in $R(N_1, m_1)$ (the case $c_{|N_2}$ in $R(N_2, m_2)$ is again similar). Since c is a

well-defined configuration, from Lemma 4.2 we have $E \wedge \underline{c}$ satisfiable. Therefore, we have $E \wedge \underline{c_{|N_1}} \wedge \underline{c_{|N_2}}$ satisfiable. By condition (A2b) of Definition 4.1, we have $c_{|N_2}$ in $R(N_2, m_2)$, as needed. □

The second result of this theorem justifies the following definition.

Definition 4.6 (Reachable Configuration)**.** *Configuration c is reachable for an equivalence statement $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$ if c is total, well-defined and $c_{|N_1} \in R(N_1, m_1)$ (resp. $c_{|N_2} \in R(N_2, m_2)$).*

The previous fundamental results demonstrate the possibilities of TFGs to reason about the state space of the initial net from the one of the reduced net and vice versa.

4.5 Marking Reachability

We illustrate the benefit of Token Flow Graphs by describing a simple model checking algorithm. The goal is to decide if a marking, say m'_1, is reachable in the initial net (N_1, m_1) by checking a reachability property on the smaller net (N_2, m_2). We start by proving some auxiliary results.

Lemma 4.5 (Unicity of Marking Reduction)**.** *Assume that $[\![E']\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$. Given a marking m'_1 of N_1 there exists at most one total, well-defined configuration c such that $c \equiv m'_1$.*

Proof. Let c_1 and c_2 be two total, well-defined configurations such that $c_1 \equiv m'_1$ and $c_2 \equiv m'_1$. Let X be the set of nodes x such that $c_1(x) \neq c_2(x)$. By contradiction, we suppose that X is not empty (that is, we suppose that $c_1 \neq c_2$). For each node x in X, we know that x does not belong to P_1, since c_1 and c_2 agree on m'_1. Consequently, by virtue of property (T6) of Definition 4.4, for each x in X either it admits an output $x \circ\!\!\rightarrow y$, or x is a slack variable (there is some constant root $r \in K$ such that $r \circ\!\!\rightarrow x \not\rightarrow$). We examine the two cases in turn.

- We consider an element x_0 of X such that $x_0 \circ\!\!\rightarrow Y_0$ holds with Y_0 disjoint from X (such an element x_0 necessarily exists if some there is some $x \in X$ and y such that $x \circ\!\!\rightarrow y$; otherwise, X would contain a cycle of $\circ$-arcs, which is forbidden by the acyclic property (T5) of the well-formed TFG). Then, since c_1 is well-defined, we know that $c_1(x_0) = \sum_{y \in Y_0} c_1(y)$ by property (CEq). However, we have $c_1(y) = c_2(y)$ for all $y \in Y_0$ since Y_0 is disjoint from X. Hence, $c_1(x_0) = \sum_{y \in Y_0} c_2(y) = c_2(x_0)$, which contradicts $x_0 \in X$.
- We consider an element x_0 of X such that $x_0 \not\rightarrow$ and a constant root r such that $r \circ\!\!\rightarrow Y$ with $x_0 \in Y$. By condition (T7), for all nodes v in Y we have either $v \in P_1$ (then not in X), or $v \rightarrow v'$ for some node v'. Our previous case demonstration

ensures that $(Y \setminus \{x_0\}) \cap X = \emptyset$. We have $c_1(r) = c_2(r)$, then by property (CEq) $\Sigma_{v \in Y} c_1(v) = \Sigma_{v \in Y} c_2(v)$, which contradicts $x_0 \in X$, since $c_1(v) = c_2(v)$ for all nodes in $Y \setminus \{x\}$.

In conclusion, X must be empty, that is, $c_1 = c_2$. There can be, at most, one well-defined configuration. □

Then, as a corollary of this lemma and of Theorem 4.4, we get:

Theorem 4.6 (Reachability Decision)**.** *Assume that $[\![E']\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$. Deciding if a marking m'_1 is reachable in $R(N_1, m_1)$ amounts to constructing a total, well-defined configuration c such that $c \equiv m'_1$ and then checking if $c_{|N_2}$ is reachable in $R(N_2, m_2)$.*

Proof. Let m'_1 a marking of N_1. Assume that there exists a total, well-defined configuration c of $[\![E']\!]$ such that $c \equiv m'_1$. By Lemma 4.5 we know that such configuration c is unique. Applying Theorem 4.4 ensures that m_1 is reachable in $R(N_1, m_1)$ if and only if $c_{|N_2}$ is reachable in $R(N_2, m_2)$.

Otherwise, if no such configuration c exists, we can immediately conclude that m'_1 is not reachable by Theorem 4.4. □

Hence, given an equivalence $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$ and the associated TFG $[\![E']\!]$, we first extend our marking of interest m'_1 into a total well-defined configuration c, as done by Algorithm 4.1, next. (Lemma 4.5 ensures that if such configuration exists, then it is unique.) As stated in Theorem 4.6, if c restricted to N_2 is a marking reachable in (N_2, m_2), then m'_1 is reachable in (N_1, m_1). Otherwise, $\neg \underline{m'_1}$ is an invariant on $R(N_1, m_1)$.

4.5.1 Examples of Marking Projection

We illustrate this algorithm by taking two concrete examples on the marked net M_1 given in Fig. 4.1. Assume we want to decide if marking $m'_1 \triangleq (p_0 = 0\,,\, p_1 = 2\,,\, p_2 = 0\,,\, p_3 = 1\,,\, p_4 = 1\,,\, p_5 = 1\,,\, p_6 = 0)$ is reachable in (N_1, m_1), for m_1 as depicted in Fig. 4.1. This marking can be extended into a total, well-defined configuration c, with $c(a_1) = c(a_2) = 2$. And so, deciding of the reachability of marking m'_1 in (N_1, m_1) is equivalent to deciding whether marking $m'_2 \triangleq (p_0 = 0\,,\, a_2 = 2\,,\, p_6 = 0)$ is reachable in (N_2, m'_2) (which it is not). Observe that m'_1 would be reachable if the initial marking m_1 was $(p_0 = 2, p_6 = 1)$ and the other places empty.

Conversely, assume that our marking of interest is m''_1 such that $m''_1(p_4) = 2$ and $m''_1(p_1) = m''_1(p_2) = 0$. It is not possible to extend this marking into a well-defined configuration c, since $c(a_1) = m''_1(p_1) + m''_1(p_2) = 0$ and $c(a_1) = c(a_2) > m''_1(p_4)$. In this case, we directly obtain that m''_1 is not reachable in (N_1, m_1) for every initial marking m_1.

4.5.2 Description of the Algorithm

The Reachable function (Algorithm 4.1) is a direct implementation of Theorem 4.6: it builds a total configuration c, then checks that it is well-defined (we omit the function Well-Defined, which is obvious), and finally finds out if $c_{|N_2}$ is reachable in (N_2, m_2). This algorithm relies on the recursive procedure BottomUp (Algorithm 4.2) that extends the marking of interest m'_1 into a total, well-defined configuration if there is one.

Algorithm 4.1 Reachable$(m'_1, [\![E']\!], (N_2, m_2))$

In: m'_1 : a marking of N_1,
(N_2, m_2): reduced net such that $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$ holds,
$[\![E']\!]$: well-formed TFG for the E-equivalence above.
Out: a boolean indicating if $m'_1 \in R(N_1, m_1)$.

1: *;; c is a configuration of $[\![E']\!]$.*
2: $c \leftarrow \bot$
3:
4: **for all** $p \in P_1$ **do** $c[p] \leftarrow m_1[p]$
5: **for all** $v \in K(n)$ for some n **do** $c[v] \leftarrow n$
6:
7: *;; $[\![E']\!]$ is (V, R, A), as in Definition 4.3.*
8: **for all** $v \in V$ **do** BottomUp$(c, v, [\![E']\!])$
9:
10: **return** Well-Defined$(c) \wedge c_{|N_2} \in R(N_2, m_2)$

Algorithm 4.2 BottomUp$(c, v, [\![E']\!])$

In: $[\![E']\!]$: the TFG structure,
v : a node in $[\![E']\!]$.
In out: c : a partial configuration of $[\![E']\!]$.
Post: c is defined for all nodes of $\downarrow v$.

1: **for all** v' **such that** $v \to v'$ **do**
2: BottomUp$(c, v', [\![E']\!])$
3:
4: **if** $v \in K(n)$ for some n and $\exists v' \notin P_1$ such that $v \circ\!\!\to v' \not\twoheadrightarrow$ **then**
5: $c[v'] \leftarrow c[v] - \sum_{w \in X \setminus \{v'\} | v \circ\!\!\to X} c[v']$
6:
7: **if** $v \circ\!\!\to X$ **then** $c[v] \leftarrow \sum_{v' \in X} c[v']$

We note that the second algorithm, which is recursive, always terminates since it simply follows the TFG structure. We still have to prove that Algorithm 4.1 always returns the correct answer.

Theorem 4.7 (Algorithm 4.1 is Sound and Complete)**.** *The verdict returned by* Reachable$(m_1', [\![E']\!], (N_2, m_2))$ *is equivalent to deciding wether* $m_1' \in R(N_1, m_1)$.

Proof. We consider two cases.

Case C1: the algorithm returns false because c is not well-defined (line 10). In this case, we show that no well-defined configuration c extending m_1' exists, and thus m_1' is not reachable by Theorem 4.4.

Case C2: the algorithm returns the value of $c_{|N_2} \in R(N_2, m_2)$. Thanks to Theorem 4.6, it suffices to show that c is total, well-defined, and extends m_1'.

We start with case C2, which states the algorithm's soundness. Let us show that c is total: for every node v, if v is a constant, or if it belongs to P_1, it is set by lines 4 and 5 of Algorithm 4.1. Otherwise, v is a slack variable or not a $\circ$-leaf (by property (T6) of Definition 4.4), hence it is set by lines 5 or 7 of Algorithm 4.2, which is invoked on every node of the TFG (line 8, Algorithm 4.1). Additionally, c is well-defined because it passed the test line 10. It also extends m_1' by consequence of line 4 (those values are not overwritten later in Algorithm 4.2 because of property (T6)).

Case C1 states the completeness of the algorithm. By contraposition, we suppose that there exists a total well-defined configuration c' extending m_1', and show by induction on the recursive calls to BottomUp that the algorithm builds a configuration c equal to c'. More precisely, we show that an invocation of BottomUp$(c, v, [\![E']\!])$ returns a configuration c that coincides with c' on all nodes of $\downarrow v$. The initial configuration c built by the algorithm extends m_1' and sets the constants (lines 4 and 5 of Algorithm 4.1). Then, for any invocation of BottomUp$(c, v, [\![E']\!])$, for every node w in $\downarrow v$, if w is a node in P_1, then $c(v) = c'(v)$ holds immediately. Otherwise, w is a slack variable or not a $\circ$-leaf. If $w \neq v$, then w is in $\downarrow u$ for some child u of v, and $c(v) = c'(v)$ holds by induction hypothesis on the recursive call to BottomUp$(c, u, [\![E']\!])$ that occurred in line 2. In the special case where w is a slack variable, then v is a constant root, and we have $c(v) = c(v')$, still by induction hypothesis on the call line 5. If $w = v$, then $c(v)$ is set by line 7, that is $c(v) = \sum_{v' \in X} c(v')$. By the induction hypothesis, we have $c(v') = c'(v')$ for all v' children of v (the recursive call occurred also in line 2). Hence, $c(v) = \sum_{v' \in X} c'(v') = c'(v)$ by property (Ceq) since c' is well-defined. Consequently, $c(w) = c'(w)$ for all w in $\downarrow v$. As a result, $c = c'$, since BottomUp is invoked on all nodes of the TFG. □

4.5.3 State Space Partition

We can use the previous results to derive an interesting result about the state space of equivalent Petri nets where the associated TFG is well-formed. Indeed, we can prove that, in this case, we can build a partition of the reachable markings of (N_1, m_1) that is in bijection with the reachable markings of (N_2, m_2).

Given a marking m'_2 of the reduced net N_2, we define $\mathrm{Inv}_{[\![E']\!]}(m'_2)$ as the set of markings of the initial net N_1 compatible with m'_2.

$$\mathrm{Inv}_{[\![E']\!]}(m'_2) \triangleq \{c_{|N_1} \mid c \text{ total, well-defined configuration of } [\![E']\!] \text{ such that } c \equiv m'_2\} \tag{4.1}$$

Theorem 4.8 (State Space Partition)**.** *Assume that $[\![E']\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$. The family of sets $S \triangleq \{\mathrm{Inv}_{[\![E']\!]}(m'_2) \mid m'_2 \in R(N_2, m_2)\}$ is a partition of $R(N_1, m_1)$.*

Proof. The set S is a partition as a consequence of the following points:

No empty set in S. For any marking m'_2 in $R(N_2, m_2)$, by Theorem 4.4, there exists a total, well-defined configuration c such that $c \equiv m$. Thus, $\mathrm{Inv}_{[\![E']\!]}(m'_2)$ is not empty. This implies $\emptyset \notin S$.

The union $\cup_{A\in S} A$ is equal to $R(N_1, m_1)$. We prove the inclusion in both ways separately.

- Take a marking m'_1 in $R(N_1, m_1)$. From Theorem 4.4 there exists a total, well-defined configuration c such that $c \equiv m$ and $c_{|N_2} \in R(N_2, m_2)$. Hence, $R(N_1, m_1) \subseteq \cup_{A\in S} A$.
- Take a set A in S and a marking m'_1 from it. By construction, there is some marking $m'_2 \in R(N_2, m_2)$ and total, well-defined configuration c such that $c \equiv m'_1$ and $c \equiv m'_2$. From Theorem 4.4, since m'_2 is reachable in (N_2, m_2) we have m'_1 reachable in (N_1, m_1). Hence, $\cup_{A\in S} A \subseteq R(N_1, m_1)$.

Pairwise disjoint. Take two different markings m'_2 and m''_2 in $R(N_2, m_2)$. From Lemma 4.5 we have $\mathrm{Inv}_{[\![E']\!]}(m'_2) \cap \mathrm{Inv}_{[\![E']\!]}(m''_2) = \emptyset$ since every marking of (N_1, m_1) can be extended into at most one possible configuration c. □

As a corollary, when a marked net (N_1, m_1) can be partially reduced, we know how to partition its state space into a union of disjoint *convex sets*, meaning sets of markings defined as solutions to a system of linear equations.

This result is fundamental for the first application of polyhedral reduction, model counting [BLD18; BLD19], that is, counting the number of reachable markings of a net without having to enumerate them first. Computing the cardinality of the reachability set has several applications. For instance, it is a straightforward way to assess the correctness of tools—all tools should obviously find the same results on the same models.

4.6 Experimental Results

We have implemented the approach in a tool, called Kong, that performs the "inverse transforms" described in Sect. 4.5. We use the database of models provided by the Model Checking Contest (MCC) to experiment with our approach. A complete description of the toolchain can be found in Chapter 8.

4.6.1 Toolchain Description

Figure 4.6 depicts the toolchain used for checking if a given marking, m'_1, is reachable in an input net (N_1, m_1). In this case, marking m'_1 is defined in an input file using a simple textual format. The tool Kong retrieves the reduction system, E, computed with Reduce and uses it to project m'_1 into a marking m'_2, if possible. If the projection returns an error, we know that m'_1 cannot be reachable. Otherwise, we call an auxiliary tool, in this case, Sift, to explore the state space of (N_2, m_2) and try to find marking m'_2.

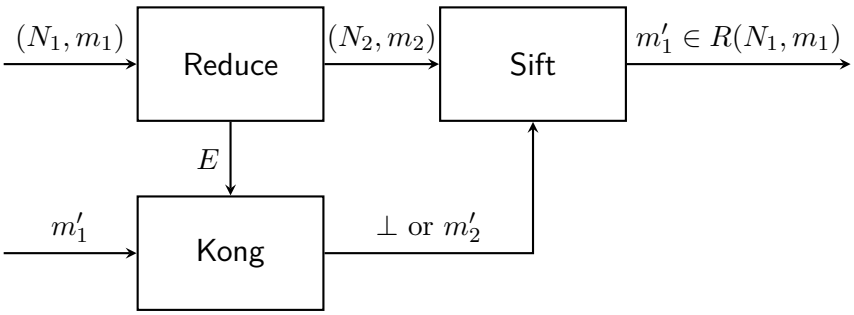


Fig. 4.6 Toolchain of the marking reachability decision procedure.

4.6.2 Distribution of Reduction Ratios for TFGs

As in Chapter 3, we computed the reduction ratio (r), obtained using Reduce, on all the instances (see Fig. 4.7). Here, we consider two values for the reduction ratio: one for reductions leading to a well-formed TFG (in light orange), the other for the best possible reduction with Reduce (in dark blue), used for instance in the SMPT model checker (Chapter 3).

A ratio of 100% ($r = 1$) means that the net is *fully reduced*; the residual net has no places, and in this case all the roots in its TFG are constants. We see that there is a surprisingly high number of models whose size is more than halved with our approach (about 27% of the instances have a ratio $r \geqslant 0.5$), with approximately 40% of the instances that can be reduced by a ratio of 30% or more.

We also observe that we lose few opportunities to reduce a net due to our well-formedness constraint. We mostly lose the ability to simplify some instances of "partial" marking graphs that could be reduced using inhibitor arcs or weights on the arcs, and opportunities to apply the [GENERAL LOOP AGG] rule (see Fig. 4.5).

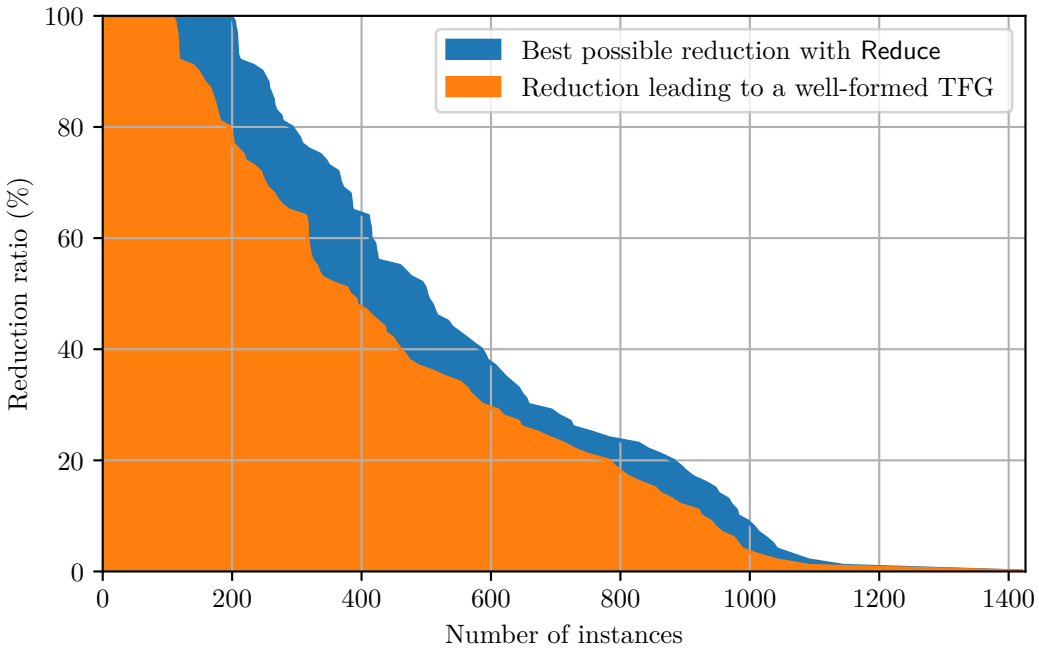


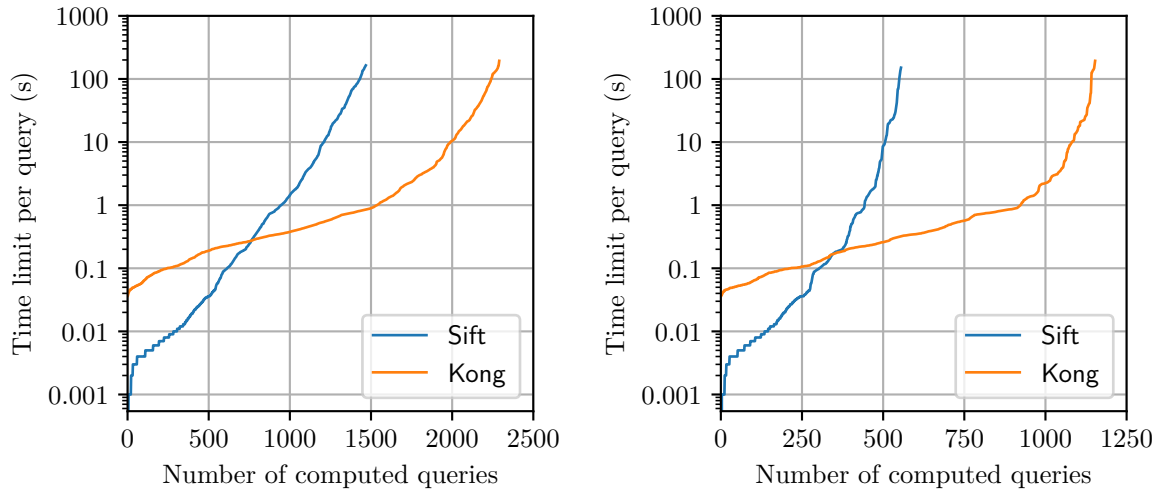
Fig. 4.7 Distribution of reduction ratios in the MCC.

4.6.3 Impact on the Marking Reachability Problem

We evaluated the performance of Kong for the marking reachability problem using a selection of 1 092 Petri nets taken from instances with a reduction ratio greater than 1% (using TFGizable reductions). For each instance, we generated 4 queries that are markings found using a "random walk" on the state space of the net (for this, we used the tool Walk that is part of the Tina toolbox [BRV04; LAA23]). We ran Kong and Sift on each query with a time limit of 180 s.

We display our results in the charts of Fig. 4.8, which compare the minimal time limit per query to compute a given number of queries, with and without the use of reductions. (Note that we use a logarithmic scale for the time value). We consider two different samples of instances: first, only the instances with a high reduction ratio (in the interval $[0.5, 1]$), then the complete set of instances ($r \geqslant 1$).

We observe a clear advantage when we use reductions. For instance, with instances that have a reduction ratio in the interval $[0.5, 1]$, and with a time limit of 180 s, we double the number of computed queries (from 556 with Sift alone, versus 1 154 with Kong). On the opposite, the slight advantage of Sift alone, when the running time is below 0.1 s, can be explained by the fact that we integrate the running time of Reduce to the one of Kong.



(a) $r \in [0.01, 1]$ (b) $r \in [0.5, 1]$

Fig. 4.8 Number of computed reachability queries given the query time limit for (a) all instances, (b) instances with $r \in [0.5, 1]$.

4.7 Discussion

In this chapter, we proposed a new data structure to transpose the computation of reachability problems from an initial "high-dimensionality" domain (the set of places in the initial net) into a smaller one (the set of places in the reduced net). Token Flow Graphs (TFGs) precisely capture the structure of our reduction equations.

We showed how to use the TFGs to accelerate the marking reachability problem. We use TFGs to prove a strong property for the marking reachability problem (see Sect. 4.5), namely that, given a target marking m'_1 for N_1, we can effectively compute a marking m'_2 of N_2 such that m'_1 is reachable in N_1 if and only if m'_2 is reachable in N_2. This can be more efficient than our previous method since the E-transform formula $F_2(\boldsymbol{p_2}) \triangleq \exists \boldsymbol{p_1} \,.\, \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge F_1(\boldsymbol{p_1})$ can be quite complex in practice, even though the property for marking reachability is a simple conjunction of equality constraints. For instance, we performed our experiments using only a bare, explicit-state model checker, that cannot deal with quantifiers.

This application of polyhedral reductions shows that TFGs are an effective method of exploiting reductions. It also bears witness to the versatility of our approach. We propose extending this approach in the following chapters. First, in Chapter 5 for projecting the E-transform formula F_2 into a quantifier-free formula with support on N_2. Second, in Chapter 6 for computing the concurrency relation of a net.

This work has been published in:

- N. Amat, S. Dal Zilio, and D. Le Botlan. "Accelerating the Computation of Dead and Concurrent Places Using Reductions". In: *Model Checking Software (SPIN)*. vol. 12864. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-84629-9_3
- N. Amat, S. Dal Zilio, and D. Le Botlan. "Leveraging polyhedral reductions for solving Petri net reachability problems". In: *International Journal on Software Tools for Technology Transfer* 25.1 (2023), pp. 95–114. DOI: 10.1007/s10009-022-00694-8

The tool related to this chapter is:

- Kong https://github.com/nicolasAmat/Kong

Chapter 5

Project and Conquer

Fast Quantifier Elimination for Checking Reachability

> The ability to simplify means to eliminate the unnecessary so that the necessary may speak.
>
> Hans Hofmann

In this chapter, we propose a method for checking generalized reachability properties on Petri nets that takes advantage of polyhedral reductions, but this time, that can be used transparently as a preprocessing step of existing model checkers. The approach is based on a new procedure that can project a reachability property, about an initial Petri net, into an equivalent formula that only refers to the reduced version of this net.

Our projection is defined as a quantifier elimination procedure for Presburger arithmetic tailored to the specific kind of constraints we handle in Token Flow Graphs (TFGs).

It has linear complexity, is guaranteed to return a sound property, and uses a simple condition to detect when the result is exact. The procedure is implemented in a tool called Octant; experimental results show that our approach works well.

5.1 Introduction

This chapter is the continuation of Chapter 3 on applying polyhedral reduction with SMT-based methods. Nevertheless, here we rely on our specific data structure, i.e., Tokens Flow Graphs described in Chapter 4.

Context. The approach we develop in this chapter relies on the notion of *polyhedral reduction* from Chapter 3, which describes a linear dependence relation, E, between the reachable markings

of a net and those of its reduced version. This abstraction denoted $(N_1, m_1) \equiv_E (N_2, m_2)$, after that, preserves enough information in E so that we can rebuild the reachable markings of N_1 knowing only those of N_2. An interesting application of this relation is the following *reachability conservation theorem* (Theorem 3.10): assume that we have $(N_1, m_1) \equiv_E (N_2, m_2)$, then property F_1 is reachable in N_1 if and only if $F_2(\boldsymbol{p_2}) \triangleq \exists \boldsymbol{p_1} \,.\, \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge F_1(\boldsymbol{p_1})$ is reachable in N_2.

We also presented a specific data structure, called Token Flow Graph, that captures the particular structure of constraints occurring in E, and that permits to reason on the reachable markings of N_1 by playing a "token game" on the nodes of the TFG.

Challenge. Nevertheless, a complication arises from the fact that formula F_2 usually includes existentially quantified variables, standing for places that no longer occur in the reduced net N_2. This can complicate some symbolic verification techniques, such as k-induction [SSS00], and impede the use of explicit, enumerative approaches, such as random walk state space exploration. Indeed, in the latter case, we need to solve an integer linear problem for each new state instead of just evaluating a closed formula.

Proposal. To overcome this problem, we propose a new method for projecting the formula F_2 into a quantifier-free one, F_2', that only refers to the places of N_2 and preserves the verdict.

We define our projection as a procedure for quantifier elimination in Presburger Arithmetic (PA) tailored to the specific constraints we handle in E. Whereas quantifier elimination for existential formulas has an exponential complexity in general, our construction has linear complexity and can only decrease the size of a formula. The procedure always terminates and returns either (1) an exact formula F_2' such that F_2' is reachable in N_2 if and only if F_1 is reachable in N_1; or (2) a formula F_2' that is (only) sound, meaning it under-approximates the set of reachable models and, therefore, a witness of F_2' in N_2 necessarily corresponds to a witness of F_1 in N_1, but not conversely. Additionally, our approach includes a simple condition on F_1 that is enough to detect when our result is exact.

We have implemented this procedure into a new tool, called Octant, that can act as a preprocessor, allowing any model checker to benefit from our optimization transparently. Something that was not possible with the E-transform formula from Chapter 3. It means we can use our approach as a front-end to accelerate any model checking tool that supports generalized reachability properties without modifying them.

An interesting outcome of our work is the definition of a non-trivial fragment of existential Presburger arithmetic with good complexity properties that we hope could be applicable in other settings.

Outline and Contributions. We define our quantifier elimination algorithm in Sect. 5.4 and prove its soundness. Our method has been implemented, and we report on the results of

several experiments (Sect. 5.5). We give quantitative evidence about several natural questions raised by our approach. We start by proving the effectiveness of our optimization on both k-induction and random walk. Then, we show that our method can be transparently added to several off-the-shelf verification tools. We demonstrate this fact using three different tools: ITS-Tools [Thi15]; LoLA [Wol18]; and TAPAAL [Dav+12]. Our experiments illustrate the ability to use our optimization as a preprocessing step for any existing tool that can accept the standard input formats used in the MCC. Our experiments include the top three performing tools that participated in the reachability category of this competition, which are, therefore, already optimized for the type of models and formulas used in our benchmark. The results show that reductions are effective on a large set of queries and that their benefits do not overlap with other existing optimizations, an observation already made in [Bøn+19]. We also prove that our procedure often computes an exact projection and compares favorably well with the quantifier elimination method for full Presburger arithmetic implemented in Redlog [DS97] and isl [Ver10]. This indicates that we can solve non-trivial quantifier elimination problems.

5.2 Two Examples of Reachability Formulas

We start by illustrating how some formulas can be projected (and simplified). We use the example in Fig. 5.1 throughout this chapter, composed of two nets (M_1, m_1) and (M_2, m_2), and the relation E, for which we can prove that $(M_1, m_1) \equiv_E (M_2, m_2)$. Since every marking m_2' with $m_2'(p_2) \geqslant 0$ is reachable in M_2, we can deduce, by condition (A2b) of the relaxed E-equivalence (see Definition 4.1), that the reachable markings of M_1 are exactly the solutions of the system $(p_3 = p_0 + p_2 + 4) \wedge (p_0 + p_1 = 10) \wedge (p_4 + p_5 \leqslant 5)$. We also deduce that the net M_1 is unbounded, which is not a problem with our approach.

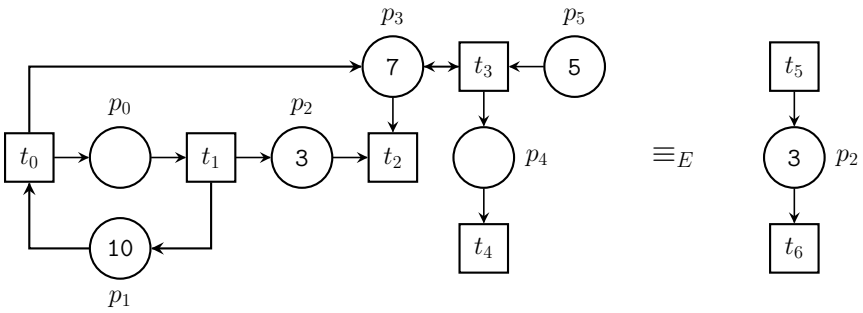


Fig. 5.1 An example of Petri net, (M_1, m_1) (left), and one of its polyhedral reductions, (M_2, m_2) (right), with $E \triangleq \exists a_1, a_2 \,.\, (p_3 = p_0 + p_2 + 4) \wedge (a_1 = p_0 + p_1) \wedge (a_2 = p_4 + p_5) \wedge (a_1 = 10) \wedge (a_2 \leqslant 5)$.

We display the equations generated on our running example in Fig. 5.2, where annotations R and A indicate if an equation is a redundancy or an agglomeration; and the corresponding (unique) TFG.

```
# R |- p3 = p0 + p2 + 4
# A |- a1 = p0 + p1
# A |- a2 = p4 + p5
# R |- a1 = 10
# R |- a2 <= 5
```

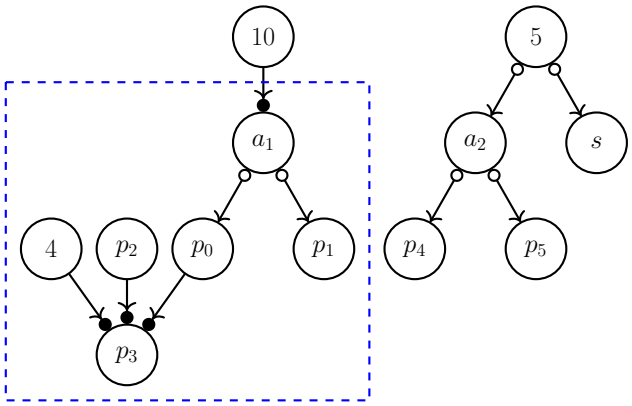


Fig. 5.2 Equations generated from the polyhedral reduction in Fig. 5.1, and the associated TFG (where s is a slack variable for the constraint $a_2 \leqslant 5$).

To keep things simple, we consider the reduction system, $E \triangleq (p_3 = p_0 + p_2 + 4) \wedge (a_1 = p_0 + p_1)$, that is a subset of the constraints obtained in our running example (see the circled part in the TFG of Fig. 4.4).

A first example of a reachability formula is $G_1 \triangleq (p_0 - p_1 + p_3 \leqslant 4)$. We want to eliminate variables $\{p_0, p_1, p_3\}$ from $E \wedge G_1$ to keep only $\{a_1, p_2\}$. Using substitutions and quantifying the variables we wish to eliminate, we map $E \wedge G_1$ into the equivalent formula $\exists p_0, p_1 \,.\, (2\,p_0 - p_1 + p_2 + 4 \leqslant 4) \wedge (a_1 = p_0 + p_1) \equiv \exists p_0 \,.\, (3\,p_0 + p_2 \leqslant a_1)$. From this formula, we can obtain an exact projection of G_1 by using both the isl numerical library [Ver10] and our fast projection method, described below. This gives the projected formula $(p_2 \leqslant a_1)$, whose satisfiability in N_2 is equivalent to G_1 in N_1. We can observe that non-trivial coefficients (like $3\,p_0$) can naturally appear in the problem, even though all the coefficients are 1 or -1 in the initial constraints.

Another example is $G_2 \triangleq (p_0 - p_1 + p_3 = 4)$, whose integer shadow on $\{a_1, p_2\}$ are the solutions to the PA formula $(a_1 \equiv p_2 \bmod 3)$. This set is not convex, since $(0,0)$ and $(0,3)$ are in the integer shadow, but not $(0,2)$. Our fast projection method will compute the formula $(a_1 = p_2 = 0)$ and flag it as an under-approximation.

In the following, we focus on the quantifier-free fragment of PA ($\exists$PA). Without loss of generality, we can consider only formulas in disjunctive normal forms (DNF), with *linear predicates* of the form $(\sum k_i\, x_i) + b \geqslant 0$. We deliberately omit to add a divisibility operator $k \mid \alpha$, which requires that k evenly divides α since it can already be expressed with linear predicates, though at the cost of an extra existentially quantified variable. As a reminder, this fragment corresponds to the set of reachability formulas supported by many model checkers for Petri nets, such as [BRV04; Dav+12; Thi15; Amp+16; Wol18].

5.3 Combining Reduction with Reachability

We can define a counterpart to our notion of polyhedral abstraction which relates to reachability formulas. We show that this equivalence can be used to speed up the verification of properties by checking formulas on a reduced net instead of the initial one (see Theorem 5.1 and its corollary). In the following, we assume that we have two marked nets such that $(N_1, m_1) \equiv_E (N_2, m_2)$. Our goal is to define a relation $F_1 \equiv_E F_2$, between reachability formulas, such that F_1 and F_2 have the same truth values on equivalent models, with respect to E.

Definition 5.1 (Equivalence Between Formulas)**.** *Assume that F_1, F_2 are reachability formulas with variables in P_1 and P_2, respectively, and that $\mathrm{FV}(E) \subseteq P_1 \cup P_2$. We say that formula F_2 implies F_1 up-to E, denoted $F_2 \sqsubseteq_E F_1$, if for every marking $m'_2 \in \mathbb{N}^{P_2}$ such that $m'_2 \models E \wedge F_2$ there exists at least one marking $m'_1 \in \mathbb{N}^{P_1}$ such that $m'_1 \equiv_E m'_2$ and $m'_1 \models E \wedge F_1$.*

$$F_2 \sqsubseteq_E F_1 \quad \textit{iff} \quad \forall m'_2 \,.\, (m'_2 \models E \wedge F_2) \implies \exists m'_1 \,.\, (m'_1 \equiv_E m'_2 \wedge m'_1 \models E \wedge F_1)$$

We say that F_1 and F_2 are equivalent, denoted $F_1 \equiv_E F_2$, when both $F_1 \sqsubseteq_E F_2$ and $F_2 \sqsubseteq_E F_1$.

This notion is interesting when F_1, F_2 are reachability formulas on the nets N_1, respectively N_2. Indeed, we prove that when $F_2 \sqsubseteq_E F_1$, it is enough to find a witness of F_2 in N_2 to prove that F_1 is reachable in N_1.

Theorem 5.1 (Finding a Witness)**.** *Assume that $(N_1, m_1) \equiv_E (N_2, m_2)$ and $F_2 \sqsubseteq_E F_1$, and take a marking m'_2 reachable in (N_2, m_2) such that $m'_2 \models F_2$. Then there exists $m'_1 \in R(N_1, m_1)$ such that $m'_1 \equiv_E m'_2$ and $m'_1 \models F_1$.*

Proof. Assume that we have m'_2 reachable in N_2 such that $m'_2 \models F_2$. By property (A2a) of E-equivalence (Definition 4.1), formula $E \wedge \underline{m'_2}$ is satisfiable, which gives $m'_2 \models E \wedge F_2$. By definition of the E-implication $F_2 \sqsubseteq_E F_1$, we get a marking m'_1 such that $m'_1 \models F_1$ and $m'_1 \equiv_E m'_2$. We conclude that m'_1 is reachable in N_1 thanks to property (A2b). □

Hence, when $F_2 \sqsubseteq_E F_1$ holds, F_2 reachable in N_2 implies that F_1 is reachable in N_1. We can derive stronger results when F_1 and F_2 are equivalent.

Corollary 5.2. *Assume that $(N_1, m_1) \equiv_E (N_2, m_2)$ and $F_1 \equiv_E F_2$, with $\mathrm{FV}(F_i) \subseteq P_i$ for all $i \in 1..2$, then:* (CEX) *property F_1 is reachable in N_1 if and only if F_2 is reachable in N_2; and* (INV) *F_1 is an invariant on N_1 if and only if F_2 is an invariant on N_2.*

Theorem 5.1 means that we can check the reachability (or invariance) of a formula on the net N_1 by checking instead the reachability of another formula (F_2) on N_2. But it does not

indicate how to compute a good candidate for F_2. By Definition 5.1, a natural choice is to select $F_2 \triangleq E \wedge F_1$. We can actually do a bit better. It is enough to choose a formula F_2 that has the same (integer points) solution as $E \wedge F_1$ over the places of N_2. More formally, let $A \triangleq P_1 \setminus P_2$, then if F_2 has the same integer solutions over $\mathbb{N}^{P_2}$ than the Presburger formula $\exists A \,.\, (E \wedge F_1)$, we have $F_1 \equiv_E F_2$. We say in this case that F_2 is the projection of $E \wedge F_1$ on the set P_2, by eliminating the variables in A.

In the next section, we show how to compute a candidate projection formula without resorting to a classical, complete variable elimination procedure on $\exists A \,.\, E \wedge F_1$, when $E \triangleq \exists Q.E'$ is in linear system form. This eliminates a potential source of complexity blow-up. This projection procedure applies to *cubes* only, meaning a conjunction of literals $\bigwedge_{i \in 1..n} \alpha_i$. Given a formula F_1, assumed in DNF, we can apply the projection procedure to each of its cubes, separately. Then the projection of F_1 is the disjunction of the projected cubes. Hence, for the sake of simplicity, we assume from now on that F_1 is a cube formula.

We can use Fourier-Motzkin elimination (FM) as a point of reference. Given a system of linear inequalities S, with variables in V, we denote $\mathrm{FM}_A(S)$ the system obtained by FM elimination of variables in A from S. (We do not describe the construction of $\mathrm{FM}_A(S)$ here, since there exists many good references [Imb93; Mon10] on the subject.) Borrowing an intuition popularized by Pugh in its Omega test [Pug91], we can define two distinct notions of "shadows" cast by the projection of S. On the one hand, we have the *real shadow*, relative to A, which are the integer points (in $\mathbb{N}^{V \setminus A}$) solutions of $\mathrm{FM}_A(S)$. On the other hand, the *integer shadow* of S is the set of markings m' with an integer point antecedent in S. We need the latter to check a query on N_1.

The main source of complexity comes from the following: although the real shadow would be exact when dealing with rational variables, this is no longer true in the integer domain, where the real shadow may contain strictly more solutions than the integer shadow. Moreover, while the real shadow of a convex region is necessarily convex, this is no longer true with the integer shadow. Like with the real shadow, the set of equations computed with our fast projection will always be convex. Unlike FM, our procedure will compute an under-approximation of the integer shadow, not an over-approximation. Also, we never rearrange or create more inequalities than the one contained in S; but instead rely on variable substitution.

5.4 Formula Rewriting

We assume given a relation $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$, and its associated well-formed TFG written $[\![E']\!]$. We consider that F_1 is a cube of n literals, $F_1 \triangleq \bigwedge_{i \in 1..n} \alpha_i^0$. Our algorithm rewrites each α_i^0 by applying iteratively an elimination step, described next, according to the constraints expressed in $[\![E']\!]$. The final result is a conjunction $F_2 \triangleq \bigwedge_{i \in 1..n} \beta_i$, where each literal β_i has support in N_2. Rewriting can only replace a variable with a group of other variables that are its predecessors in the TFG, which ensures termination in polynomial time

(in the size of E'). Although the result has the same number of literals, it usually contains many redundancies and trivial constant comparisons so that, after simplification, F_2 can actually be much smaller than F_1.

A reduction step (to be applied repeatedly) takes as input the current set of literals, $C \triangleq (\alpha_i)_{i\in 1..n}$, and modifies it. To ease the presentation, we also keep track of a set of variables, B, such that $\bigcup_{i\in 1..n} \mathrm{FV}(\alpha_i) \subseteq B$. We assume that every literal is in *normal form*, $\alpha_i \triangleq (\sum_{p_j\in B} k^i_j\, p_j) + b_i \geqslant 0$, where the k^i_j's and b_i are in $\mathbb{Z}$. In the following, we denote $\alpha_i(q)$ the coefficient associated with variable q in α_i. We also use $\max_X \alpha_i$ and $\min_X \alpha_i$ for the maximal (resp. minimal) coefficient associated with variables in $X \subseteq B$.

$$\alpha_i \triangleq \sum_{p\in B} \alpha_i(p)\, p + b_i \geqslant 0 \qquad \text{and} \qquad \max_X \alpha_i \triangleq \max\ \{\alpha_i(p) \mid p \in X\} \tag{5.1}$$

5.4.1 Highest Literal Factor

We define the *Highest Literal Factor* (HLF) of a set of variables X with respect to a set of normalized literals $(\alpha_i)_{i\in I}$. In the simplest case, the HLF of X with respect to a single literal, α, is the subset of variables in X with the highest coefficients in α. Then, the HLF of X with respect to a set of literals is the—possibly empty—intersection of the HLFs of X with respect to each literal. When non-empty, it means that at least one variable in X always has the highest coefficient, and we say then that the whole set X is *polarized* with respect to the literals (α_i).

$$\begin{aligned} \mathrm{HLF}_X(\alpha_i) &\triangleq \{p \in X \mid \alpha_i(p) = \max_X \alpha_i\} \\ \mathrm{HLF}_X(\alpha_i)_{i\in I} &\triangleq \bigcap_{i\in I} \mathrm{HLF}_X(\alpha_i) \end{aligned} \tag{5.2}$$

Definition 5.2 (Polarized Set of Constraints)**.** *A set of variables $X \subseteq \mathrm{FV}(C)$ is said polarized with respect to a set of normalized literals C when $\mathrm{HLF}_X(C) \neq \emptyset$.*

We prove below that our procedure is exact when the variables we eliminate are polarized. While this condition looks pretty restrictive, we observe that it is often true with the queries used in our experiments (our projection is complete for 80% of the formulas used in the MCC).

5.4.2 Formal Procedure

An elimination step is a reduction written $(B, C) \mapsto (B', C')$ where $C \triangleq (\alpha_i)_{i\in 1..n}$ and $B' \subsetneq B$, defined as one of the three cases below (one for redundancy, and two for agglomerations, depending on whether the removed variables are polarized or not). We assume that literals are in normal form and that X is a set of variables $\{x_1, \ldots, x_k\}$. Note the precondition $\downarrow X \cap B \subseteq X$

(or $\downarrow p \cap B \subseteq \{p\}$) on all rules, which forces them to be applied bottom-up on the TFG (remember it is a DAG). We give a short example of how to apply rules (AGP) and (AGD) just after Theorem 5.3.

(RED) If $X \twoheadrightarrow p$ and $\downarrow p \cap B \subseteq \{p\}$ then $(B, C) \mapsto (B', C')$ holds, where $B' \triangleq B \setminus \{p\}$ and C' is the set of literals α_i' obtained by normalizing the linear constraint $\alpha_i\{p \leftarrow x_1 + \cdots + x_k\}$. That is, we substitute p with $\sum_{x_i \in X} x_i$ in C, which is the meaning of the redundancy equation (constraint (T4) in Definition 4.4).

(AGP) If $a \circ\!\!\rightarrow X$ with $\downarrow X \cap B \subseteq X$, $a \in B$ and X polarized with respect to C, then $(B, C) \mapsto (B', C')$ holds, where $B' \triangleq B \setminus X$, and, by taking $x_j \in \mathrm{HLF}_X(C)$, we define C' as the set of literals α_i' obtained by normalizing the linear constraint $\alpha_i\{x_l \leftarrow 0\}_{l \neq j}\{x_j \leftarrow a\}$. That is, we eliminate the variables x_l, different from x_j, from C and replace x_j with a, where x_j is a variable of X that always has the highest coefficient in each literal (among the ones of X).

(AGD) If $a \circ\!\!\rightarrow X$ with $\downarrow X \cap B \subseteq X$, $a \in B$, and X is not polarized with respect to C. Then $(B, C) \mapsto (B', C')$ holds, where $B' \triangleq B \setminus X$ and C' is the set of literals α_i' obtained by normalizing the linear constraint $\alpha_i\{x_l \leftarrow 0\}_{l \neq j}\{x_j \leftarrow a\}$ such that $\alpha_i(x_j) = \min_X \alpha_i$. Meaning we eliminate the variables x_l different from x_j from α_i and replace x_j with a, where x_j is a variable with the smallest coefficient in α_i (among the ones of X). Note that the chosen variable x_j is not necessarily the same in every literal of C.

We aim to preserve the semantics of formulas at each reduction step, in the sense of the relations $\sqsubseteq_E$ and $\equiv_E$. In the following, we use C to represent both a set of literals $(\alpha_i)_{i \in I}$ and the cube formula $\bigwedge_{i \in I} \alpha_i$. We can prove that the elimination steps corresponding to the redundancy (RED) and polarized agglomeration (AGP) cases preserve the semantics of the formula C. On the other hand, a non-polarized agglomeration step (AGD) may lose some markings.

5.4.3 Proof of the Procedure

We prove the main result of the chapter (Theorem 5.3), namely that fast quantifier elimination preserves the integer solutions of a system when we only have polarized agglomerations. To this end, we need to prove two results. First, Theorem 5.3, which entails the soundness of one elimination step. It also entails completeness for rules (RED) and (AGP). Second, we prove a progress property (Theorem 5.6 below), which guarantees that we can apply elimination steps until we reach a set of literals C' with support on the reduced net N_2.

Theorem 5.3 (Projection Equivalence)**.** *If* $(B, C) \mapsto (B', C')$ *is a (RED) or (AGP) reduction then* $C' \equiv_{\exists Q \setminus B.E'} C$*; otherwise* $C' \sqsubseteq_{\exists Q \setminus B.E'} C$*.*

We prove Theorem 5.3 in two steps. We start by proving that elimination steps are sound, meaning that the integer solutions of C' are also solutions of C (up-to $\exists Q \setminus B \,.\, E'$). Then, we prove that elimination is complete for rules (RED) and (AGP). In the following, we use C to represent both a set of literals $(\alpha_i)_{i \in I}$ and the cube formula $\bigwedge_{i \in I} \alpha_i$.

Lemma 5.4 (Soundness)**.** *If* $(B, C) \mapsto (B', C')$ *then* $C' \sqsubseteq_{\exists Q \setminus B.E'} C$*.*

Proof. Take a valuation m' of $\mathbb{N}^{B'}$ such that $m' \models E_B \wedge C'$, where E_B is $\exists Q \setminus B \,.\, E'$. We want to show that there exists a marking m of $\mathbb{N}^B$ such that $m \equiv_{E_B} m'$ satisfying $E_B \wedge C$.

We have three possible cases corresponding to rule (RED), (AGP), or (AGD). In each case, we provide a marking m built from m'. Since $m \equiv_{E_B} m'$ is enough to prove $m \models E_B$, we only need to check two properties: first that $m \equiv_{E_B} m'$ (i), then that $m \models \alpha$ for every literal α in C (ii).

(RED) In this case we have $X \twoheadrightarrow p$ and $B' \triangleq B \setminus \{p\}$, with $X \triangleq \{x_1, \ldots, x_k\}$. We can extend m' into the unique valuation m of $\mathbb{N}^B$ such that $m(p) = m'(x_1) + \cdots + m'(x_k)$ and $m(v) = m'(v)$ for all other nodes v in $B \setminus \{p\}$. Since $p = x_1 + \cdots + x_k$ is an equation of E_B (condition (T4)) we obtain that $m' \equiv_{E_B} m$ and therefore also $m \models E_B$ (i).
We now prove that $m \models C$. The literals in C' are of the form $\alpha\sigma$ with σ the substitution $\{p \leftarrow x_1 + \cdots + x_k\}$ and α in C. Remember that, with our notations, we have $m \models \alpha$ if and only if $\alpha\{m\}$ is satisfiable. By hypothesis, $m' \models \alpha\sigma$. Hence, $\alpha\sigma\{m'\}$ is satisfiable, which is equivalent to $\alpha\{m\}$ satisfiable, and therefore $m \models \alpha$ (ii), as required.

(AGP) In this case we have $a \circ\!\!\rightarrow X$ with $X \triangleq \{x_1, \ldots, x_k\}$, polarized relative to C, and $B' \triangleq B \setminus X$. We consider x_j in X the variable in $\mathrm{HLF}_X(C)$ that was chosen in the reduction; meaning that C' is a conjunction of literals of the form $\alpha\{x_l \leftarrow 0\}_{l \neq j}\{x_j \leftarrow a\}$, with α a literal of C. Given m' a model of C', we define m the unique marking on $\mathbb{N}^B$ such that $m(x_j) = m(a)$, $m(x_l) = 0$ for all $l \neq j$, and $m(v) = m'(v)$ for all other variables v in $B \setminus X$.
From Lemma 4.3 of Chapter 4 (the "token propagation" property of TFGs), we know that any distribution of $m(a)$ tokens, in place a, over the $(x_i)_{i \in 1..k}$, is also a model of E', hence of E_B. Which means that $m \models E_B$ $(*)$. Note that the token propagation Lemma does not imply that the value of $m(v)$, for the nodes "below X" (v in $\downarrow X$), is unchanged. This is not problematic since the side condition $\downarrow X \cap B \subseteq X$ ensures that these nodes are not in B and, therefore, cannot influence the value of $\alpha\{m\}$.

Consider a literal α in C. Since $m' \models C'$, we have $\alpha\{x_l \leftarrow 0\}_{l\neq j}\{x_j \leftarrow a\}\{m'\}$ satisfiable, which is exactly $\alpha\{m\}$, since $\downarrow X \cap B \subseteq X$, as needed (ii).

(AGD) In this case we have $a \circ\!\!\rightarrow X$ with $X \triangleq \{x_1, \ldots, x_k\}$, non-polarized relative to C, and $B' \triangleq B \setminus X$. We know that $m' \models E_B$, therefore there is a marking m of $\mathbb{N}^B$ that extends m' such that $m \equiv_{E_B} m'$ (i).
Consider a literal α in C. By definition of (AGD), we have an associated literal $\alpha' \triangleq \alpha\{x_l \leftarrow 0\}_{l\neq j}\{x_j \leftarrow a\}$ in C' such that $\alpha(x_j) = \min_X \alpha_i$. Since the coefficient of x_j is minimal, we have $\sum_{i\in 1..k} \alpha(x_i)\, m(x_i) \geqslant \alpha(x_j) \sum_{i\in 1..k} m(x_i) = \alpha(x_j)\, m'(a)$, and therefore $\sum_{v\in B} \alpha(v)\, m(v) \geqslant \sum_{v\in B'} \alpha'(v)\, m'(v)$. The result follows from the fact that $\alpha\{m\}$ is satisfiable (ii).

□

Now, we prove that our quantifier elimination step for the (RED) and (AGP) cases leads to a complete projection; that is, any solution of the initial formula corresponds to a projected solution in the projected formula.

Lemma 5.5 (Completeness)**.** *If $(B, C) \mapsto (B', C')$ is a (RED) or (AGP) reduction then $C \sqsubseteq_{\exists Q\setminus B.E'} C'$.*

Proof. Take a marking m of $\mathbb{N}^B$ such that $m \models E_B \wedge C$, where E_B is $\exists Q \setminus B \,.\, E'$. We want to show that there exists a valuation m' of $\mathbb{N}^{B'}$ such that $m \equiv_{E_B} m'$ (i) and $m' \models C'$ (ii). This is enough to prove $m' \models E_B \wedge C'$. We have two cases corresponding to the rules (RED) and (AGP).

(RED) In this case we have $X \twoheadrightarrow p$ with $X \triangleq \{x_1, \ldots, x_k\}$ and $B' \triangleq B \setminus \{p\}$. We define m' as the (unique) projection of m on B'. Since $m \models E_B$ we have $m' \equiv_{E_B} m$ (i).
Also, literals in C' are of the form $\alpha' \triangleq \alpha\{p \leftarrow x_1 + \cdots + x_k\}$ where α is a literal of C. Since $m(p) = \sum_{i\in 1..k} m(x_i)$ and m is a model of α, it is also the case that m' is a model of α' $(**)$.

(AGP) In this case we have $a \circ\!\!\rightarrow X$ with $X \triangleq \{x_1, \ldots, x_k\}$ and $B' \triangleq B \setminus X$. We define m' as the (unique) projection of m on B', by taking $m'(a) = \sum_{i\in 1..k} m(x_i)$. Since $m \models E_B$ we have $m' \equiv_{E_B} m$ (i).
We consider x_j in X the variable in $\mathrm{HLF}_X(C)$ that was chosen in the reduction; meaning that C' is a conjunction of literals of the form $\alpha\{x_l \leftarrow 0\}_{l\neq j}\{x_j \leftarrow a\}$, with α a literal of C. Since $\sum_{i\in 1..k} \alpha(x_i)\, m(x_i) \leqslant \alpha(x_j) \sum_{i\in 1..k} m(x_i) = \alpha(x_j)\, m'(a)$, we have m' is a model of α' (ii).

□

The final step of our proof relies on a *progress property*, meaning there is always a reduction step to apply except when all the literals have their support on the reduced net, N_2. This property relies on relation $\mapsto^*$, the transitive closure of $\mapsto$. Together with Theorem 5.3, the

progress theorem ensures the existence of a sequence $(P, C) \mapsto^* (P_2, C')$, such that $C \equiv_{E'} C'$ (or $C' \sqsubseteq_{E'} C$ if we have at least one non-polarized agglomeration). In this context, P is the set of all variables occurring in the TFG of E'; therefore, it contains $P_1 \cup P_2$.

Theorem 5.6 (Progress)**.** *Assume that* $(P, F_1) \mapsto^* (B, C)$ *then either* $B \subseteq P_2$*, the set of places of* N_2*, or there is an elimination step* $(B, C) \mapsto (B', C')$ *such that* $\mathrm{FV}(C') \subseteq B'$ *and the places removed from* B *have no successors in* B'*: for all places* p *in* $B \setminus B'$*, we have* $\downarrow p \cap B \subseteq \{p\}$*.*

Proof. Assume that we have $(P, F_1) \mapsto^* (B, C)$ and $B \nsubseteq P_2$.

By condition (T6) in Definition 4.4, we know that P_2 are roots in the TFG $[\![E']\!]$. We consider the set of nodes in $B \setminus P_2$, corresponding to nodes in B with at least one parent. Also, by condition (T5), we know that $[\![E']\!]$ is acyclic, then there are nodes in $B \setminus P_2$ with no successors in B. We call this set L. Hence, $L \triangleq \{v \mid v \in B \setminus P_2 \wedge \downarrow v \cap B \subseteq \{v\}\}$.

Take a node p in L. We have two possible cases. If there is a set X such that $X \twoheadrightarrow\!\!\bullet\, p$, we can apply the (RED) elimination rule. Otherwise, a node a and a set $X \subseteq L$ (by condition (T2)) such as $a \circ\!\!\rightarrow X$ with $p \in X$ exists. In this case, apply rule (AGP) or (AGD), depending on whether the agglomeration is polarized. □

5.4.4 Examples on Polarized and Non-Polarized Constraints

Theorem 5.3 is enough to prove our main result, that is, $F_2 \equiv_E F_1$ when all the reduction steps corresponding to an agglomeration are on polarized variables. With two examples, let us illustrate why our approach is sound. Assume that we want to eliminate an agglomeration $a \circ\!\!\rightarrow \{q, r\}$, meaning that we have the condition $a = q + r$ and that both q and r must disappear. We consider two examples of systems, each with only two literals. One polarized (left), with the result of applying (AGP) below; another with the result of (AGD).

$$\begin{array}{l} 3p + 2q - 1r \geqslant 0 \\ 2p + 1q + 1r - 5 \geqslant 0 \\ \downarrow \\ 3p + 2a \geqslant 0 \\ 2p + 1a - 5 \geqslant 0 \end{array} \qquad \begin{array}{l} 3p + 2q - r \geqslant 0 \\ -p + q + 2r - 5 \geqslant 0 \\ \downarrow \\ 3p - a \geqslant 0 \\ -p + a - 5 \geqslant 0 \end{array}$$

In the left example, the set $\{q, r\}$ is polarized with respect to the initial system (top), with the highest literal factor being q. So we replace q with a in both literals and eliminate r. Uninvolved variables (just p in this case) are left unchanged. Both systems are equivalent because it is possible to show that every solution of the initial system (top) corresponds to a solution of the resulting system (bottom) by taking $a = q + r$. Conversely, every solution of the resulting system can be associated with a solution of the initial system by taking $q = a$ and $r = 0$.

The initial system on the right (top) is non-polarized: the HLF relative to $\{q, r\}$ is $\{q\}$ for the first literal ($+2q$ versus $-r$) and $\{r\}$ in the second ($+q$ versus $2r$). So, we substitute a to the variable with the lowest literal factor in each literal and remove the other variable (q or r). This is sound because we take into account the worst case in each literal. However, this is not complete because we may be too pessimistic. For instance, the resulting system has no solution for $p = 2$ because it entails $a \leqslant 6$ and $a \geqslant 7$. But $p = 2, q = 3, r = 2$ is a model of the initial system.

Remark. We have designed the rule (AGD) to obtain at least $F_2 \sqsubseteq_E F_1$ when the procedure is not complete (instead of $F_2 \equiv_E F_1$), which is useful for finding witnesses (see Theorem 5.1). Alternatively, we could propose a variant rule, say (AGD'), which chooses the variable x_j having the highest coefficient in α_i, that is $\alpha_i(x_j) = \max_X \alpha_i$. This variant guarantees a dual result, that is, $F_1 \sqsubseteq_E F_2$. In this case, if F_2 is not reachable, then F_1 is not reachable, which is useful to prove invariants.

5.5 Experimental Results

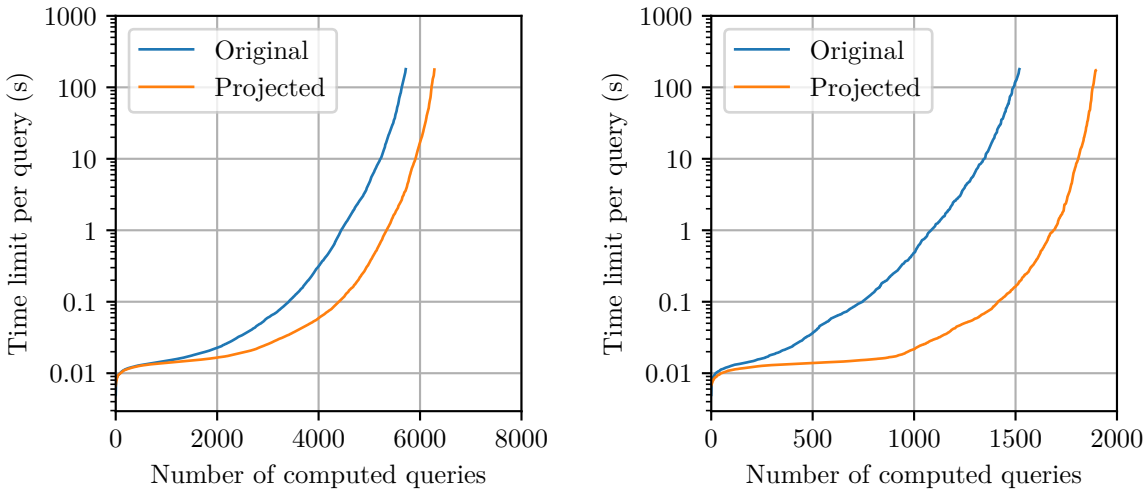
We have implemented our fast quantifier elimination procedure in a tool, called Octant. As previously, we use the set of models and formulas collected from the 2023 edition of the Model Checking Contest. The net reductions applied are the same as in the previous chapter using the tool Reduce, and we still denote r as the reduction ratio.

The size of the reduction system, E, is proportional to the number of places that are removed. To give a rough idea, the mean number of variables in E is 1 375, with a median value of 114 and a maximum of about 62 000. The number of literals is also rather substantial: a mean of 869 literals (62% of agglomerations and 38% of redundancies), with a median of 27 and a maximum of about 38 000.

We report on the results obtained on two main categories of experiments: first with model checking, to evaluate if our approach is effective in practice, using real tools; then to assess the precision and performance of our fast quantifier elimination procedure.

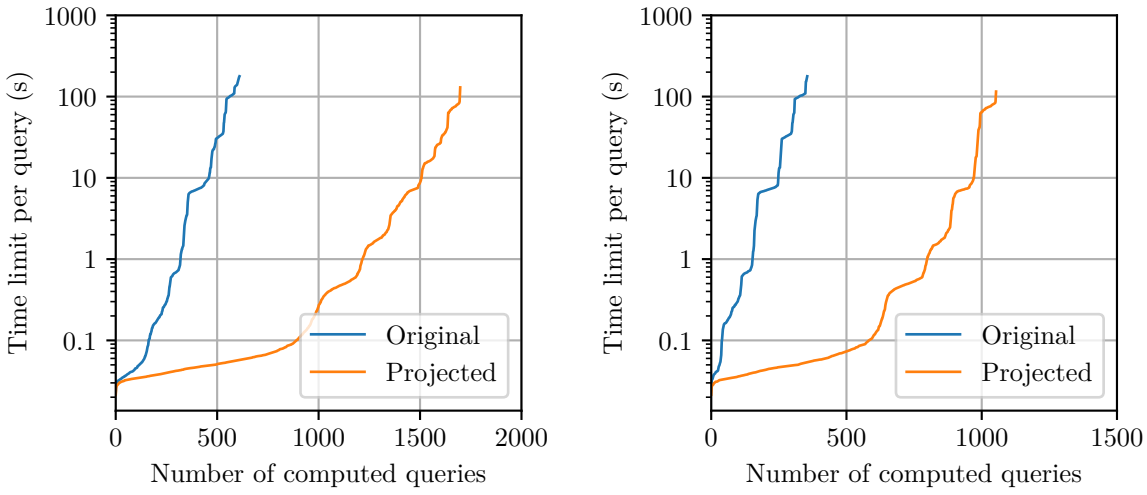
5.5.1 Impact on Standard Model Checking Procedures

We start by showing the effectiveness of our approach on both random walk and k-induction. This is achieved by comparing the computation time, with and without reductions, on a model checker that provides a "reference" implementation of these techniques. (Without any other optimizations that could interfere with our experiments.) It is interesting to test the results of our optimization separately on these two techniques. Indeed, each technique is adapted to a different category of queries: properties that can be decided by finding a witness, meaning true



(a) $r \in [0.01, 0.5[$ (b) $r \in [0.5, 1]$

Fig. 5.3 Random walk w/wo reductions.



(a) $r \in [0.01, 0.5[$ (b) $r \in [0.5, 1]$

Fig. 5.4 k-induction w/wo reductions.

EF formulas or false AG ones, can often be checked more efficiently using a random state space exploration. On the other hand, symbolic verification methods are required to check invariants.

We display our results using the four "cactus plots" in Figs. 5.3 and 5.4. (Note that we use a logarithmic scale for the time value). We distinguish between two categories of instances depending on their reduction ratio. Plots on the left are for models with a low or moderate reduction ratio (value of r less than 50%) and on the right are for models that can be reduced by more than half. The first category amounts to roughly 9 015 queries (70% of our benchmark), while the second category contains about 4 000 queries. The most interesting conclusion we can

draw from these results is that our approach is beneficial even when there is only a limited amount of reductions.

Our experiments were performed with a maximal timeout of 180 s and integrated the projection time into the total execution time. We observe moderate performance gains with random exploration (with $\times 1.05$ more computed queries on low-reduction instances and $\times 1.24$ otherwise) and good results with k-induction (respectively $\times 2.52$ and $\times 2.96$).

We obtain better results if we focus on queries that take more than 1 s on the original formula, which indicates that reductions are most effective on "difficult problems" (there is not much to gain on instances that are already easy to solve). With random walk, for instance, the gain becomes $\times 1.25$ for low-reduction instances and $\times 1.87$ otherwise. The same observation is true with k-induction, with performance gains of $\times 5.38$ and $\times 4.43$, respectively.

5.5.2 Impact Under Real Conditions

We also tested our approach by transparently adding polyhedral reductions as a front-end to three different model checkers: ITS-Tools [Thi15], LoLA [Wol18] and TAPAAL [Dav+12] that implement portfolios of verification techniques. All three tools regularly compete in the MCC (on the same set of queries we use for our benchmark), and TAPAAL and ITS-Tools share the top two places in the reachability category of the 2022 and 2023 editions.

We ran each tool on our set of complete projections, which amounts to almost 80 000 runs (one run for each tool, once on both the original and the projected query). We obtained a 100% reliability result, meaning that all tools gave compatible results on all the queries and, therefore, compatible results on the original and projected formulas.

A large part of the queries can be computed by all the tools in less than 100 ms and can be considered as easy. These queries are useful for testing reliability but can skew the interpretation of results when comparing performances. This is why we decided to focus our results on a set of 897 *challenging queries*, which we define as queries for which either TAPAAL or ITS-Tools, or both, cannot compute a result before projection. The 897 challenging queries (4% of queries) are well distributed since they cover 212 different instances (13% of all instances), themselves covering 45 different models (20% of the models).

We display the results obtained on the challenging queries, for a timeout of 180 s, in Table 5.1. We provide the number of computed queries before and after projection, with the mean and median speed-up (the ratio between the computation time with and without projection). For each tool, the "Exclusive" column reports the number of queries that can only be computed using the projected formula. Note that we may sometimes timeout with the projected query but obtain a result without. This can be explained by cases where the size of the formula blows up during the transformation into DNF.

Tool	# Computed Queries		Speed-up		# Exclusive Queries
	Original	Projected	Mean	Median	
ITS-Tools	281	333	1.63	1.04	98
LoLA	188	241	10.91	1.40	86
TAPAAL	168	274	1.43	1.10	134

Table 5.1 Impact of projection on the challenging queries.

We observe substantial performance gains with our approach and can solve about half of the challenging queries. For instance, we can compute $\times 1.63$ more challenging queries with TAPAAL using projections than without. (We display more precise results on TAPAAL, the winner of the MCC 2022 edition, in Fig. 5.5.) All these results show that polyhedral reductions are effective on a large set of queries and that their benefits do not significantly overlap with other existing optimizations. This observation was already made, independently, in [Bøn+19].

The approach implemented in Octant was partially included in the version of our model checker, called SMPT, that participated in the MCC 2023 edition. We mainly left aside the handling of under-approximated queries when the formula projection is incomplete. While SMPT placed third in the reachability category, the proportion of queries it was able to solve raised by 5.5% between 2022 (without the use of Octant) and 2023, to reach a ratio of 93.6% of all queries solved with our tool. This is a substantial result, considering that the ratios for ITS-Tools and TAPAAL in 2023 are respectively 94.6% and 94.3%.

5.5.3 Performance Evaluation of Fast Elimination

Our last set of experiments is concerned with the accuracy and performance of our quantifier elimination procedure implement in the tool Octant. We decided to compare our approach with Redlog [DS97] and isl [Ver10].

We display our results in the cactus plot of Fig. 5.6, where we compare the number of projections we can compute given a fixed timeout. We observe a significant performance gap. For instance, with a timeout of 60 s, we are able to compute 17 389 projections, out of 17 472 queries (99.5%), compared to 10 742 (61%) with isl and 5 754 (33%) with Redlog. So, an increase of $\times 1.77$. This provides more empirical evidence that the class of linear systems we manage is not trivial, or, at least, does not correspond to an easy case for the classical procedures implemented in isl and Redlog. We also have good results concerning the precision of our approach since we observe that about 80% of the projections are complete. Furthermore, projections are inexpensive. For instance, the computation time is less than 1 s for 97% of the formulas. We also obtained a median reduction ratio (computed as for the number of places) of 0.2 for the number of cubes and their respective number of literals.

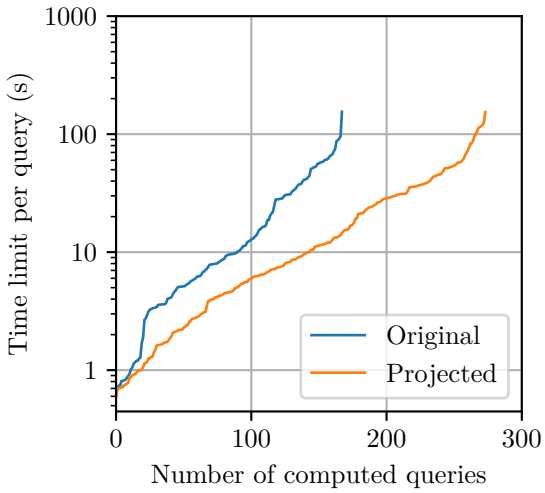


Fig. 5.5 Tapaal w/wo reductions.

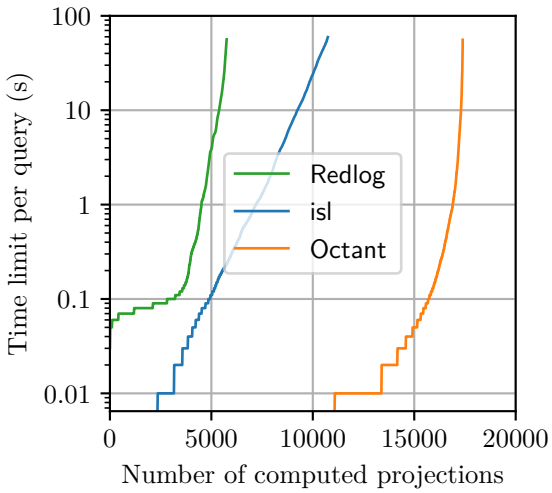


Fig. 5.6 Redlog isl vs fast elimination.

5.6 Discussion

In this chapter, we broaden the approach proposed in Chapter 3 to a larger set of verification methods, most particularly k-induction, which is useful to prove invariants, and simulation (or random walk state space exploration), which is useful for finding counter-examples. We also find a new use for TFGs (from Chapter 4) as the backbone of our variable elimination algorithm and show that we can efficiently eliminate variables in systems of the form $E \wedge F$ for an arbitrary F. There exist some well-known classes of linear systems where variable elimination has a low complexity. A famous example is given by the link between *unimodular matrices* and *integral polyhedra* [HK10], which is related to many examples found in abstract domains used in program verification, such as *systems of differences* [AS80] or *octagon* [Min06; JM09]. To the best of our knowledge, none of the known classes correspond to what we define using TFGs.

We formulated our method as a quantifier elimination procedure for a restricted class of linear systems. There is a rich literature about quantifier elimination in Presburger arithmetic, such as *Cooper's algorithm* [Coo72; Haa18] or the *Omega test* [Pug91] for instance, and how to implement it efficiently [HLL92; LS07; Mon10]. These algorithms have been implemented in several tools, using many different approaches: automata-based, e.g. TaPAS [LP09]; inside computer algebra systems, like with Redlog [DS97]; or in program analysis tools, like isl [Ver10], part of the Barvinok toolbox. Another solution would have been to retrieve "projected formulas" directly from SMT solvers for linear arithmetic, which often use quantifier elimination internally. Unfortunately, this feature is not available, even though some partial solutions have been proposed recently [Bar+22]. All the exact methods that we tested have proved impractical in our case. This was to be expected. Quantifier elimination can be very complex, with an

exponential time complexity in the worst case (for existential formulas as we target); it can generate very large formulas; and it is highly sensitive to the number of variables, when our problem often involves several hundreds and sometimes thousands of variables. Also, quantifier elimination often requires the use of a divisibility operator (also called *stride format* in [Pug91]), which is not part of the logic fragment that we target.

Another set of related work is concerned with *polyhedral techniques* [FL11], used in program analysis. For instance, our approach obviously shares similarities with works that try to derive linear equalities between variables of a program [CH78], and polyhedral reductions are very close in spirit to the notion of linear dependence between vectors of integers (markings in our case) computed in compiler optimizations. Another indication of this close relation is the fact that isl, the numerical library that we use to compare our performances, was developed to support polyhedral compilation. We need to investigate this relation further and see if our approach could find an application with program verification.

In a nutshell, we proposed a quantifier elimination procedure that can benefit from polyhedral reductions and be used transparently as a preprocessing step of existing model checkers. The main characteristic of our approach is to rely on a graph structure, the Token Flow Graphs, that encodes the specific shape of our reduction equations, E.

From a more theoretical viewpoint, we have characterized a fragment of Presburger arithmetic that has interesting complexity properties. More work is needed to fully understand if this fragment corresponds to a well-known class of constraints and if our projection algorithm could be helpful in another setting. In the meantime, we are also looking into ways to improve the precision of our projection in the case where we encounter non-polarized sets of constraints.

This work has been published in::

- N. Amat, S. Dal Zilio, and D. Le Botlan. “Project and Conquer: Fast Quantifier Elimination for Checking Petri Nets Reachability”. In: *Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Lecture Notes in Computer Science. Springer, 2024. DOI: 10.1007/978-3-031-50524-9_5

A conference artifact is available on Zenodo:

- N. Amat, S. Dal Zilio, and D. Le Botlan. *Artifact for VMCAI 2024 Paper "Project and Conquer: Fast Quantifier Elimination for Checking Petri Net Reachability"*. Zenodo, 2023. DOI: 10.5281/zenodo.10061156

The tool related to this chapter is:

- Octant https://github.com/nicolasAmat/Octant

Chapter 6

Concurrency Relation Computation

By Leveraging Token Flow Graphs

> Controlling complexity is the essence of computer programming.
>
> Brian Kernighan

In this chapter, we leverage Token Flow Graphs (TFGs) to efficiently compute the concurrency relation of a net, that is, all pairs of places that can be marked simultaneously in some reachable marking. We hope this more complex reachability problem showcases the benefits of polyhedral reduction using TFGs.

The "acceleration" algorithm is implemented in the tool Kong, which we evaluate on the collection of safe instances used during the Model Checking Contest. Even if we restrict this application to safe nets, we show that the approach works well.

6.1 Introduction

This chapter refers to the Token Flow Graph data structure introduced in Chapter 4 and concludes the problems accelerated by TFGs in this thesis.

Context. We presented some applications to leverage Token Flow Graphs in the previous chapters. In particular, we illustrated the presentation of TFGs in Chapter 4 by tackling the problem of checking the reachability of a given marking. We also used this structure in Chapter 5 to accelerate our first problem of interest, the generalized reachability problem, by eliminating variables in the E-transform formula (defined in Chapter 3) that no longer occur in the reduced net, and therefore act as existentially quantified variable.

Challenge. To demonstrate the versatility of the polyhedral approach, we apply it to the concurrent places problem, that is, enumerating all pairs of places that can be marked simultaneously in some reachable marking. Concurrent places generalize the usual notion of dead places and are particularly useful for decomposing a Petri net into synchronized automata executing in parallel (e.g., NUPNs) [BGP20; BG21].

Although such computation could be done by reusing some existing Petri net model checker, this approach would not be efficient, as the number of temporal-logic formulas to be evaluated would be quadratic in the number of places: a more "global" algorithm should be preferred. In this case, standard reduction techniques such as slicing [Rak12] or structural reductions [Thi20; Thi21] are not feasible because all the places are necessary for our problem. Here, using an equation system (E) for tracing back the effect of reductions is fundamental.

Proposal. In this chapter, we propose a similar approach to the one we proposed for the marking reachability problem (see Chapter 4). Indeed, starting from a tool for solving the concurrent places problem, we provide an augmented version of this tool that takes advantage of reductions. The augmented tool can compute the solution for an initial instance, say on some net N, by solving it on a reduced version of N and then reconstructing a correct solution for the initial instance. Our approach takes the form of an "inverse transform" that relies only on E and does not involve expensive preprocessing on the reduced net. We illustrate our approach by augmenting the tool Cæsar.BDD, part of the CADP toolbox [BG21; INR], that uses BDD techniques to explore the state space of a net and find concurrent places.

We show that our approach can result in massive speed-ups since the reduced net may have far fewer places than the initial one, and the number of places is a predominant parameter in the concurrency relation computation.

Outline and Contributions In Sect. 6.2, we define our problem of interest and one of its applications, e.g., the decomposition of Petri nets into automata networks. Then in Sect. 6.3, we present some additional results on Token Flow Graphs, mainly the safeness preservation. Section 6.4 contains our main contributions. We propose an algorithm for finding concurrent places (proof in Sect. 6.5 and running example in Sect. 6.6) and show in Sect. 6.7 how to adapt it to situations where we only have partial knowledge of the residual concurrency relation. Finally, we propose in Sect. 6.8 a simple method to transpose a NUPN decomposition of a net to an equivalent one on the reduced version of this net. We can draw a parallel with our formula projection method, that is, we start from a problem related to the initial net, in this context with NUPN information, and provide a method to project it on the reduced one.

Our approach has been implemented, and computing experiments show that reductions are effective on a large set of safe nets (Sect. 6.9). We observe that even with a moderate amount of reductions, we can compute complete results much faster with reductions than without and

we also show that we perform well with incomplete relations, where we are both faster and more accurate.

6.2 The Concurrent Places Problem and One of Its Applications

In this section, we recall the concurrent places problem, mentioned in Sect. 1.3, and its main application, the decomposition of Petri nets into automata networks.

6.2.1 The Concurrent Places Problem

Given a net, two places p and p' are *concurrent* if and only if a reachable marking m exists such that both p and p' have at least a token. This relation is symmetric and quasi-reflexive; it is reflexive if the net has no dead place [BG21], i.e., no place that has no token in any reachable marking.

Definition 6.1 (Dead and Concurrent Places)**.** *We say that a place p of (N, m_0) is* nondead *if there is m in $R(N, m_0)$ such that $m(p) > 0$. Similarly, we say that places p, q are* concurrent*, denoted $p \parallel q$, if there is m in $R(N, m_0)$ such that both $m(p) > 0$ and $m(q) > 0$. By extension, we use the notation $p \parallel p$ when p is nondead. We say that p, q are nonconcurrent, denoted $p \# q$ when they are not concurrent.*

This relation characterizes those parts of the net that can be simultaneously active. It is mentioned in many publications under various names, such as *coexistency defined by markings* [Jan84], *concurrency graph* [Kar12; Wiś+14], or *concurrency relation* [Kov92; SY95; KE96; Kov00; GS04; GS06], etc. These definitions slightly differ by minor details, such as the kind of Petri nets considered or the handling of reflexivity, i.e., whether and when a place is concurrent or not with itself.

While most of our results are valid in the general case—with nets that are not necessarily bounded and without any restrictions on the flow functions (the weights of the arcs)—our tool and experiments on the concurrency relation focus on the class of safe nets (1-bounded). The reason for this choice is that the tool we are augmenting, Cæsar.BDD, only addresses the problem on safe nets, as the main use case we present next is limited to this class.

Finally, given a net, the problem of computing all its pairs of concurrent places is PSPACE-complete [BG21]. Most approaches for decomposing a net into a set of concurrent automata or a NUPN [BGP20] require knowledge about concurrent places.

6.2.2 Nested-Unit Petri Nets

Nested-Unit Petri Nets (NUPNs, for short) [Gar15; Gar19] are an extension of Petri nets for expressing *locality* and *hierarchy* properties of concurrent systems. The concept of NUPN is

not recent (see, e.g., [GS90]). However, it has been adopted by recent Petri net analysis tools, which increase their performance by exploiting NUPN information about locality and hierarchy.

Formally, a NUPN is defined as a 9-tuple $(P, T, \mathrm{Pre}, \mathrm{Post}, m_0, U, u_0, \sqsubseteq, \mathsf{unit})$, where: $(P, T, \mathrm{Pre}, \mathrm{Post}, m_0)$ is a Petri net (as defined in Sect. 1.1); U is a finite, non-empty set such that $U \cap T = U \cap P = \emptyset$ (the elements of U are called *units*); u_0 is an element of U (u_0 is called the *root unit*); $\sqsubseteq$ is a binary relation over U such that $(U, \sqsupseteq)$ is a tree with a single root u_0; unit is a function $P \to U$ such that $\forall u \in U \setminus \{u_0\} \,.\, \exists p \in P \,.\, \mathsf{unit}(p) = u$ (intuitively, $\mathsf{unit}(p) = u$ expresses that unit u directly contains place p). The *height* of a NUPN is the height of its unit tree, not counting the root unit if it contains no place directly (i.e., for each $p \in P$, $\mathsf{unit}(p) \neq u_0$). The *width* of a NUPN is the number of leaf units in its unit tree.

The token game for NUPNs is the same as for Petri nets, meaning that introducing units does not modify the rules for firing transitions and the set of reachable markings.

A fundamental property of NUPNs is the notion of *unit safeness* [Gar19], which generalizes the one-safeness property of Petri nets. Formally, two units u_1 and u_2 are *disjoint* if $(u_1 \not\sqsubseteq u_2)$ and $(u_2 \not\sqsubseteq u_1)$, meaning that both units are neither equal nor contained one in the other. A NUPN is *unit-safe* if and only if its reachable markings only contain pairs of places located into disjoint units, meaning that each unit, or two transitively nested units, may not contain two tokens simultaneously. This property enables logarithmic reductions in the number of bits or Boolean variables needed to represent reachable markings [Gar19].

In practice, the unit-related information, namely $(U, u_0, \sqsubseteq, \mathsf{unit})$, is directly obtained when the NUPN is produced from a higher-level model [Gar19]. For instance, if the NUPN is generated from a process-calculus language such as LOTOS [ISO89] or LNT [GLS17], the unit tree can be deduced from the parallel composition operators present in the source specifications; if the NUPN is generated from a network of automata, the unit tree represents the various automata that execute concurrently; etc.

6.3 Safeness in Token Flow Graphs

In the following, we will focus on safe nets. Fortunately, our reduction rules preserve safeness (see Corollary 6.2). Hence, we do not need to check if (N_2, m_2) is safe when (N_1, m_1) is. The fact that the nets are safe has consequences on configurations.

Lemma 6.1 (Safe Configurations)**.** *Assume that $[\![E']\!]$ is a well-formed TFG for $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$ with (N_1, m_1) a safe Petri net. Then for every total, well-defined configuration c of $[\![E']\!]$ such that $c_{|N_1}$ reachable in (N_1, m_1), and every node v (not in K), we have $c(v) \in \{0, 1\}$.*

Proof. We prove the result by contradiction. Take a total, well-defined configuration c such that $c_{|N_1}$ is reachable in (N_1, m_1) and a node v such that $c(v) > 1$. Since (N_1, m_1) is safe, v does not belong to P_1. We consider two cases: whether some place belongs to P_1 in $\downarrow v$, or not.

First, suppose that a place p of N_1 exists, such as $v \to^\star p$. By Lemma 4.3 (Forward), we can find a well-defined configuration c' of $[\![E']\!]$ such that $c'(p) \geqslant c'(v) = c(v) > 1$ and $c'(w) = c(w)$ for every node w not in $\downarrow v$. By condition (T6) the places of N_2 correspond to the roots of $[\![E']\!]$ (if we forget about constant nodes); the latter implies $c'_{|N_2} = c_{|N_2}$. Therefore, $c'_{|N_2}$ is also reachable in (N_2, m_2).

Second, suppose that $\downarrow v \cap P_1 = \emptyset$, then by condition (T7) v corresponds to some slack variable and there is some node w such that $\downarrow w \cap P_1 \neq \emptyset$ and $w \circ\!\!\to v$. Take some node v' such that $w \circ\!\!\to v'$ and $\downarrow v' \cap P_1 \neq \emptyset$. By Lemma 4.3 (Agglomeration), we can find a well-defined configuration c'' of $[\![E']\!]$ such that $c''(v') = c(v)$. As in the previous case, we can now find some configuration c' and some node p in P_1 such that $v' \to^\star p$ and $c'(p) \geqslant c''(v') = c(v)$.

Then, by Theorem 4.4, $c'_{|N_1}$ is reachable in (N_1, m_1). However, $c'(p) > 1$ is in contradiction with the safeness of (N_1, m_1). □

Corollary 6.2 (Safeness Preservation)**.** *Assume that $[\![E']\!]$ is a well-formed TFG for $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$. If (N_1, m_1) is safe then (N_2, m_2) is safe.*

We base our approach on the fact that we can extend the notion of concurrent places (in a marked net) to the notion of concurrent nodes in a TFG, meaning nodes that can be marked together in a reachable configuration (as defined in Definition 4.6).

By Theorem 4.4, if we take reachable markings in N_2—meaning we fix the values of roots in $[\![E']\!]$—we can find places of N_1 that are marked together by propagating tokens from the roots to the leaves (Lemma 4.3). In our algorithm, next, we show that we can compute the concurrency relation of N_1 by considering two cases: (1) we start with a token in a single root p, with p nondead, and propagate this token forward until we find a configuration with two places in N_1 marked together (which is basically due to some redundant places); or (2) we do the same but placing a token in two separate roots, p_1, p_2, such that $p_1 \parallel p_2$.

6.4 Dimensionality Reduction Algorithm

We assume that $[\![E']\!]$ is a well-formed TFG for the relation $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$. We use symbol $\parallel_2$ for the concurrency relation on (N_2, m_2) and $\parallel_1$ on (N_1, m_1). The set of nodes of $[\![E']\!]$ is P.

We define an algorithm that takes as inputs a well-formed TFG $[\![E']\!]$ plus the concurrency relation $\parallel_2$ on the net (N_2, m_2), and outputs the concurrency relation $\parallel_1$ on (N_1, m_1). Our algorithm computes a *concurrency matrix*, C, that is a symmetric matrix such that $\mathrm{C}[v, w] = 1$

when the nodes v, w can be marked together in a reachable configuration, and 0 otherwise. We prove (Theorem 6.8) that the relation induced by C matches with $\|_1$ on N_1. Our algorithm can be pragmatically interrupted after a given time limit; it then returns a partial relation $\|_2$. Undefined cases are written $\mathrm{C}[v, w] = \bullet$ in matrix C, which is then qualified as *incomplete*.

The complexity of computing the concurrency relation highly depends on the number of places in the net. For this reason, we say that our algorithm performs some sort of "dimensionality reduction" because it allows us to solve a problem in a high-dimension space (the number of places in N_1) by solving it first on a lower dimension space (since N_2 may have far fewer places) and then transporting back the result to the original net. In practice, we compute the concurrency relation on (N_2, m_2) using the tool Cæsar.BDD from the CADP toolbox [BG21; INR], but we can rely on any kind of "oracle" to compute this relation for us. This step is unnecessary when the initial net is fully reducible; in this case, the concurrency relation for N_2 is trivial, and all the roots in $[\![E']\!]$ are constants.

To simplify our notations, we assume that $v \|_2 w$ when v is a constant node in $K(1)$ and w is nondead. On the opposite, $v \#_2 w$ when $v \in K(0)$ or w is dead.

Our algorithm is divided into two main functions, shown in Algorithms 6.1 and 6.2. It also implicitly relies on an auxiliary function that returns the successors $\downarrow x$ for a given node x (we omit the details). In the main function, Matrix (Algorithm 6.1), we iterate over the nondead roots of $[\![E']\!]$ and recursively propagates the information that node v is nondead: the call to Propagate in line 6 of Algorithm 6.1 updates the concurrency matrix C by finding all the concurrent nodes that arise from a unique root v. We can prove that all such cases arise from redundancy arcs originating in $\downarrow v$. More precisely, we prove in Lemma 6.6 that if $v \rightarrow\!\!\!\!\bullet\, w$ holds, then the nodes in the set $\downarrow v \setminus \downarrow w$ are concurrent to all the nodes in $\downarrow w$. This is made explicit in the `for` loop, line 10 of Algorithm 6.2. Next, in the second `for` loop of Matrix (Algorithm 6.1 line 9), we compute the concurrent nodes that arise from two distinct nondead roots (v, w). In this case, we can prove that all the successors of v are concurrent with successors of w: all the pairs in $\downarrow v \times \downarrow w$ are concurrent.

We can perform a cursory analysis of the complexity of our algorithm. We update the matrix by recursively invoking Propagate (Algorithm 6.2), along the edges of $[\![E']\!]$, starting from the roots. (Of course, an immediate optimization consists of marking the visited nodes so that the function Propagate is never invoked twice on the same node. We do not provide the details of this optimization since it has no impact on soundness, completeness, or theoretical complexity.) More precisely, we call Propagate only on the nodes that are nondead in $[\![E']\!]$. Hence, our algorithm performs a number of function calls that is linear in the number of nondead nodes. During each call to Propagate, we may update at most $O(N^2)$ values in C, where N is the number of nodes in $[\![E']\!]$ (see the `for` loop line 10). As a result, the complexity of our algorithm is in $O(N^3)$, given the concurrency relation $\|_2$. This has to be compared with the complexity of building then checking the state space of the net, which is PSPACE. Thus, computing the

Algorithm 6.1 MATRIX($[\![E']\!]$, $\|_2$)

In: $[\![E']\!]$: the TFG structure,
$\|_2$: concurrency relation on (N_2, m_2).
Out: the concurrency matrix C.

```
1:  ;; C is a matrix indexed by P × P.
2:  C ← 0
3:
4:  ;; v (∈ P2) is nondead if and only if v ||2 v holds.
5:  for all v nondead root node in [[E']] do
6:      PROPAGATE([[E']], C, v)
7:
8:  ;; v and w (∈ P2) are concurrent if and only if v ||2 w holds.
9:  for all (v, w) distinct concurrent roots in [[E']] do
10:     for all(v', w') ∈ ↓v × ↓w do
11:         C[v', w'] ← 1
12:         C[w', v'] ← 1
13:
14: return C
```

Algorithm 6.2 PROPAGATE($[\![E']\!]$, C, v)

In: $[\![E']\!]$: the TFG structure,
v: node.
In out: C: the concurrency matrix.
Post: C contains all the concurrency relations induced by knowing that v is nondead.

```
1:  ;; This loop includes C[v, v] ← 1.
2:  for all w ∈ ↓v do
3:      C[v, w] ← 1
4:      C[w, v] ← 1
5:
6:  for all w such that v → w do
7:      PROPAGATE([[E']], C, w)
8:
9:  for all w such that v →• w do
10:     for (v', w') ∈ ((↓v \ ↓w) × ↓w) do
11:         C[v', w'] ← 1
12:         C[w', v'] ← 1
```

concurrency relation $\|_2$ of (N_2, m_2), with a lower dimension, and tracing it back to (N_1, m_1) is quite benefiting.

In practice, our algorithm is efficient, and its execution time is often negligible compared to the other tasks involved when computing the concurrency relation. We give some results on our performances in Sect. 6.9.

6.5 Proof of Correctness

The soundness and completeness proofs of the algorithm rely on the following definition:

Definition 6.2 (Concurrent Nodes)**.** *The concurrency relation of $[\![E']\!]$, denoted $\mathcal{C}$, is the relation between pairs of nodes in $[\![E']\!]$ such that $v \mathcal{C} w$ holds if and only if there is a total, well-defined configuration c where: (1) c is reachable, meaning $c_{|N_2} \in R(N_2, m_2)$; and (2) $c(v) > 0$ and $c(w) > 0$.*

The concurrency relation $\mathcal{C}$ of $[\![E']\!]$ is a generalization of both the concurrency relation $\|_1$ of N_1 and $\|_2$ of N_2: for any pair of places $(p, q) \in P_1^2$, by Theorem 4.4, we have $p \|_1 q$ if and only if $p \mathcal{C} q$. Similarly, for $(p, q) \in P_2^2$: $p \|_2 q$ if and only if $p \mathcal{C} q$. We say in the latter case that p, q are *concurrent roots*. As a result, $\mathcal{C}$ is symmetric, and $v \mathcal{C} v$ means that v is nondead (that is, there is a valuation c with $c(v) > 0$). We can extend this notion to constants: we say that two roots v_1, v_2 are concurrent when $v_1 \mathcal{C} v_2$ holds, and that root v_1 is nondead when we have $v_1 \mathcal{C} v_1$. This includes cases where v_1 or v_2 are in $K(1)$ (constants with value 1).

We prove some properties about the relation $\mathcal{C}$ that are direct corollaries of our token propagation properties. For all the following results, we implicitly assume that $[\![E']\!]$ is a well-formed TFG for the relation $(N_1, m_1) \equiv_{\exists Q.E'} (N_2, m_2)$, that both marked nets are safe, and that $\mathcal{C}$ is the concurrency relation of $[\![E']\!]$.

6.5.1 Checking Nondead Nodes

We start with a property (Lemma 6.3) stating that the successors of a nondead node are also nondead. Lemma 6.4 provides a dual result useful to prove the completeness of our approach; it states that it is enough to explore the nondead roots to find all the nondead nodes.

Lemma 6.3. *If $v \mathcal{C} v$ and $v \to^\star w$ then $w \mathcal{C} w$ and $v \mathcal{C} w$.*

Proof. Assume that $v \mathcal{C} v$. This means that there is a total, well-defined configuration c such that $c(v) > 0$ and $c_{|N_2} \in R(N_2, m_2)$. Take a successor node of v, say $v \to^\star w$. By Lemma 4.3, we can find another reachable configuration c' such that $c'(w) \geqslant c'(v) = c(v)$ and $c'(x) = c(x)$ for all nodes x not in $\downarrow v$. Therefore, we have $w \mathcal{C} w$ and $v \mathcal{C} w$. □

Lemma 6.4. *If $v \mathcal{C} v$ then there is a root v_0 such that $v_0 \mathcal{C} v_0$ and $v_0 \to^\star v$.*

Proof. Assume that $v \mathcal{C} v$. Then there is a total, well-defined configuration c such that $c_{|N_2} \in R(N_2, m_2)$ and $c(v) > 0$. By the backward propagation property of Lemma 4.3, we know that there is a root, say v_0, such that $c(v_0) \geqslant c(v)$ and $v_0 \to^\star v$. Hence, v_0 is nondead in $[\![E']\!]$. □

6.5.2 Checking Concurrent Nodes

We can prove similar results for concurrent nodes instead of nondead ones. We consider the two cases considered by function Matrix: when concurrent nodes are obtained from two concurrent roots (Lemma 6.5); or when they are obtained from a single nondead root (Lemma 6.6), because of redundancy arcs. Finally, Lemma 6.7 provides the associated completeness result.

Lemma 6.5. *Assume that v, w are two nodes in $[\![E']\!]$ such that $v \notin {\downarrow}w$ and $w \notin {\downarrow}v$. If $v \mathcal{C} w$ then $v' \mathcal{C} w'$ for all pairs of nodes $(v', w') \in {\downarrow}v \times {\downarrow}w$.*

Proof. Assume that $v \mathcal{C} w$, $v \notin {\downarrow}w$ and $w \notin {\downarrow}v$. By definition, there exists a total, well-defined configuration c such that $c(v) > 0$, $c(w) > 0$ and $c_{|N_2} \in R(N_2, m_2)$.

Take a successor v' in ${\downarrow}v$, by applying the token propagation from Lemma 4.3 we can construct a total, well-defined configuration c' of $[\![E']\!]$ such that $c'(v') \geqslant c'(v) = c(v)$ and $c'(x) = c(x)$ for any node x not in ${\downarrow}v$. Hence, $c'(w) = c(w) > 0$.

We can use the token propagation property again on c'. This gives a total, well-defined configuration c'' such that $c''(w') \geqslant c''(w) = c'(w) = c(w)$ and $c''(x) = c'(x)$ for any node x not in ${\downarrow}w$.

We still have to prove $v' \notin {\downarrow}w$ (which would be immediate in a tree structure but requires extra proof in our DAG structure). Then, we will be able to conclude by observing that it implies $c''(v') = c'(v') \geqslant c(v)$ and therefore $v' \mathcal{C} w'$ as needed.

We prove $v' \notin {\downarrow}w$ by contradiction. Indeed, suppose that $v' \in {\downarrow}w$. Hence, ${\downarrow}v \cap {\downarrow}w \neq \emptyset$. Moreover, since $[\![E']\!]$ is a well-formed TFG, there must exist (condition (T3)) three nodes p, q, r such that $X \twoheadrightarrow r$, $p \in {\downarrow}v \cap X$ and $q \in {\downarrow}w \cap X$. Like in the proof of Lemma 4.3, we can propagate the tokens contained in v, w to p, q, and obtain $c''(r) > 1$ from (CEq), which contradicts our assumption that the nets are safe. □

Lemma 6.6. *If $v \mathcal{C} v$ and $v \twoheadrightarrow w$ then $v' \mathcal{C} w'$ for every pair of nodes (v', w') such that $v' \in ({\downarrow}v \setminus {\downarrow}w)$ and $w' \in {\downarrow}w$.*

Proof. Assume that $v \mathcal{C} v$ and $v \rightarrow\!\!\bullet\, w$. By definition, there is a total, well-defined configuration c such that $c(v) > 0$ and $c_{|N_2} \in R(N_2, m_2)$. Furthermore, by $v \rightarrow\!\!\bullet\, w$ and condition (CEq), we have $c(w) > 0$.

Take w' in $\downarrow w$. From Lemma 4.3 we can find a total, well-defined configuration c' such that $c'(w') \geqslant c'(w) = c(w) > 0$ and $c'(x) = c(x)$ for any node x not in $\downarrow w$. Since v is not in $\downarrow w$ we have $c'(v) = c(v)$. Likewise, places from N_2 are roots and therefore cannot be in $\downarrow w$. So we have $c'_{|N_2} \equiv c_{|N_2}$, which means $c'_{|N_2}$ is reachable in (N_2, m_2). At this point, we have $v \mathcal{C} w'$.

Now, consider $v' \in \downarrow v \setminus \downarrow w$ with $v' \neq v$ (the expected result already holds if $v' = v$). Necessarily, there exists $v_0 \neq w$ such that $v \rightarrow v_0$ and $v_0 \rightarrow^\star v'$. We can use the forward propagation in Lemma 4.3 on c' to find a total, well-defined configuration c'' such that $c''(v_0) \geqslant c''(v) = c'(v)$ and $c''(x) = c(x)$ for all nodes x not in $\downarrow v$, and so, $c''_{|N_2}$ is reachable in (N_2, m_2). Since configuration c'' is well-defined, we have (condition CEq) that $c''(w) \geqslant c''(v)$. We consider three cases.

- Either $v_0 \notin \downarrow w$ and $w \notin \downarrow v_0$, and we conclude by Lemma 6.5 that $v' \mathcal{C} w'$ holds for every node $v' \in \downarrow v_0$.
- Or $v_0 \in \downarrow w$: this case cannot happen since by hypothesis $v' \notin \downarrow w$ and $v_0 \rightarrow^\star v'$.
- Or $w \in \downarrow v_0$: by applying the same proof as the one at the end of Lemma 6.5, we can show that this case leads to a non-safe marking, which is therefore excluded.

As a result, we have $v' \mathcal{C} w'$ for all $v' \in \downarrow v \setminus \downarrow w$ and all $w' \in \downarrow w$. □

Lemma 6.7. *If $v \mathcal{C} w$ holds with $v \notin \downarrow w$ and $w \notin \downarrow v$ then one of the following two conditions is true.*

(Redundancy) *There is a nondead node v_0 such that $v_0 \rightarrow\!\!\bullet\, w_0$ and either (v, w) or (w, v) are in $(\downarrow v_0 \setminus \downarrow w_0) \times \downarrow w_0$.*

(Distinct) *There is a pair of distinct roots (v_0, w_0) such that $v_0 \mathcal{C} w_0$ with $v \in \downarrow v_0$ and $w \in \downarrow w_0$.*

Proof. Assume that $v \mathcal{C} w$. Then there is a total, well-defined configuration c such that $c_{|N_2} \in R(N_2, m_2)$ and $c(v) = c(w) = 1$ (the nets are safe). By the backward-propagation property in Lemma 4.3 there exists two roots v_0 and w_0 such that $c(v_0) = c(w_0) = 1$ with $v \in \downarrow v_0$ and $w \in \downarrow w_0$. We need to consider two cases.

- Either $v_0 \neq w_0$, that is condition (Distinct).
- Or we have $v_0 = w_0$. We prove that there must be a node v_1 such that $v_0 \rightarrow^\star v_1$ and $v_1 \rightarrow\!\!\bullet\, w_1$ with either (v, w) or (w, v) in $(\downarrow v_1 \setminus \downarrow w_1) \times \downarrow w_1$. We prove this result by contradiction. Indeed, if no such node exists, then both v and w can be reached from v_0 by following only edges in A (agglomeration arcs). Consider $v_0 \circ\!\!\rightarrow Y$, there are two nodes v', w' in Y such that $v \in \downarrow v'$ and $w \in \downarrow w'$. Since c is well-defined, from (CEq)

either $c(v') = 0$ or $c(w') = 0$. Take $c(v') = 0$ and the agglomeration path from v' to v, as $v' \circ\!\!\rightarrow a_0 \circ\!\!\rightarrow \cdots \circ\!\!\rightarrow a_n = v$ with $n \in \mathbb{N}$. By induction on this path, we necessarily have $c(a_i) = 0$ for all $i \in 0..n$, since c is well-defined and a node can only have one parent (condition (T3)). Hence, $c(v') = 0$ that contradicts $v \mathcal{C} w$.

□

6.5.3 Soundness and Completeness

We now prove that the algorithm is sound and complete.

Theorem 6.8 (Algorithm 6.1 is Sound and Complete)**.** *If* C *is the matrix returned by a call to* Matrix$([\![E']\!], \|)$*, with* $\|$ *the concurrency relation between roots of* $[\![E']\!]$ *(meaning* N_2 *and constants), then for all nodes* v, w *we have* $v \mathcal{C} w$ *if and only if* $\mathrm{C}[v, w] = 1$*.*

Proof. First, let us remark that the call to Matrix$([\![E']\!], \|)$ always terminates since the only recursion (Propagate) follows the DAG structure. We divide the proof into two cases: first, we show that the computation of nondead nodes (the diagonal of C and the nondead nodes of $\mathcal{C}$) is sound and complete. Next, we prove soundness and completeness for pairs of distinct nodes.

Nondead Places, Diagonal of C:

(Completeness) If $v \mathcal{C} v$ holds for some node v, then, by Lemma 6.4, there exists a nondead root v_0 with $v \in {\downarrow}v_0$. Hence, Algorithm 6.1 invokes Propagate line 6 with v_0. Then, all nodes in ${\downarrow}v_0$ are recursively visited by line 7 in Algorithm 6.2 including v. As a consequence, $\mathrm{C}[v, v]$ is set to 1 line 3 (and remains equal to 1 until the end of the algorithm, since no line of the algorithm sets values of C to 0 after the initialization line 2). This concludes the completeness part for nondead places.

(Soundness) Conversely, assume that $\mathrm{C}[v, v] = 1$ for some node v. We consider three subcases.

- $\mathrm{C}[v, v]$ was set line 11 or 12 of Algorithm 6.1: this means that there exist two distinct concurrent roots v_0 and w_0 such that $v \in {\downarrow}v_0 \cap {\downarrow}w_0$. Hence, v_0 is nondead (as well as w_0). This implies that v is nondead by Lemma 6.3.
- $\mathrm{C}[v, v]$ was set line 3 of Algorithm 6.2: the `for` loops line 5 of Algorithm 6.1 and line 6 of Algorithm 6.2 ensure that Propagate is only invoked on successors of nondead roots of $[\![E']\!]$. Hence, v belongs to ${\downarrow}v_0$ for some nondead root v_0, and thus $v \mathcal{C} v$ holds by Lemma 6.3.
- $\mathrm{C}[v, v]$ was set line 11 or 12 of Algorithm 6.2: this subcase is not possible, since these lines only consider pairs (v', w') of distinct nodes.

To conclude this first case, the algorithm is sound and complete for nondead places and the diagonal of C.

Concurrent Places:

(Completeness) We assume that $v \mathcal{C} w$ holds for two distinct nodes v and w. This implies that both v and w are nondead, that is, $v \mathcal{C} v$ and $w \mathcal{C} w$. If we have $v \in {\downarrow}w$, then $\mathrm{C}[v, w]$ is set to 1 line 3 or 4 of Algorithm 6.2, and similarly if $w \in {\downarrow}v$, which is the expected result. Hence, we now assume that $v \notin {\downarrow}w$ and $w \notin {\downarrow}w$, and thus Lemma 6.7 applies. We consider the two cases of the lemma.

- (Redundancy): then, $\mathrm{C}[v, w]$ is set to 1 line 11 or 12 of Algorithm 6.2.
- (Distinct): then $\mathrm{C}[v, w]$ is set to 1 line 11 or 12 of Algorithm 6.1.

This concludes the completeness of the algorithm for concurrent places.

(Soundness) We assume that $\mathrm{C}[v, w] = 1$ for some distinct nodes v, w. We consider three subcases.

- $\mathrm{C}[v, w]$ was set line 11 or 12 of Algorithm 6.1: we conclude by Lemma 6.5 that $v \mathcal{C} w$ holds.
- $\mathrm{C}[v, w]$ was set line 3 or 4 of Algorithm 6.2: we conclude by Lemma 6.3.
- $\mathrm{C}[v, w]$ was set line 11 or 12 of Algorithm 6.2: we conclude by Lemma 6.6.

This concludes the soundness of the algorithm for concurrent places.

As a result, the algorithm is sound and complete for nondead places and concurrent places. □

6.6 Running Example

We now propose to illustrate the previous results about the computation of dead and concurrent places on, a safe version, of the equivalence statement from Fig. 4.1 depicted in Fig. 6.1.

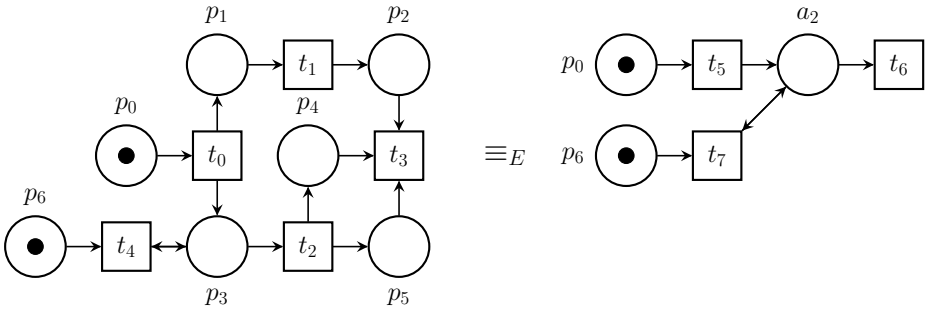


Fig. 6.1 An example of Petri net, (N_1, m_1) (left), and one of its polyhedral reductions, (N_2, m_2) (right), with $E \triangleq \exists a_1 \,.\, (p_5 = p_4) \wedge (a_1 = p_1 + p_2) \wedge (a_2 = p_3 + p_4) \wedge (a_1 = a_2)$.

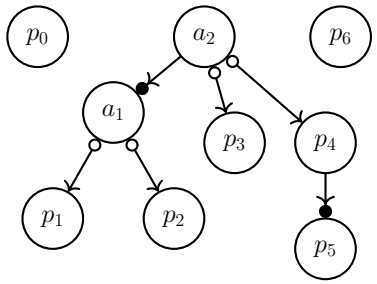


Fig. 6.2 TFG corresponding to the polyhedral equivalence in Fig. 6.1.

The concurrency matrix $C_{(N_2,m_2)}$ of the reduced net, and the incomplete matrix $C_{(N_1,m_1)}$ of the initial net are:

$$C_{(N_2,m_2)} = \begin{array}{c} \\ a_2 \\ p_0 \\ p_6 \end{array} \begin{array}{c} \begin{array}{ccc} a_2 & p_0 & p_6 \end{array} \\ \begin{bmatrix} 1 & & \\ 0 & 1 & \\ 1 & 1 & 1 \end{bmatrix} \end{array} \qquad C_{(N_1,m_1)} = \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} p_0 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{array} \\ \begin{bmatrix} 1 & & & & & & \\ \bullet & \bullet & & & & & \\ \bullet & \bullet & \bullet & & & & \\ \bullet & \bullet & \bullet & \bullet & & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \\ 1 & \bullet & \bullet & \bullet & \bullet & \bullet & 1 \end{bmatrix} \end{array}$$

Now, by leveraging the TFG in Fig. 6.2 corresponding to the polyhedral reduction of Fig. 6.1, we iteratively trace back the concurrency relation of the initial net (N_1, m_1).

1. Lemma 6.3: we have place a_2, in the reduced net N_2, nondead (because we can fire t_5). As a consequence, all the successors' nodes of a_2 in the TFG (that are also placed in N_1) must also be nondead, meaning $C[p_i, p_i] = 1$ for all i in 1..5.

$$C_{(N_1,m_1)} = \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} p_0 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{array} \\ \begin{bmatrix} 1 & & & & & & \\ \bullet & 1 & & & & & \\ \bullet & \bullet & 1 & & & & \\ \bullet & \bullet & \bullet & 1 & & & \\ \bullet & \bullet & \bullet & \bullet & 1 & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & 1 & \\ 1 & \bullet & \bullet & \bullet & \bullet & \bullet & 1 \end{bmatrix} \end{array}$$

2. Lemma 6.6: from the fact that a_2 in nondead, we can deduce that p_4 is concurrent to p_5 (meaning $C[p_4, p_5] = 1$), because of the redundancy $p_5 = p_4$, and p_1, p_2 are concurrent to p_3, p_4, p_5.

$$C_{(N_1,m_1)} = \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} p_0 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{array} \\ \left[\begin{array}{ccccccc} 1 & & & & & & \\ \bullet & 1 & & & & & \\ \bullet & \bullet & 1 & & & & \\ \bullet & 1 & 1 & 1 & & & \\ \bullet & 1 & 1 & \bullet & 1 & & \\ \bullet & 1 & 1 & \bullet & 1 & 1 & \\ 1 & \bullet & \bullet & \bullet & \bullet & \bullet & 1 \end{array}\right] \end{array}$$

3. Lemma 6.5: we have a_2 concurrent to p_6. Then, all the successor nodes of a_2 in the TFG (of N_1) are concurrent to p_6, i.e., $C[p_i, p_6] = 1$ for all i in 1..5.

$$C_{(N_1,m_1)} = \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} p_0 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{array} \\ \left[\begin{array}{ccccccc} 1 & & & & & & \\ \bullet & 1 & & & & & \\ \bullet & \bullet & 1 & & & & \\ \bullet & 1 & 1 & 1 & & & \\ \bullet & 1 & 1 & \bullet & 1 & & \\ \bullet & 1 & 1 & \bullet & 1 & 1 & \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array}$$

4. Lemma 6.7: we propagated all nondead and concurrent places, and then we can set all the unknown relations to 0 (nonconcurrent).

$$C_{(N_1,m_1)} = \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} p_0 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{array} \\ \left[\begin{array}{ccccccc} 1 & & & & & & \\ 0 & 1 & & & & & \\ 0 & 0 & 1 & & & & \\ 0 & 1 & 1 & 1 & & & \\ 0 & 1 & 1 & 0 & 1 & & \\ 0 & 1 & 1 & 0 & 1 & 1 & \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array}$$

6.7 Extensions to Incomplete Concurrency Relations

We only write 1s into the concurrency matrix C with our approach. This is enough since we know relation $\|_2$ exactly and, in this case, relation $\|_1$ must also be complete (we can have only 0s or 1s in C). This is made clear by the fact that C is initialized with 0s everywhere. We can extend our algorithm to support the case where we only have partial knowledge of $\|_2$.

This is achieved by initializing C with the special value • (undefined) and adding rules that let us "propagate 0s" on the TFG in the same way that our total algorithm only propagates 1s. For example, we know that if $\mathrm{C}[v, w] = 0$ (v, w are nonconcurrent) and $v \circ\!\!\to v'$ (we know that always $c(v) \geqslant c(v')$ on reachable configurations) then certainly $\mathrm{C}[v', w] = 0$. Likewise, we can prove that the following rule for propagating "dead nodes" is sound: if $X \to\!\!\bullet\, v$ and $\mathrm{C}[w, w] = 0$ (node w is dead) for all $w \in X$ then $\mathrm{C}[v, v] = 0$.

Partial knowledge of the concurrency relation can be useful. Indeed, many use cases can deal with partial knowledge or only rely on the nonconcurrency relation (a 0 on the concurrency matrix). This is the case, for instance, when computing a NUPN decomposition, where it is always safe to replace a • with a 1. It also means that knowing that two places are nonconcurrent is often more valuable than knowing that they are concurrent; 0s are more informative than 1s.

Using this idea, we have implemented an extension of our algorithm for the case of incomplete matrices, and we report some results obtained with it. It is slightly more involved than the complete case and is based on a collection of six additional axioms.

In the following, we use the notation $v \,\bar{\mathcal{C}}\, w$ to say $\neg(v \mathrm{C} w)$, meaning v, w are nonconcurrent according to C. With our notations, $v \,\bar{\mathcal{C}}\, v$ means that v is dead: there is no well-defined, reachable configuration c with $c(v) > 0$.

6.7.1 Propagation of Dead Nodes

We prove that a dead node, v, is necessarily nonconcurrent to all the other nodes. Also, if all a node's "direct successors" are dead, then also is the node.

Lemma 6.9. *Assume that v is a node in $[\![E']\!]$. If $v \,\bar{\mathcal{C}}\, v$ then for all nodes w in $[\![E']\!]$ we have $v \,\bar{\mathcal{C}}\, w$.*

Proof. Assume that $v \,\bar{\mathcal{C}}\, v$. Then for any total, well-defined configuration c such that $c_{|N_2}$ is reachable in (N_2, m_2) we have $c(v) = 0$. By definition of the concurrency relation $\mathcal{C}$, v cannot be concurrent to any node. □

Lemma 6.10. *Assume that v is a node in $[\![E']\!]$ such that $v \circ\!\!\to X$ or $X \to\!\!\bullet\, v$. Then $v \,\bar{\mathcal{C}}\, v$ if and only if $w \,\bar{\mathcal{C}}\, w$ for all nodes w in X.*

Proof. We prove by contradiction both directions.

Suppose that $v \,\bar{\mathcal{C}}\, v$ and take $w \in X$ such that $w \,\mathcal{C}\, w$. Then there is a total, well-defined configuration c such that $c(w) > 0$. Necessarily, since $v \,\bar{\mathcal{C}}\, v$ we have $c(v) = 0$, which contradicts (CEq).

Next, suppose that $v \mathcal{C} v$ and $w \bar{\mathcal{C}} w$ for every node $w \in X$. Then there is a total, well-defined configuration c such that $c(v) > 0$. Necessarily, for all nodes $w \in X$, we have $c(w) = 0$, which contradicts (CEq). □

These properties imply the soundness of the following three axioms:

1. If $\mathrm{C}[v, v] = 0$ then $\mathrm{C}[v, w] = 0$ for all node w in $[\![E']\!]$.
2. If $v \circ\!\!\to X$ or $X \to\!\!\bullet\, v$ and $\mathrm{C}[w, w] = 0$ for all nodes $w \in X$ then $\mathrm{C}[v, v] = 0$.
3. If $v \circ\!\!\to X$ or $X \to\!\!\bullet\, v$ and $\mathrm{C}[v, v] = 0$ then $\mathrm{C}[w, w] = 0$ for all nodes $w \in X$.

6.7.2 Nonconcurrency Between Siblings

We prove that direct successors of a node are nonconcurrent from each other (in the case of safe nets). This is a consequence of the fact that $c(v) = c(w) + c(w') + \ldots$ and $c(v) \leqslant 1$ implies that at most one of $c(w)$ and $c(w')$ can be equal to 1 when the configuration is fixed.

Lemma 6.11. *Assume that v is a node in $[\![E']\!]$ such that $v \circ\!\!\to X$ or $X \to\!\!\bullet\, v$. For every pair of nodes w, w' in X we have $w \neq w'$ implies $w \bar{C} w'$.*

Proof. The proof is by contradiction. Take a pair of distinct nodes w, w' in X and suppose that $w \mathcal{C} w'$. Then there exists a total, well-defined configuration c such that $c(w) = 1$ and $c(w') = 1$, with $c_{|N_2}$ reachable in (N_2, m_2). Since c must satisfy (CEq), we have $c(v) \geqslant 2$, which contradicts the fact that our nets are safe; see Lemma 6.1. □

This property implies the soundness of the following axiom:

4. If $v \circ\!\!\to X$ or $X \to\!\!\bullet\, v$ then $\mathrm{C}[w, w'] = 0$ for all pairs of nodes $w, w' \in X$ such that $w \neq w'$.

6.7.3 Heredity and Nonconcurrency

We prove that if v and v' are nonconcurrent, then v' must be nonconcurrent from all the direct successors of v (and reciprocally). This is basically a consequence of the fact that $c(v) = c(w) + \ldots$ and $c(v) + c(v') \leqslant 1$ implies that $c(w) + c(v') \leqslant 1$.

Lemma 6.12. *Assume that v is a node in $[\![E']\!]$ such that $v \circ\!\!\to X$ or $X \to\!\!\bullet\, v$. Then for every node v' such that $v \bar{\mathcal{C}} v'$ we also have $w \bar{\mathcal{C}} v'$ for every node w in X. Conversely, if $w \bar{\mathcal{C}} v'$ for every node w in X then $v \bar{\mathcal{C}} v'$.*

Proof. We prove by contradiction each property separately.

Suppose that $v \bar{\mathcal{C}} v'$ and take $w \in X$ such that $w \mathcal{C} v'$. Then there is a total, well-defined configuration c such that $c(w) > 0$ and $c(v') > 0$. Necessarily, since $v \bar{\mathcal{C}} v'$ we must have $c(v) = 0$ or $c(v') = 0$. We already know that $c(v') > 0$, so $c(v) = 0$, which contradicts (CEq) since $w \in X$.

Next, suppose that $w \bar{\mathcal{C}} v'$ for all nodes $w \in X$, and that we have $v \mathcal{C} v'$. Then there is a total, well-defined configuration c such that $c(v) > 0$ and $c(v') > 0$. Necessarily, for all nodes $w \in X$ we have $c(w) = 0$ or $c(v') = 0$. We already know that $c(v') > 0$, so $c(w) = 0$ for all nodes $w \in X$, which also contradicts (CEq). □

These properties imply the soundness of the following two axioms:

5. If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[w, v'] = 0$ for all nodes $w \in X$ then $\mathrm{C}[v, v'] = 0$.
6. If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[v, v'] = 0$ for all nodes w in X then $\mathrm{C}[w, v'] = 0$

This concludes the set of axioms that are implemented in Kong, for computing nonconcurrent and dead places, in the case where the concurrency relation of the reduced net is partial.

6.8 Transposing Nested-Unit Petri Nets

With the concurrent places problem, an interesting approach is to transpose the unit-related information (if unit-safe) from the initial net to the reduced one. This is particularly valuable when the initial net admits a NUPN decomposition generated from LOTOS, in which case, this "free" concurrency information can be used by standard model checkers when checking reachability properties (such as with the approach developed in Chapter 5).

We assume that the net N_1 is associated with some unit-related information $(U, u_0, \sqsubseteq, \mathsf{unit}_1)$, such that the NUPN is unit-safe. We prove that every place p of the reduced net N_2 can be associated with the unit of any successors of p in $[\![E']\!]$ by preserving unit-safeness.

Theorem 6.13 (Unit-Safe Transposition)**.** *Their exists some relation* unit_2 *such that* $\forall p_2 \in P_2 \;.\; \exists p_1 \in P_1 \;.\; \mathsf{unit}_2(p_2) = \mathsf{unit}_1(p_1) \wedge p_1 \in {\downarrow}p_2 \cap P_1$*. And if* (N_1, m_1) *associated to* $(U, u_0, \sqsubseteq, \mathsf{unit}_1)$ *is unit-safe then the same holds for* (N_2, m_2) *with* $(U, u_0, \sqsubseteq, \mathsf{unit}_2)$*.*

Proof. We prove the results in two steps.

First, we prove that some relation unit_2 exists, satisfying the construction assumption. That is for all places p_2 in P_2 there exists some place p_1 in P_1 such that $p_1 \in {\downarrow}p_2$. This is entailed by condition (T6) of Definition 4.4.

Now, we prove that the obtained unit-related information satisfies the unit-safeness condition. That is, for all reachable markings m'_2 in $R(N_2, m_2)$, if $m'_2(p_2) = m'_2(p'_2) = 1$ for two disjoint places p_2, p'_2 then $\mathsf{unit}_2(p_2)$ and $\mathsf{unit}_2(p'_2)$ are disjoint (meaning $\mathsf{unit}_2(p_2) \not\sqsubseteq \mathsf{unit}_2(p'_2)$ and $\mathsf{unit}_2(p'_2) \not\sqsubseteq \mathsf{unit}_2(p_2)$). We prove the result by contradiction. Suppose that p_2, p'_2 is a pair of disjoint places from P_2 such that there is some reachable marking m'_2 in $R(N_2, m_2)$ for which $m'_2(p_2) = m'_2(p'_2) = 1$. Also, suppose that either $\mathsf{unit}_2(p_2) \sqsubseteq \mathsf{unit}_2(p'_2)$ or $\mathsf{unit}_2(p'_2) \sqsubseteq \mathsf{unit}_2(p_2)$. We can consider the case where $\mathsf{unit}_2(p_2) \sqsubseteq \mathsf{unit}_2(p'_2)$ (the other one is similar). By construction of unit_2, there is a pair of places p_1, p'_1 from P_1, such that $p_1 \in {\downarrow}p_2$, $p'_1 \in {\downarrow}p'_2$, $\mathsf{unit}_2(p_2) = \mathsf{unit}_1(p_1)$, and $\mathsf{unit}_2(p'_2) = \mathsf{unit}_1(p'_1)$. Since $\mathsf{unit}_1(p_1) \sqsubseteq \mathsf{unit}_1(p'_1)$ we

have $p_1 \neq p'_1$. The contradiction arises from Lemma 6.5, that entails the existence of a reachable marking m'_1 in $R(N_1, m_1)$ such that $m'_1(p_1) = m'_1(p'_1) = 1$. □

Since the decomposition problem usually admits multiplies solutions, we can elaborate some heuristics to select, for each place p_2 of the reduced net N_2, the unit among the ones of its successors. For example, we chose to maximize the number of places per unit.

6.9 Experimental Results

We have implemented the tool Kong—for Koncurrent places Grinder—, that is in charge of performing the "inverse transforms" that we described in Sect. 6.4.

6.9.1 Toolchain Description

We describe our toolchain in Fig. 6.3, which is the second toolchain of the tool after the one depicted in Fig. 4.6. After computing a polyhedral reduction with Reduce, we compute the concurrency matrix of the reduced net (N_2, m_2) using Cæsar.BDD, which is part of the CADP toolbox [BG21; INR]. Our experimental results have been computed with version v3.7 of Cæsar.BDD, part of CADP version 2023-h "Aachen", published in August 2023. The tool Kong takes this concurrency relation, denoted $\|_2$, and the reduction system, E, then reconstructs the concurrency relation on the initial net.

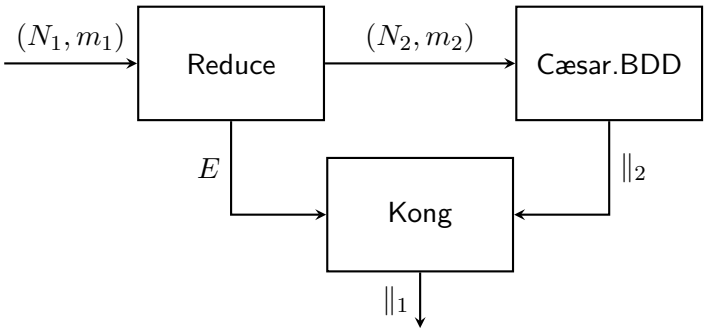


Fig. 6.3 Toolchain of the concurrency acceleration algorithm.

To compute the concurrent places, Cæsar.BDD uses dedicated data structures and implements four methods, which are detailed in [BG21] and used in combination:

1. *Marking graph exploration* performs a forward traversal of the state space, starting from the initial marking. The visited markings are stored symbolically using BDDs, as implemented in the Cudd library. The user can bound the exploration by setting an environment variable to a maximum number of seconds or setting an environment variable to a maximum depth. Once the exploration terminates, the BDD containing all visited markings is queried repeatedly to decide whether a given pair of places belongs to at least one visited marking. The concurrency matrix is complete if the exploration is complete;

otherwise, only a subset of concurrent pairs of places can be inferred from the visited markings.

2. *Structural rules* are a collection of propositions that enable one to conclude that certain pairs of places are concurrent (or nonconcurrent) by examining only their local context. In particular, if the net is a unit-safe NUPN, this information is exploited to conclude that two places belonging to the same unit or two nested units are nonconcurrent. Structural rules are applied repeatedly until saturation.
3. *Quadratic under-approximation* explores an abstraction of the marking graph by approximating a reachable marking m by the set of all pairs of places having a token in m. This is an under-approximation because the algorithm may miss exploring certain pairs of places that are reachable and concurrent. The exploration progresses forward, starting from the initial marking (or, better, from all pairs of places already known to be concurrent), and produces a subset of concurrent pairs of places.
4. *Quadratic over-approximation* also does a forward exploration of the marking graph, again abstracted away using a set of pairs of places, but performs (improving the prior approach of [KE96]) an over-approximation instead of an under-approximation. Indeed, the algorithm explores all markings that it assumes to be potentially reachable because all the pairs of places in each of these markings are potentially concurrent. If the exploration is completed, it produces a subset of nonconcurrent pairs of places.

Cæsar.BDD applies these four complementary methods in sequence, in order 1-2-3-4. The execution may terminate earlier once the concurrency matrix no longer contains unknown values.

6.9.2 Distribution of Reduction Ratios for Safe Nets

Our benchmark is built from the 685 instances detected as safe during the 2023 edition of the MCC by the tools competing in the *onesafe* category.

The observations on the distribution of reduction ratios made across all instances in Sect. 4.6.2 still hold for safe nets (see Fig. 6.4). There is a high number of models whose size is more than halved with our approach (about 30% of the instances have a ratio $r \geqslant 0.5$), with approximately half of the instances that can be reduced by a ratio of 30% or more. In Fig. 6.4, we consider two values for the reduction ratio: one for reductions leading to a well-formed TFG (in light orange), the other for the best possible reduction with Reduce (in dark blue).

As a reminder, we mostly lose the ability to simplify some instances of "partial" marking graphs that could be reduced using inhibitor arcs or weights on the arcs (two features not supported by Cæsar.BDD).

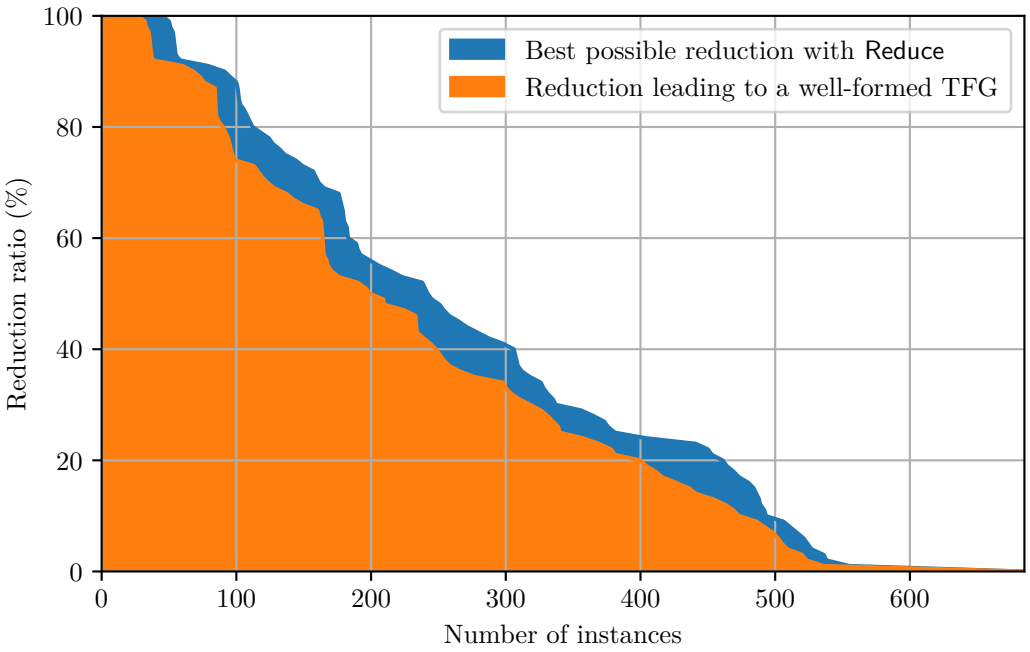


Fig. 6.4 Distribution of reduction ratios over all the safe instances in the MCC.

We evaluated the performance of Kong on the 536 instances of safe Petri nets with a reduction ratio greater than 1%. We ran Kong and Cæsar.BDD on each of those instances in two main modes: first with a time limit of 30 min to compare the number of fully solved instances (when the tool computes a complete concurrency matrix); following with a timeout of 60 s to compare the number of values (the filling ratios) computed in the partial matrices. Computation of a partial concurrency matrix with Cæsar.BDD is done in two phases: a "BDD exploration" phase that the user can stop and a post-processing phase that cannot be stopped. In practice, the execution time on the initial net is often longer than with the reduced one: the mean computation time for Cæsar.BDD is about 319 s and less than 119 s for Kong. In each test, we compared the output of Kong with the values obtained on the initial net with Cæsar.BDD, and achieved 100% reliability.

Next, we give details about the results obtained from our experiments and analyze the impact of using reductions.

6.9.3 Impact on Fully Computed Concurrency Matrices

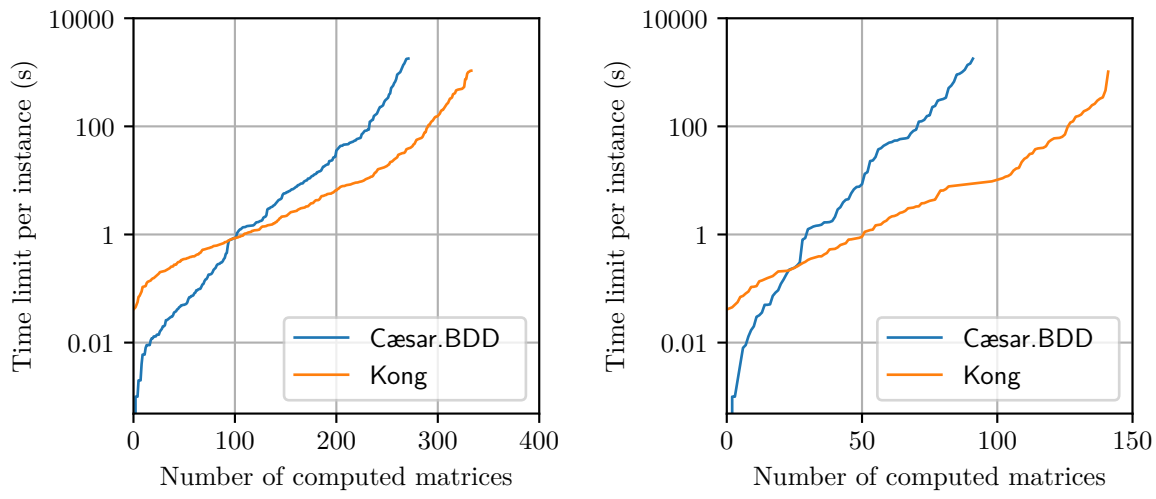
Our subsequent results are for the computation of complete matrices, with a timeout of 30 min. We give the number of computed instances in Table 6.1. We split the results along three different categories of instances, *Low*/*Fair*/*High*/*Full*, associated with different ratio ranges. We observe that we can compute more results with reductions than without (+22%). As could

be expected, the gain is more significant in category *High* (+57%), but it is still significant with the *Fair* instances (+20%).

Reduction Ratio (r)		# Test Cases	# Computed Matrices		
			Kong	Cæsar.BDD	
Low	$r \in [0.01, 0.25[$	196	116	117	×0.99
Fair	$r \in [0.25, 0.5[$	141	76	63	×1.2
High	$r \in [0.5, 1]$	172	115	73	×1.57
Full	$r = 1$	27	27	19	×1.42
Total	$r \in [0.01, 1]$	536	334	272	×1.22

Table 6.1 Number of concurrency matrices computed in 30 min w/wo reduction.

Like in the previous case, we study the speed-up obtained with Kong using charts that compare the time needed to compute a given number of instances; see Fig. 6.5.



(a) $r \in [0.01, 1]$ (b) $r \in [0.5, 1]$

Fig. 6.5 Number of computed concurrency matrices given the query time limit for: (a) all instances, $r \in [0.01, 1]$ and (b) instances with $r \in [0.5, 1]$.

6.9.4 Impact on Partial Matrices

We can also compare the "accuracy" of our approach when we have incomplete results. To this end, we compute the concurrency relation with a timeout of 60 s on Cæsar.BDD. We compare the *filling ratio* obtained with and without reductions. For a net with n places, this ratio is given by the formula $2\,|\mathrm{C}|/(n^2+n)$, where $|\mathrm{C}|$ is the number of 0s and 1s in the matrix.

We display our results using a scatter plot with a linear scale; see Fig. 6.6. We observe that almost all the data points are on one side of the diagonal, meaning in this case that reductions increase the number of computed values, with many examples (top line of the plot) where we can compute the complete relation in 60 s only using reductions. The graphic does not discriminate between the number of 1s and 0s. However, we obtain similar good results when considering the filling ratio for only the concurrent places (the 1s) or the nonconcurrent places (the 0s).

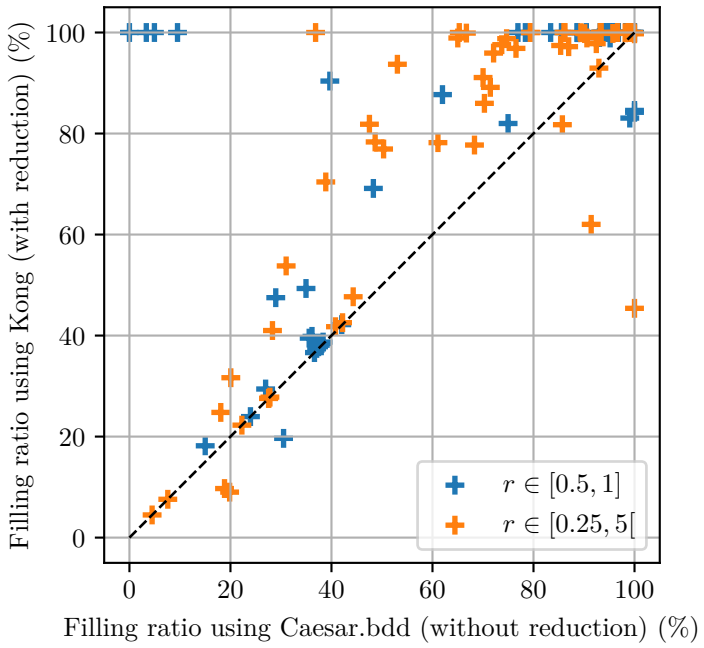


Fig. 6.6 Comparing the filling ratio for partial matrices with Kong (y-axis) and Cæsar.BDD (x-axis) for instances with $r \in [0.25, 0.5[$ (light orange) and $r \in [0.5, 1]$ (dark blue). (Computations done with a BDD exploration timeout of 60 s.)

6.10 Discussion

The main result of our work is a new approach for computing the *concurrency relation* of a Petri net. This problem has practical applications, for instance, because of its use for decomposing a Petri net into the product of concurrent processes [BGP20; Gar19]. It also provides an interesting example of reachability property that nicely extends the notion of *dead places*, meaning places that can never be reached in an execution. These problems raise difficult technical challenges and provide an opportunity to test and improve new model checking techniques [Gar20].

Several works address the problem of finding or characterizing the concurrent places of a Petri net. The main motivation is that the concurrency relation characterizes the sub-parts in a

net that can be simultaneously active. Therefore, it plays a valuable role when decomposing a net into a collection of independent components. This is the case in [WWJ19], where the authors draw a connection between concurrent places and the presence of "sequential modules" (state machines). Another example is the decomposition of nets into unit-safe NUPNs (Nested-Unit Petri Nets) [Gar19; BGP20], for which the computation of the concurrency relation is one of the main bottlenecks.

We know only a couple of tools that support the computation of the concurrency relation. A recent tool is part of the Hippo platform [WWJ19], available online. Our reference tool in this work is Cæsar.BDD, from the CADP toolbox [BG21; INR]. It supports the computation of a partial relation and can output the "concurrency matrix" of a net using a specific, compressed, textual format [Gar20]. We adopt the same format since we use Cæsar.BDD to compute the concurrency relation on the residual net, N_2, and as a yardstick in our benchmarks.

We propose (as in Chapters 4 and 5) a new method to transpose the computation of reachability problems from an initial "high-dimensionality" domain (the set of places in the initial net) into a smaller one (the set of places in the reduced net). Likewise, we show how to use the TFGs to accelerate the computation of the concurrency relation, both in the complete and partial cases.

We have several ideas on how to apply TFGs to other problems and how to extend them. A natural application would be for model counting (the original goal in [BLD18]), where the TFG could lead to new algorithms for counting the number of (integer) solutions in the systems of linear equations we manage. For future work, we would like to answer even more difficult questions, such as proofs of *generalized mutual exclusion constraints* [GDS92], that require checking invariants involving weighted sums over the marking of places of the form $\sum_{p \in P} w_p.m(p)$. Another possible application is the *max-marking* problem, which means finding the maximum of the expression $\sum_{p \in P} m(p)$ over all reachable markings. This amounts to finding the maximum number of places that can be marked together on safe nets. We can easily adapt our algorithm to compute this value and even compute the result when the net is unsafe.

We can even manage a more general problem related to the notion of *max-concurrent* sets of places. We say that the set S is concurrent if there is a reachable m such that $m(p) > 0$ for all places p in S. (This subsumes the case of pairs and singleton of places.) The set S is *max-concurrent* if no superset $S' \supsetneq S$ is concurrent. Computing the max-concurrent sets of a net is interesting for several reasons. First, it gives an alternative representation of the concurrency relation that can sometimes be more space efficient: (1) the max-concurrent sets provide a unique cover of the set of places of a net, and (2) we have $p \parallel q$ if and only if there is S max-concurrent such that $\{p, q\} \subseteq S$. Obviously, on safe nets, the size of the biggest max-concurrent set is the answer to the *max-marking* problem.

Another possible extension will be to support non-ordinary nets (that would require adding weights on the TFG's arcs) and unsafe nets (that can already be done with our current approach but requires changing some "axioms" used in our algorithm).

This work has been published in:

- N. Amat, S. Dal Zilio, and D. Le Botlan. "Accelerating the Computation of Dead and Concurrent Places Using Reductions". In: *Model Checking Software (SPIN)*. vol. 12864. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-84629-9_3
- N. Amat, S. Dal Zilio, and D. Le Botlan. "Leveraging polyhedral reductions for solving Petri net reachability problems". In: *International Journal on Software Tools for Technology Transfer* 25.1 (2023), pp. 95–114. DOI: 10.1007/s10009-022-00694-8

The tool related to this chapter is:

- Kong https://github.com/nicolasAmat/Kong

Chapter 7

Proving Polyhedral Equivalences

Using an Automated Method

> When there are disputes among persons, we can simply say: Let us calculate [calculemus], without further ado, to see who is right.
>
> Gottfried Wilhelm Leibniz

In this chapter, we propose an automated procedure to prove polyhedral equivalences. Our approach relies on an encoding into a set of SMT formulas whose satisfaction implies that the equivalence holds. The difficulty, in this context, arises from the fact that we need to handle infinite-state systems. For completeness, we exploit a connection with a class of Petri nets that have Presburger-definable reachability sets.

We have implemented our procedure a the tool, called Reductron, and we illustrate its use in several examples.

7.1 Introduction

This chapter concludes the theoretical contributions of the thesis, by providing an automated method for proving polyhedral equivalences. It shares some similarities with the philosophy of Chapter 2 (in which we compute certificates of invariance), that is, getting formal proofs of the correctness of our methods and tools.

Context. We introduced the concept of polyhedral reduction to solve reachability problems more efficiently. We applied this approach to two main problems: to check reachability formulas (Chapters 3 and 5); and finally, to speed up the computation of concurrent places (places that can be marked simultaneously in a reachable marking); see Chapter 6.

We proved in Chapter 3 that deciding the correctness of a polyhedral equivalence is undecidable (Theorem 3.1). This decidability result is not surprising since most equivalence problems on Petri nets are undecidable [EN94; Esp98]. Indeed, polyhedral equivalence is by essence related to the *marking equivalence* problem, which amounts to deciding if two Petri nets with the same set of places have the same reachable markings; a problem proved undecidable by Hack [Hac76]. Also, polyhedral equivalence (such as marking equivalence) entails trace equivalence, another well-known undecidable equivalence problem when we consider general Petri nets [Hac76; Hir94].

Challenge. In this context, we use the term *parametric* to stress that we manipulate semilinear sets of markings, meaning sets that can be defined using a Presburger arithmetic formula C. In particular, we reason about parametric nets (N, C) instead of marked nets (N, m_0), with the intended meaning that all markings satisfying C are potential initial markings of N. We also define an extended notion of polyhedral equivalence between parametric nets, called *parametric polyhedral equivalence* and denoted $(N_1, C_1) \approxeq_E (N_2, C_2)$, whereas our original definition (Definition 3.2) was between marked nets only. Although the parametric polyhedral equivalence is a subcase of our original equivalence relation, we show that its additional constraints ensure the decidability of the problem. Our challenge here is to provide a procedure to automatically prove the correctness of such parametric equivalence.

Proposal. We describe a procedure to automatically prove polyhedral equivalences between pairs of parametric Petri nets. We show that given a valid equivalence statement $(N_1, C_1) \approxeq_E (N_2, C_2)$, it is possible to derive a Presburger formula in a constructive way, whose satisfaction implies that the equivalence holds. We implemented this procedure in the tool Reductron, on top of an SMT-solver for Linear Integer Arithmetic (LIA) and show that our approach is applicable in practice. Even if we prove that this problem is decidable (see Theorem 7.12), our implementation is only a semi-decision procedure since we rely on the external tool FAST, which may not terminate if the equivalence does not hold. If anything, it makes the fact that we may translate our problem into Presburger arithmetic quite remarkable.

Outline and Contributions. The chapter is organized as follows. After presenting an overview of the method in Sect. 7.2, we define our central notion of *parametric polyhedral equivalence* in Sect. 7.3 and prove several of its properties in Sect. 7.8. In particular, we prove that this new polyhedral equivalence is preserved when "duplicating labeled transitions". These properties mean that every abstraction rule we prove can be safely applied in every context and that each rule can be used as a "rule schema". Our definition relies on our notion of polyhedral equivalence; see Definition 3.2 in Chapter 3. We describe our proof procedure in Sect. 7.4, defined as constructing a set of four *core requirements*, each expressed as separate quantified LIA formulas. A key ingredient in this translation is to build a predicate, $\tau^\star_C$, which encodes

the markings reachable by firing only the silent transitions of a net. We defer the definition of this predicate until Sect. 7.5, where we show how it can be obtained using the output of the FAST tool. From this procedure, we prove that our problem is decidable in Sect 7.6, and we conclude by presenting the results obtained with our tool Reductron implementing our approach on some concrete examples.

7.2 Overview of the Approach

Our proof procedure is based on the standard E-abstraction equivalence defined in Chapter 3 (Definition 3.2), and not on its relaxed relation (Definition 4.1). This choice is motivated by the fact that we want to prove reduction rules that can be composable, hence the need for a sequence-based relation.

Our approach can be summarized as follows. We start from an initial net (N_1, C_1) and derive a polyhedral equivalence $(N_1, C_1) \approx_E (N_2, C_2)$ by applying a set of *reduction rules* in an iterative and compositional way. We have presented in Chapter 3 some hand-proved structural reduction rules. However, we also implement several other kinds of reduction rules—often subtler to use and more complicated to prove correct—which explains why we want machine-checkable proofs of equivalence. For instance, some of our rules are based on the identification of Petri nets subclasses in which the set of reachable markings equals the set of potentially reachable ones, a property we call the PR-R equality in [Huj+20a; Huj+20b]. We use this kind of rule in the example of the "SwimmingPool" model of Fig. 7.8, a classical example of Petri net often used in case studies (see e.g. [BF99]).

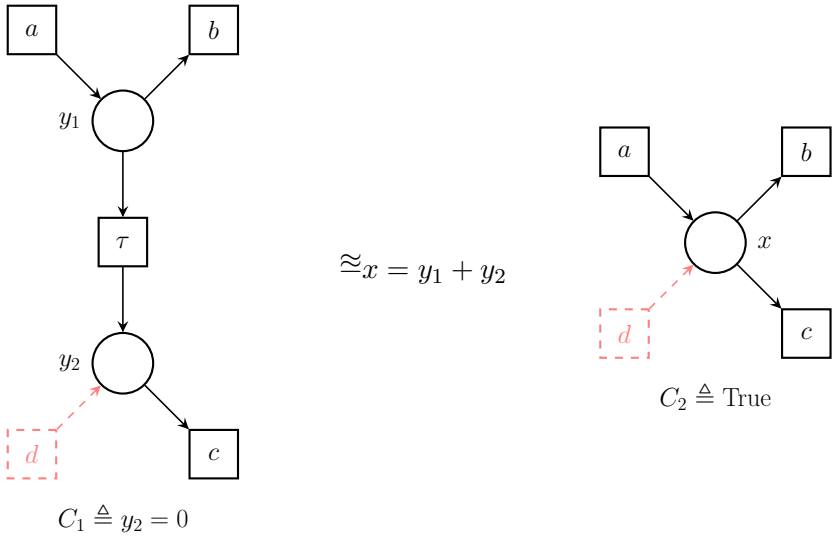


Fig. 7.1 Parametrized equivalence rule [CONCAT], $(N_1, C_1) \approx_E (N_2, C_2)$, between nets N_1 (left) and N_2 (right), for the relation $E \triangleq (x = y_1 + y_2)$.

We give in Fig. 7.1 a parametric version of the fundamental [CONCAT] reduction rule (Fig. 3.4) that allows us to fuse two places connected by a direct, silent transition. We give another example with [MAGIC], in Fig. 7.2, which illustrates a more complex agglomeration rule (more details about the "blue dashed" transitions are given in Sect.7.8), and refer to other examples in Sect. 7.9.

The parametric net (N_1, C_1) (left of Fig. 7.1) has a condition that entails that place y_2 should be empty initially ($y_2 = 0$), whereas net (N_2, C_2) has a trivial constraint, which can be interpreted as simply $x \geqslant 0$. We can show (see Sect. 7.3) that nets N_1 and N_2 are E-equivalent, which amounts to prove that any marking $(y_1 : k_1,\ y_2 : k_2)$ of N_1, reachable by firing a transition sequence ϱ, can be associated with the marking $(x : k_1 + k_2)$ of N_2, also reachable by the same firing sequence. Actually, we prove that this equivalence is sound when no transition can input a token directly into place y_2 of N_1. This means that the rule is correct in the absence of the "red dashed" transition (with label d), but that our procedure should flag the rule as unsound when transition d is present.

The results presented in this chapter provide an automated technique for proving the correctness of polyhedral reduction rules. This helps us gain more confidence in the correctness of our tools and is also helpful if we want to add new reduction rules. Indeed, up until now, all our rules were proven using "manual theorem proving" (see Sect. 3.4.1), which can be tedious and error-prone. Incidentally, the theory we developed for this chapter also helped us better

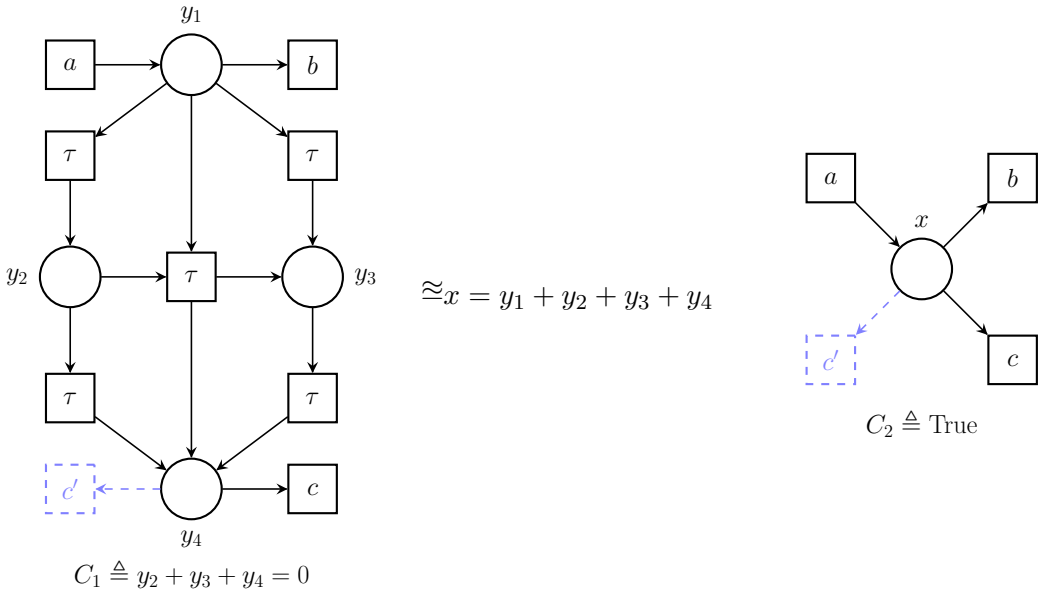


Fig. 7.2 Parametrized equivalence rule [MAGIC].

understand the constraints necessary when designing new reduction rules. A critical part of our approach relies on the ability, given a Presburger predicate C, to encode the set of markings reachable from C by firing only silent transitions, that we denote $\tau_C^\star$ in the following. Our

approach draws a connection with previous works [Bar+03; LS05; Bar+08; Ler13] that study the class of Petri nets that have Presburger-definable reachability sets, called *flat nets*. We should also use a tool implemented by the same authors, called FAST, which provides a method for representing the reachability set of flat nets. Basically, we gain the insight that polyhedral reduction provides a way to abstract away (or collapse) the sub-parts of a net that are flat. Note that our approach may work even though the reachability set of the whole net is not semilinear since only the abstracted part must be flat. We also prove that when $(N_1, C_1) \approxeq_E (N_2, C_2)$ then necessarily the sets $\tau^\star_{C_1}$ and $\tau^\star_{C_2}$ are semilinear.

7.3 Parametric Reduction Rules and Equivalence

E-abstraction equivalence is defined on marked nets (Definition 3.2); thus the reduction rules defined in Chapter 3, which also are E-abstraction equivalences mention marked nets as well. Their soundness was proven manually, using constrained parameters for initial markings. Such constraints on markings are called *coherency constraints*.

7.3.1 Coherency Constraints

We define a notion of *coherency constraint*, C, that must hold not only in the initial state but also in a sufficiently large subset of reachable markings. We have already seen an example with the constraint $C_1 \triangleq (y_2 = 0)$ used in rule [CONCAT]. Without the use of C_1, rule [CONCAT] would be unsound since net N_2 (right of Fig. 7.1) could fire transition b more often than its counterpart, N_1.

Since C is a predicate on markings, we equivalently consider it as a subset of markings or as a logic formula so that we may equivalently write $m \models C$ or $m \in C$ to indicate that $C(m)$ is true.

Remember we have already defined observable sequences $(N, m) \stackrel{\sigma}{\Rightarrow} (N, m')$ in Sect. 1.1.4. Now, we must consider firing sequences that must not finish with τ transitions. Hence, we define a complementary relation $(N, m) \stackrel{\sigma\rangle}{\Longrightarrow} (N, m')$, written simply $m \stackrel{\sigma\rangle}{\Longrightarrow} m'$, as follows:

- $(N, m) \stackrel{\epsilon\rangle}{\Longrightarrow} (N, m)$ holds for any marking m.
- $(N, m) \stackrel{\sigma.a\rangle}{\Longrightarrow} (N, m')$ holds for any markings m, m' and $a, \sigma \in \Sigma \times \Sigma^*$, if there exists a marking m'' and a transition t such that $l(t) = a$ and $(N, m) \stackrel{\sigma}{\Rightarrow} (N, m'') \stackrel{t}{\rightarrow} (N, m')$.

It is immediate that $m \stackrel{\sigma\rangle}{\Longrightarrow} m'$ implies $m \stackrel{\sigma}{\Rightarrow} m'$. Note the difference between $m \stackrel{\epsilon}{\Rightarrow} m'$, which stands for any sequence of τ transitions, and $m \stackrel{\epsilon\rangle}{\Longrightarrow} m'$, which implies $m = m'$ (the sequence is empty).

Definition 7.1 (Coherent Net)**.** *Given a Petri net N and a predicate C on markings, we say that N satisfies the coherency constraint C, or equivalently, that (N, C) is a coherent net, if and only if for all firing sequences $m \stackrel{\sigma}{\Rightarrow} m'$ with $m \in C$ we have*

$$\exists m'' \in C \;.\; m \stackrel{\sigma\rangle}{\Longrightarrow} m'' \wedge m'' \stackrel{\epsilon}{\Rightarrow} m'$$

Intuitively, if all τ transitions are irreversible, we can define a partial order on markings with $m < m'$ whenever $m \xrightarrow{\tau} m'$ holds. Then, markings satisfying the coherency constraint C must be minimal with respect to this partial order.

In this chapter, we wish to prove automatically the soundness of a given reduction rule. A reduction rule basically consists of two nets with their coherency constraints and a Presburger relation between markings.

Definition 7.2 (Parametric Reduction Rule)**.** *A parametric reduction rule is written $(N_1, C_1) >_E (N_2, C_2)$, where (N_1, C_1) and (N_2, C_2) are both coherent nets, and C_1, C_2 and E are Presburger formulas whose free variables are in $P_1 \cup P_2$.*

A given reduction rule $(N_1, C_1) >_E (N_2, C_2)$ is a candidate, which we will analyze to prove its soundness: is it an E-abstraction equivalence?

7.3.2 Parametric E-Equivalence Definition

Our analysis relies on a richer definition of E-abstraction, namely *parametric* E-abstraction (Definition 7.3, next), which includes the coherency constraints C_1, C_2. Parametric E-abstraction entails E-abstraction for each instance of its parameters (Theorem 7.1). Essentially, for any sequence $m_1 \stackrel{\sigma}{\Rightarrow} m_1'$ with $m_1 \in C_1$, there exists a marking m_2' such that $m_1' \equiv_E m_2'$; and for every marking $m_2 \in C_2$ related to m_1 up-to E, i.e., $m_1 \equiv_E m_2$, all markings m_2' related to m_1' (i.e., $m_1' \equiv_E m_2'$) can be reached from m_2 by the same observable sequence σ.

To ease the presentation, we define the notation

$$m_1 \langle C_1 E C_2 \rangle m_2 \triangleq m_1 \models C_1 \wedge m_1 \equiv_E m_2 \wedge m_2 \models C_2 \tag{7.1}$$

Definition 7.3 (Parametric E-Abstraction)**.** *Assume that* $(N_1, C_1) >_E (N_2, C_2)$ *is a parametric reduction rule. We say that* (N_2, C_2) *is a parametric* E*-abstraction of* (N_1, C_1)*, denoted* $(N_1, C_1) \preceq_E (N_2, C_2)$ *if and only if:*

(S1) *for all markings* m_1 *satisfying* C_1 *there exists a marking* m_2 *such that* $m_1 \langle C_1 E C_2 \rangle m_2$*;*
(S2) *for all firing sequences* $m_1 \stackrel{\epsilon}{\Rightarrow} m_1'$ *and all markings* m_2*, we have* $m_1 \equiv_E m_2$ *implies* $m_1' \equiv_E m_2$*;*
(S3) *for all firing sequences* $m_1 \stackrel{\sigma}{\Rightarrow} m_1'$ *and all marking pairs* m_2*,* m_2'*, if* $m_1 \langle C_1 E C_2 \rangle m_2$ *and* $m_1' \equiv_E m_2'$ *then we have* $m_2 \stackrel{\sigma}{\Rightarrow} m_2'$*.*

We say that (N_1, C_1) *and* (N_2, C_2) *are in parametric* E*-equivalence, denoted* $(N_1, C_1) \cong_E (N_2, C_2)$*, when we have both* $(N_1, C_1) \preceq_E (N_2, C_2)$ *and* $(N_2, C_2) \preceq_E (N_1, C_1)$*.*

Condition (S1) corresponds to the solvability of the Presburger formula E with respect to the marking predicates C_1 and C_2. Condition (S2) ensures that silent transitions of N_1 are abstracted away by the formula E and are therefore invisible to N_2. Condition (S3) closely follows condition (A2) of the standard E-abstraction equivalence.

Note that equivalence $\cong$ is still not a bisimulation, in the same way, that $\equiv$ from Definition 3.2. It is defined only for observable sequences starting from states satisfying the coherency constraint C_1 of N_1 or C_2 of N_2, and so this relation is usually not valid on every pair of equivalent markings $m_1 \equiv_E m_2$.

7.3.3 Instantiation Laws

Parametric E-abstraction implies E-abstraction for every instance pair satisfying the coherency constraints C_1, C_2.

Theorem 7.1 (Parametric E-Abstraction Instantiation)**.** *Assume that* $(N_1, C_1) \preceq_E (N_2, C_2)$ *is a parametric* E*-abstraction. Then for every pair of markings* m_1, m_2*,* $m_1 \langle C_1 E C_2 \rangle m_2$ *implies* $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$*.*

Proof. Consider $(N_1, C_1) \preceq_E (N_2, C_2)$, a parametric E-abstraction, and m_1, m_2 such that $m_1 \langle C_1 E C_2 \rangle m_2$ holds. By definition of $m_1 \langle C_1 E C_2 \rangle m_2$, see Equation (7.1), condition (A1) of Definition 3.2 is immediately satisfied. We show (A2) by considering an observable sequence $(N_1, m_1) \stackrel{\sigma}{\Rightarrow} (N_1, m_1')$. Since m_1 satisfies the coherency constraint C_1, we get from Definition 7.1 a marking $m_1'' \in C_1$ such that $m_1 \stackrel{\sigma\rangle}{\Longrightarrow} m_1'' \stackrel{\epsilon}{\Rightarrow} m_1'$ holds. By applying (S1) to m_1'', we get a marking m_2' such that $m_1'' \langle C_1 E C_2 \rangle m_2'$ holds, which implies $m_1'' \equiv_E m_2'$. Then, by applying (S2) to $m_1'' \stackrel{\epsilon}{\Rightarrow} m_1'$, we obtain the expected result $m_1' \equiv_E m_2'$. Finally, for all markings m_2' such that $m_1' \equiv_E m_2'$, we conclude $m_2 \stackrel{\sigma}{\Rightarrow} m_2'$ from (S3). Condition (A2) is proved, hence $(N_1, m_1) \sqsubseteq_E (N_2, m_2)$ holds. □

7.4 Automated Proof Procedure

Our automated proof procedure receives a candidate reduction rule (Definition 7.2) as input and has three possible outcomes: (i) the candidate is proven sound, congratulations you have established a new parametric E-abstraction equivalence; (ii) the candidate is proven unsound, try to understand why and fix it; or (iii) we cannot conclude, because part of our procedure relies on a semi-algorithm (see Sect. 7.5) for expressing the set of reachable markings of a flat subnet as a linear constraint.

Given the candidate reduction rule, the procedure generates SMT queries, which we call *core requirements* (defined in Sect. 7.4.2) that are solvable if and only if the candidate is a parametric E-abstraction (Theorems 7.9 and 7.10, Sect. 7.4.3). We express these constraints into Presburger predicates, so it is enough to use solvers for the theory of formulas on Linear Integer Arithmetic, what is known as LIA in SMT-LIB [BFT17]. We illustrate the results given in this section using a diagram (Fig. 7.3) that describes the dependency relations between conditions (S1), (S2), (S3) and their encoding as core requirements.

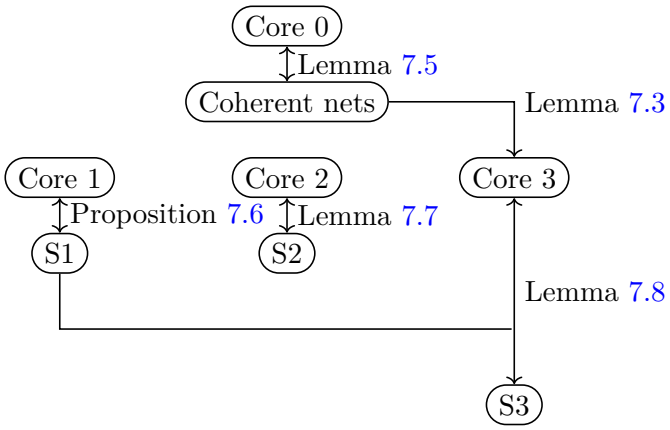


Fig. 7.3 Detailed dependency relations.

7.4.1 Presburger Encoding of Parametric Petri Net Semantics

We start by defining a few formulas that ease the subsequent expression of core requirements. This will help with the most delicate point of our encoding, which relies on how to encode sequences of transitions. Note that the coherency constraints of reduction rules are already defined as such.

In order to keep track of fired transitions in our encoding, and without any loss of generality we assume that our alphabet of labels Σ is a subset of the natural numbers ($\Sigma \subset \mathbb{N}^*$), except 0 that is reserved for τ.

We overload the Presburger predicate $\mathrm{T}(\boldsymbol{p}, \boldsymbol{p'})$ from Sect. 1.2.3 into $\mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}, a)$ to describe the relation between the markings before ($\boldsymbol{p}$) and after ($\boldsymbol{p'}$) firing a transition with label a.

With this convention, formula $\mathrm{T}(m, m', a)$ holds if and only if $m \xrightarrow{t} m'$ holds for some transition t such that $l(t) = a$ (which implies $a \neq 0$).

$$\mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}, a) \triangleq \bigvee_{t \in T} \left(\mathrm{ENBL}_t(\boldsymbol{p}) \wedge \Delta_t(\boldsymbol{p}, \boldsymbol{p'}) \wedge a = l(t)\right) \tag{7.2}$$

We admit the following, for all markings m, m' and label a:

$$\models \mathrm{T}(m, m', a) \iff \exists t \,.\, m \xrightarrow{t} m' \wedge l(t) = a \tag{7.3}$$

In order to define the core requirements, we additionally require a predicate $\tau_C^*(\boldsymbol{p}, \boldsymbol{p'})$ encoding the markings reachable by firing any sequence of silent transitions from a state satisfying the coherency constraint C. And so, the following constraint must hold:

$$\models m \in C \implies (\tau_C^*(m, m') \iff m \overset{\epsilon}{\Rightarrow} m') \tag{7.4}$$

Since $m \overset{\epsilon}{\Rightarrow} m'$ may fire an arbitrary number of silent transitions τ, the predicate τ_C is not guaranteed to be expressible as a Presburger formula in the general case. Yet, in Sect. 7.5, we characterize the Petri nets for which τ_C can be expressed in Presburger arithmetic, including all the polyhedral reductions we meet in practice (we explain why).

Thanks to this predicate, we define the formula $\acute{\mathrm{T}}_C(\boldsymbol{p}, \boldsymbol{p'}, a)$ encoding the reachable markings from a marking satisfying the coherency constraint C, by firing any number of silent transitions, followed by a transition labeled with a. Then, we define $\hat{\mathrm{T}}$ which extends $\acute{\mathrm{T}}$ with any number of silent transitions after a and also allows for only silent transitions (no transition a).

$$\acute{\mathrm{T}}_C(\boldsymbol{p}, \boldsymbol{p'}, a) \triangleq \exists \boldsymbol{x} \,.\, \tau_C^*(\boldsymbol{p}, \boldsymbol{x}) \wedge \mathrm{T}(\boldsymbol{x}, \boldsymbol{p'}, a) \tag{7.5}$$

$$\hat{\mathrm{T}}_C(\boldsymbol{p}, \boldsymbol{p'}, a) \triangleq \left(\exists \boldsymbol{y} \,.\, \acute{\mathrm{T}}_C(\boldsymbol{p}, \boldsymbol{y}, a) \wedge C(\boldsymbol{y}) \wedge \tau_C^*(\boldsymbol{y}, \boldsymbol{p'})\right) \tag{7.6}$$

$$\vee \left(a = 0 \wedge \tau_C^*(\boldsymbol{p}, \boldsymbol{p'})\right) \tag{7.7}$$

Lemma 7.2. *For any markings m, m' and label a such that $m \in C$, we have $\models \acute{\mathrm{T}}_C(m, m', a)$ if and only if $m \overset{a\rangle}{\Longrightarrow} m'$ holds.*

Proof. We show both directions separately.

- Assume that $m \overset{a\rangle}{\Longrightarrow} m'$. By definition, this implies that there exists m'' and a transition t such that $l(t) = a$ and $m \overset{\epsilon}{\Rightarrow} m'' \xrightarrow{t} m'$. Therefore, $\tau_C^*(m, m'')$ is valid by Equation (7.4), and $\mathrm{T}(m'', m', a)$ is valid by Equation (7.3), hence the expected result $\models \acute{\mathrm{T}}_C(m, m', a)$.
- Conversely, assume that $\acute{\mathrm{T}}_C(m, m', a)$ is valid. Then, by Equation (7.5) there exists a marking m'' such that both $\tau_C^*(m, m'')$ and $\mathrm{T}(m'', m', a)$ are valid. From Equation (7.4),

we get $m \overset{\epsilon}{\Rightarrow} m''$, and Equation (7.3) implies $\exists t \,.\, m'' \xrightarrow{t} m' \wedge l(t) = a$. Thus, $m \overset{\epsilon}{\Rightarrow} m'' \xrightarrow{t} m'$, that is the expected result $m \overset{a\rangle}{\Longrightarrow} m'$.

□

Lemma 7.3. *Given a coherent net (N, C), for any markings m, m' such that $m \in C$ and $a \in \Sigma \cup \{0\}$, we have $\models \hat{\mathrm{T}}_C(m, m', a)$ if and only if either $m \overset{\epsilon}{\Rightarrow} m'$ and $a = 0$, or $m \overset{a}{\Rightarrow} m'$.*

Proof. We show both directions separately.

- Assume that $m \overset{\epsilon}{\Rightarrow} m'$ and $a = 0$, then $\tau_C^*(m, m')$ is valid by Equation (7.4), hence the expected result $\models \hat{\mathrm{T}}_C(m, m', a)$ from Equation (7.7).
- Assume that $m \overset{a}{\Rightarrow} m'$. From Definition 7.1 (coherent net), there exists $m'' \in C$ such that $m \overset{a\rangle}{\Longrightarrow} m'' \overset{\epsilon}{\Rightarrow} m'$. Then, we get $\models \acute{\mathrm{T}}_C(m, m'', a)$ from Lemma 7.2, and $\models \tau_C^*(m'', m')$ from Equation (7.4). Consequently, $\hat{\mathrm{T}}_C(m, m', a)$ is valid from Equation (7.6).
- Conversely, assume that $\hat{\mathrm{T}}_C(m, m', a)$ holds by Equation (7.7), then $a = 0$ and $\models \tau_C^*(m, m')$, which implies $m \overset{\epsilon}{\Rightarrow} m'$ by Equation (7.4). This is the expected result.
- Finally, assume that $\hat{\mathrm{T}}_C(m, m', a)$ holds by Equation (7.6), then there exists a marking $m'' \in C$ such that $\models \acute{\mathrm{T}}_C(m, m'', a)$ and $\models \tau_C^*(m'', m')$. This implies $m \overset{a\rangle}{\Longrightarrow} m'' \overset{\epsilon}{\Rightarrow} m'$ from Lemma 7.2 and Equation (7.4). This implies the expected result $m \overset{a}{\Rightarrow} m'$.

□

Remember, we denote $\tilde{E}(\boldsymbol{x}, \boldsymbol{y})$ the formula obtained from E where free variables are substituted as follows: place names in N_1 are replaced with variables in $\boldsymbol{x}$, and place names in N_2 are replaced with variables in $\boldsymbol{y}$ (making sure that bound variables of E are renamed to avoid interference). When the same place occurs in both nets, say $p_i^1 = p_j^2$, we also add the equality constraint $(x_i = y_j)$ to $\tilde{E}$ in order to preserve this equality constraint.

7.4.2 Core Requirements: Parametric E-Abstraction Encoding

In order to check conditions (S1)–(S3) of parametric E-abstraction (Definition 7.3), we define a set of Presburger formulas, called *core requirements*, to be verified using an external SMT solver ((Core 1) to (Core 3)). You will find an illustration of these requirements in Figs. 7.4–7.7. The satisfaction of these requirements entails the parametric E-abstraction relation. We have deliberately stressed the notations to prove that (N_2, C_2) is a parametric E-abstraction of (N_1, C_1). Of course, each constraint must be checked in both directions to obtain the equivalence. Also, so as not to complexify the notations, we assume that it is obvious from the context if a transition relation belongs to N_1 or N_2.

Verifying That a Net is Coherent

The first step is verifying that both nets N_1 and N_2 satisfy their coherency constraints C_1 and C_2 (the coherency constraint is depicted in Figure 7.4). We recall Definition 7.1:

Definition (Coherent Net)**.** *For all firing sequences* $m \stackrel{\sigma}{\Rightarrow} m'$ *with* $m \in C$ *there exists a marking* m'' *satisfying* C *such that* $m \stackrel{\sigma\rangle}{\Longrightarrow} m''$ *and* $m'' \stackrel{\epsilon}{\Rightarrow} m'$*.*

Below, we encode a simpler relation with sequences σ of size 1. This relies on the following result:

Lemma 7.4. *Parametric net* (N, C) *is coherent if and only if for all firing sequence* $m \stackrel{a\rangle}{\Longrightarrow} m'$ *with* $m \in C$ *and* $a \in \Sigma$ *we have* $\exists m'' \in C \;.\; m \stackrel{a\rangle}{\Longrightarrow} m'' \wedge m'' \stackrel{\epsilon}{\Rightarrow} m'$*.*

We deliberately consider a firing sequence $m \stackrel{a\rangle}{\Longrightarrow} m'$ (and not $m \stackrel{a}{\Rightarrow} m'$), since the encoding relies only on $\acute{\mathrm{T}}_C$ (that is, $\stackrel{a\rangle}{\Longrightarrow}$), not on $\hat{\mathrm{T}}_C$ (that is, $\stackrel{a}{\Rightarrow}$).

Proof. The "only if" part is immediate, as a particular case of Definition 7.1 and noting that $m \stackrel{a\rangle}{\Longrightarrow} m'$ implies $m \stackrel{a}{\Rightarrow} m'$. Conversely, assume that the property stated in the lemma is true. Then, we show by induction on the size of σ that Definition 7.1 holds for any σ. Note that the base case $\sigma = \epsilon$ always holds, for any net, by taking $m'' = m$. Now, consider a non-empty sequence $\sigma = \sigma'.a$ and $m \stackrel{\sigma'.a}{\Longrightarrow} m'$ with $m \in C$. By definition, there exist m_1 and m_2 such that $m \stackrel{\sigma'}{\Rightarrow} m_1 \stackrel{a\rangle}{\Longrightarrow} m_2 \stackrel{\epsilon}{\Rightarrow} m'$. By induction hypothesis, on $m \stackrel{\sigma'}{\Rightarrow} m_1$, there exists $m_3 \in C$ such that $m \stackrel{\sigma'\rangle}{\Longrightarrow} m_3 \stackrel{\epsilon}{\Rightarrow} m_1$. Therefore, we have $m \stackrel{\sigma'\rangle}{\Longrightarrow} m_3 \stackrel{\epsilon}{\Rightarrow} m_1 \stackrel{a\rangle}{\Longrightarrow} m_2 \stackrel{\epsilon}{\Rightarrow} m'$, which can simply be written $m \stackrel{\sigma'\rangle}{\Longrightarrow} m_3 \stackrel{a\rangle}{\Longrightarrow} m_2 \stackrel{\epsilon}{\Rightarrow} m'$. Using the property stated in the lemma on $m_3 \stackrel{a\rangle}{\Longrightarrow} m_2$, we get a marking $m_4 \in C$ such that $m_3 \stackrel{a\rangle}{\Longrightarrow} m_4 \stackrel{\epsilon}{\Rightarrow} m_2$. Hence, $m \stackrel{\sigma'\rangle}{\Longrightarrow} m_3 \stackrel{a\rangle}{\Longrightarrow} m_4 \stackrel{\epsilon}{\Rightarrow} m_2 \stackrel{\epsilon}{\Rightarrow} m'$ holds, which can be simplified as $m \stackrel{\sigma'.a\rangle}{\Longrightarrow} m_4 \stackrel{\epsilon}{\Rightarrow} m'$. This is the expected result. □

Therefore, we can encode Definition 7.1 using the following formula:

$$\forall \boldsymbol{p}, \boldsymbol{p'}, a \;.\; C(\boldsymbol{p}) \wedge \acute{\mathrm{T}}_C(\boldsymbol{p}, \boldsymbol{p'}, a) \implies \exists \boldsymbol{p''} \;.\; C(\boldsymbol{p''}) \wedge \acute{\mathrm{T}}_C(\boldsymbol{p}, \boldsymbol{p''}, a) \wedge \tau_C^*(\boldsymbol{p''}, \boldsymbol{p'}) \qquad \text{(Core 0)}$$

Lemma 7.5. *Given a Petri net* N*, the constraint (Core 0) is valid if and only if the net satisfies the coherency constraint* C*.*

Proof. Constraint (Core 0) is an immediate translation of the property stated in Lemma 7.4. □

Given a net N, a constraint C expressed as a Presburger formula, and a formula τ_C^* that captures $\stackrel{\epsilon}{\Rightarrow}$ transitions (as obtained in Sect. 7.5), we are now able to check automatically that a net (N, C) is coherent. Thus, from now on, we assume that the considered nets (N_1, C_1) and (N_2, C_2) are indeed coherent.

Coherent Solvability

The first requirement of the parametric E-abstraction relates to the solvability of formula E with regard to the coherency constraint C_1 and is encoded by (Core 1). This requirement

ensures that every marking of N_1 satisfying C_1 can be associated with at least one marking of N_2 satisfying C_2. Let us recall (S1), taken from Definition 7.3:

Definition (S1). *For all markings m_1 satisfying C_1 there exists a marking m_2 such that $m_1 \langle C_1 E C_2 \rangle m_2$.*

Condition (S1) is depicted in Figure 7.5. We propose to encode it by the following Presburger formula:

$$\forall \boldsymbol{p_1} \,.\, C_1(\boldsymbol{p_1}) \implies \exists \boldsymbol{p_2} \,.\, \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge C_2(\boldsymbol{p_2}) \qquad \text{(Core 1)}$$

Since the encoding is immediate, we admit this proposition:

Proposition 7.6. *The constraint (Core 1) is valid if and only if (S1) holds.*

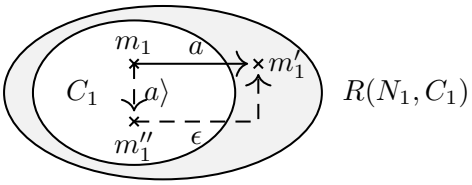


Fig. 7.4 Illustration of (Core 0).

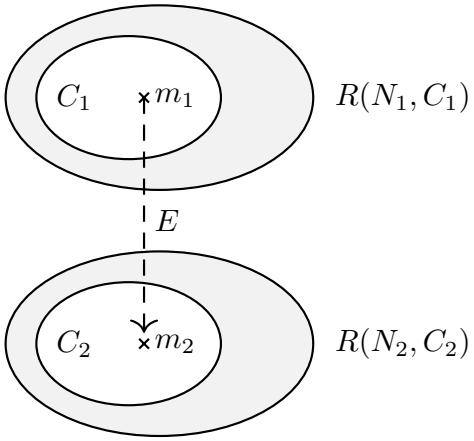


Fig. 7.5 Illustration of (Core 1).

Silent Constraints

So far, we have focused on the specific case of coherent nets, which refers to intermediate coherent markings. Another notable feature of parametric E-abstractions is the ability to fire any number of silent transitions without altering the solutions of E. In other words, if two markings, m_1 and m_2, are solutions of E, then firing any silent sequence from m_1 (or m_2) will always lead to a solution of $E \wedge m_2$ (or $E \wedge m_1$). This means that silent transitions must be invisible to the other net.

Let us recall (S2), taken from Definition 7.3:

Definition (S2). *For all firing sequences $m_1 \overset{\epsilon}{\Rightarrow} m_1'$ and all markings m_2, we have $m_1 \equiv_E m_2$ implies $m_1' \equiv_E m_2$.*

It actually suffices to show the result for each silent transition $t \in T_1$ taken separately:

Lemma 7.7. *Condition (S2) holds if and only if, for all markings m_1, m_2 such that $m_1 \equiv_E m_2$, and for all $t_1 \in T_1$ such that $l_1(t_1) = \tau$, we have $m_1 \xrightarrow{t_1} m_1'$ implies $m_1' \equiv_E m_2$.*

Proof. The "only if" way is only a particular case of (S2) with a single silent transition t_1. For the "if" way, (S2) is shown from the given property by transitivity. □

Thanks to this result, we encode (S2) by the following core requirement:

$$\forall \boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_1'} \;.\; \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge \tau(\boldsymbol{p_1}, \boldsymbol{p_1'}) \implies \tilde{E}(\boldsymbol{p_1'}, \boldsymbol{p_2}) \qquad \text{(Core 2)}$$

where $\tau(\boldsymbol{p_1}, \boldsymbol{p_1'})$ is defined as $\tau(\boldsymbol{p_1}, \boldsymbol{p_1'}) \triangleq \bigvee_{t \in T | l(t) = \tau} (\mathrm{ENBL}_t(\boldsymbol{p_1}) \wedge \Delta_t(\boldsymbol{p_1}, \boldsymbol{p_1'}))$

Reachability

Let us recall the definition of (S3), taken from Definition 7.3:

Definition (S3)**.** *For all firing sequences $m_1 \stackrel{\sigma}{\Rightarrow} m_1'$ and all marking pairs m_2, m_2', if $m_1 \langle C_1 E C_2 \rangle m_2$ and $m_1' \equiv_E m_2'$ then we have $m_2 \stackrel{\sigma}{\Rightarrow} m_2'$.*

Condition (S3) mentions sequences σ of arbitrary length. We encode it with a formula dealing only with sequences of length at most 1, thanks to the following result:

Lemma 7.8. *Given a parametric reduction rule $(N_1, C_1) >_E (N_2, C_2)$ which satisfies condition (S1), then condition (S3) holds if and only if for all firing sequences $m_1 \stackrel{\sigma}{\Rightarrow} m_1'$ with $\sigma = \epsilon$ or $\sigma = a$ with $a \in \Sigma$, and all markings m_2, m_2', we have $m_1 \langle C_1 E C_2 \rangle m_2 \wedge m_1' \equiv_E m_2' \implies m_2 \stackrel{\sigma}{\Rightarrow} m_2'$.*

Proof. The given property is necessary as a particular case of (S3) taking $\sigma = a$ or $\sigma = \epsilon$. Conversely, assume that the given property holds. We show by induction on the size of σ that (S3) holds for any sequence σ. The hypothesis ensures the base cases $\sigma = a$ and $\sigma = \epsilon$. Now, consider a non-empty sequence $\sigma = \sigma'.a$, and $m_1 \stackrel{\sigma}{\Rightarrow} m_1'$ (i), as well as markings m_2, m_2' such that $m_1 \langle C_1 E C_2 \rangle m_2$ and $m_1' \equiv_E m_2'$ holds. We have to show $m_2 \stackrel{\sigma}{\Rightarrow} m_2'$. From (i), we have $m_1 \stackrel{\sigma'.a}{\Longrightarrow} m_1'$, that is, there exists a marking u_1 such that $m_1 \stackrel{\sigma'}{\Rightarrow} u_1 \stackrel{a}{\Rightarrow} m_1'$ (ii). By Definition 7.1, there exists $u_1' \in C_1$ such that $m_1 \stackrel{\sigma'\rangle}{\Longrightarrow} u_1' \stackrel{\epsilon}{\Rightarrow} u_1$ (iii). Also, by condition (S1), there exists a marking u_2' of N_2 such that $u_1' \langle C_1 E C_2 \rangle u_2'$, which implies $u_1' \equiv_E u_2'$ (iv). Hence, by induction hypothesis on $m_1 \stackrel{\sigma'}{\Rightarrow} u_1'$, we have $m_2 \stackrel{\sigma'}{\Rightarrow} u_2'$ (α). From (ii) and (iii), we get $u_1' \stackrel{a}{\Rightarrow} m_1'$ (v). Applying the property of the lemma on (iv) and (v), we get $u_2' \stackrel{a}{\Rightarrow} m_2'$ (β). Combining (α) and (β) leads to $m_2 \stackrel{\sigma'.a}{\Longrightarrow} m_2'$, that is the expected result $m_2 \stackrel{\sigma}{\Rightarrow} m_2'$. □

Thanks to Lemma 7.8, we can encode (S3) by the following formula:

$$\forall \boldsymbol{p_1}, \boldsymbol{p_2}, a, \boldsymbol{p'_1}, \boldsymbol{p'_2} \;.\; \langle C_1 E C_2 \rangle(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge \hat{\mathrm{T}}_{C_1}(\boldsymbol{p_1}, \boldsymbol{p'_1}) \wedge \tilde{E}(\boldsymbol{p'_1}, \boldsymbol{p'_2}) \implies \hat{\mathrm{T}}_{C_2}(\boldsymbol{p_2}, \boldsymbol{p'_2}) \quad \text{(Core 3)}$$

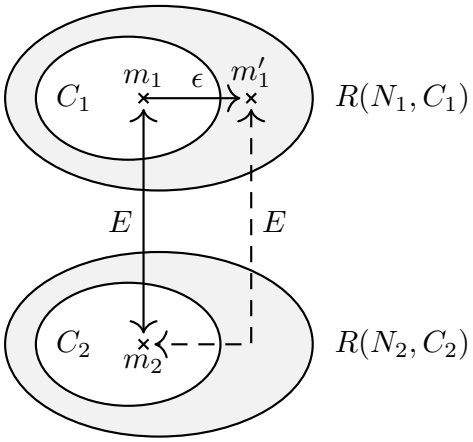


Fig. 7.6 Illustration of (Core 2).

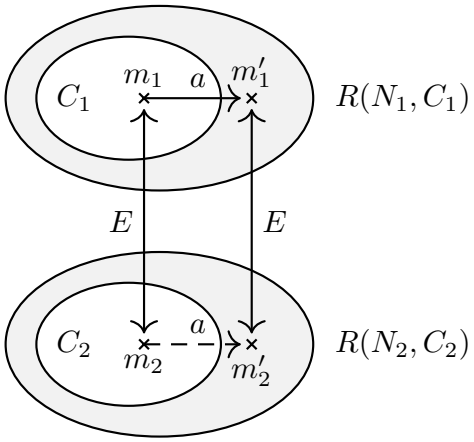


Fig. 7.7 Illustration of (Core 3).

7.4.3 Global Procedure

This section considers the entire process for proving parametric E-abstraction. We demonstrate that verifying requirements (Core 0) to (Core 3) is sufficient for obtaining a sound abstraction (Theorem 7.9). We also prove these conditions are necessary (Theorem 7.10).

Theorem 7.9 (Soundness)**.** *Given two nets N_1, N_2 and constraints C_1, C_2 expressed as Presburger formulas, if core requirement (Core 0) holds for both (N_1, C_1) and (N_2, C_2), and if core requirements (Core 1), (Core 2), and (Core 3) are valid, then the rule is a parametric E-abstraction: $(N_1, C_1) \preceq_E (N_2, C_2)$.*

Proof. If (Core 0) holds for (N_1, C_1), then (N_1, C_1) is a coherent net by Lemma 7.5. Similarly, for (N_2, C_2). Hence, $(N_1, C_1) >_E (N_2, C_2)$ is a parametric reduction rule. By Proposition 7.6, and since (Core 1) is valid, we get (S1) from Definition 7.3. Similarly, by Lemma 7.7, and since (Core 2) is valid, we get (S2). Finally, (S3) holds by Lemma 7.8 since (Core 3) is valid and since (S1) is known to hold. (S1), (S2), (S3) entail $(N_1, C_1) \preceq_E (N_2, C_2)$ by Definition 7.3. □

The converse also holds:

Theorem 7.10 (Completeness)**.** *Given a parametric E-abstraction $(N_1, C_1) \preceq_E (N_2, C_2)$, then core requirements (Core 1), (Core 2), and (Core 3) are valid, and (Core 0) holds for both (N_1, C_1) and (N_2, C_2).*

Proof. By hypothesis, conditions (S1), (S2) and (S3) hold and (N_1, C_1) and (N_2, C_2) are coherent nets. Then, Lemma 7.5 implies that (Core 0) holds for both nets. Besides, Proposition 7.6 and Lemmas 7.7 and 7.8 ensure that (Core 1), (Core 2), and (Core 3) are valid. □

Consequently, checking E-abstraction equivalence, i.e., $(N_1, C_1) \approx_E (N_2, C_2)$, amounts to check that SMT formulas (Core 0)–(Core 3) are valid on both nets.

Our approach relies on our ability to express (arbitrarily long) sequences $m \overset{\epsilon}{\Rightarrow} m'$ thanks to a formula $\tau_C^*(\boldsymbol{p}, \boldsymbol{p'})$. This is addressed in the next section.

7.5 Accelerating the Silent Transition Relation

The previous results, including Theorems 7.9 and 7.10, rely on our ability to express the reachability set of silent transitions as a Presburger predicate, denoted τ_C^*. Finding a finite formula τ_C^* that captures an infinite state space is not granted since τ-sequences may be of arbitrary length. However, we now show that, since τ transitions must be abstracted away by E in order to define a valid parametric E-equivalence (condition (S2)), and since E is itself a Presburger formula, this implies that τ_C^* corresponds to the reachability set of a *flat* subnet [Ler13], which is expressible as a Presburger formula too.

We define the *silent reachability set* of a net N from a coherent constraint C as $R_\tau(N, C) \triangleq \{m' \mid m \models C \wedge m \overset{\epsilon}{\Rightarrow} m'\}$. We now want to find a predicate $\tau_C^*(\boldsymbol{p}, \boldsymbol{p'})$ that satisfies the relation:

$$R_\tau(N, C) = \{m' \mid m' \models \exists \boldsymbol{p} \,.\, C(\boldsymbol{p}) \wedge \tau_C^*(\boldsymbol{p}, \boldsymbol{p'})\} \tag{7.4}$$

In order to express the formula τ_C^*, we first use the tool FAST [Bar+03], designed for the analysis of infinite systems, and that permits to compute the reachability set of a given Vector Addition System with States (VASS). As mentioned in Sect. 1.4.3, a Petri net can be transformed to an equivalent VASS with the same reachability set. The algorithm implemented in FAST is a semi-decision procedure, for which we have some termination guarantees whenever the net is flat [Bar+05; Bar+08], i.e., its corresponding VASS can be unfolded into a VASS without nested cycles, called a flat VASS. Equivalently, a net N is flat for some coherent constraint C if its language is flat, that is, there exists some finite sequence $\varrho_1 \dots \varrho_k \in T^*$ such that for every initial marking $m \models C$ and reachable marking m' there is a sequence $\varrho \in \varrho_1^* \dots \varrho_k^*$ such that $m \xrightarrow{\varrho} m'$. In short, all reachable markings can be reached by simple sequences belonging to the language: $\varrho_1^* \dots \varrho_k^*$. Last but not least, the authors stated in [Ler13] that a net is flat if and only if its reachability set is Presburger-definable:

Theorem ([Ler13])**.** *For every VASS V, for every Presburger set C_{in} of configurations, the reachability set* $\mathrm{ReachV}(C_{in})$ *is Presburger if, and only if V is flatable from C_{in}.*

Consequently, FAST's algorithm terminates when its input is Presburger-definable. We show in Theorem 7.11 that given a parametric E-abstraction equivalence $(N_1, C_1) \approxeq_E (N_2, C_2)$, the silent reachability sets for both nets N_1 and N_2 with their coherency constraints C_1 and C_2 are indeed Presburger-definable—we can even provide the expected formulas. Nevertheless, our computation is complete only if the candidate reduction rule is a parametric E-abstraction equivalence (then, we are able to compute the τ_C^* relation), otherwise FAST, and therefore our procedure, too, may not terminate.

Theorem 7.11 (Silent Reachability Sets are Presburger-Definable)**.** *Given a parametric E-abstraction equivalence $(N_1, C_1) \approxeq_E (N_2, C_2)$, the silent reachability sets $R_\tau(N_1, C_1)$ and $R_\tau(N_2, C_2)$ are Presburger-definable.*

Proof. We prove only the result for (N_1, C_1), the proof for (N_2, C_2) is similar since $\approxeq$ is a symmetric relation. We first propose an expression that computes $R_\tau(N_1, m_1)$ for any marking m_1 satisfying C_1. Consider an initial marking m_1 in C_1. From condition (S1) (solvability of E), there exists a compatible marking m_2 satisfying C_2, meaning $m_1 \langle C_1 E C_2 \rangle m_2$ holds. Take a silent sequence $m_1 \overset{\epsilon}{\Rightarrow} m_1'$. From condition (S2) (silent stability), we have $m_1' \equiv_E m_2$. Hence, $R_\tau(N_1, m_1) \subseteq \{m_1' \mid \exists m_2 \,.\, C_2(m_2) \wedge \tilde{E}(m_1, m_2) \wedge \tilde{E}(m_1', m_2)\}$. Conversely, we show that all m_1' solution of $\tilde{E}(m_1', m_2)$ are reachable from m_1. Take m_1' such that $m_1' \equiv_E m_2$. Since we have $m_2 \overset{\epsilon}{\Rightarrow} m_2$, by condition (S3) we must have $m_1 \overset{\epsilon}{\Rightarrow} m_1'$. And finally we obtain $R_\tau(N_1, m_1) = \{m_1' \mid m_1' \models \exists \boldsymbol{p_1}, \boldsymbol{p_2} \,.\, \underline{m_1}(\boldsymbol{p_1}) \wedge C_2(\boldsymbol{p_2}) \wedge \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge \tilde{E}(\boldsymbol{p_1'}, \boldsymbol{p_2})\}$.

We can generalize this reachability set for all coherent markings satisfying C_1. We first recall its definition, $R_\tau(N_1, C_1) \triangleq \{m_1' \mid \exists m_1 \,.\, m_1 \models C_1 \wedge m_1 \overset{\epsilon}{\Rightarrow} m_1'\}$. From condition (S1), we can rewrite this set as $\{m_1' \mid \exists m_1, m_2 \,.\, m_1 \langle C_1 E C_2 \rangle m_2 \wedge m_1 \overset{\epsilon}{\Rightarrow} m_1'\}$ without losing any marking. Finally, thanks to the previous result we get $R_\tau(N_1, C_1) = \{m_1' \mid m_1' \models P\}$ with $P \triangleq \exists \boldsymbol{p_1}, \boldsymbol{p_2} \,.\, \langle C_1 E C_2 \rangle(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge \tilde{E}(\boldsymbol{p_1'}, \boldsymbol{p_2})$ a Presburger formula. Because of the E-abstraction equivalence, (S1) holds in both directions, which gives $\forall \boldsymbol{p_2} \,.\, C_2(\boldsymbol{p_2}) \implies \exists \boldsymbol{p_1} \,.\, \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge C_1(\boldsymbol{p_1})$. Hence, P can be simplified into $\exists \boldsymbol{p_2} \,.\, C_2(\boldsymbol{p_2}) \wedge \tilde{E}(\boldsymbol{p_1'}, \boldsymbol{p_2})$.

Note that this expression of $R_\tau(N, C)$ relies on the fact that the equivalence $(N_1, C_1) \approxeq_E (N_2, C_2)$ already holds. Thus, we cannot conclude that a candidate rule is an E-abstraction equivalence using this formula at once without the extra validation of FAST. □

Verifying FAST Results

We have shown that FAST terminates in case of a correct parametric E-abstraction. We now show that it is possible to check that the predicates $\tau_{C_1}^*$ and $\tau_{C_2}^*$, computed from the result of FAST (see Theorem 7.11) are indeed correct (if the equivalence holds).

Assume that τ_C^* is, according to FAST, equivalent to the language $\varrho_1^* \dots \varrho_n^*$ with $\varrho_i \in T^*$. We encode this language with the following Presburger predicate, which uses the formulas $H(\sigma^{k_i})$ and $\Delta(\sigma^{k_i})$ earlier in Chapter 2.

$$\tau^*_C(\boldsymbol{p^1}, \boldsymbol{p^{n+1}}) \triangleq \exists k_1 ... k_n, \boldsymbol{p^2} \dots \boldsymbol{p^{n-1}} \;.\; \bigwedge_{i \in 1..n} \left((\boldsymbol{p^i} \geqslant H(\varrho_i^{k_i})) \wedge \Delta(\varrho_i^{k_i})(\boldsymbol{p^i}, \boldsymbol{p^{i+1}}) \right) \tag{7.8}$$

This definition introduces *acceleration* variables k_i (that we called saturation variables in Chapter 2), encoding the number of times we fire the sequence ϱ_i. The hurdle and delta of the sequence of transitions ϱ_i^k, which depends on k, are written $H(\varrho_i^k)$ and $\Delta(\varrho_i^k)$, respectively. Their formulas are given in Equations (7.9) and (7.10) below, where $\mathbb{1}_{>0}(k) \triangleq 1$ if and only if $k > 0$, and 0 otherwise.

$$H(\varrho^k) \triangleq \mathbb{1}_{>0}(k) \times (H(\varrho) + (k-1) \times (-\Delta(\varrho))^+) \tag{7.9}$$

$$\Delta(\varrho^k) \triangleq k \times \Delta(\varrho) \tag{7.10}$$

Note that the definition differs slightly from that in Chapter 2. Here we also encode the possibility of not firing the sequence ϱ (hence the adding of $\mathbb{1}_{>0}$).

Finally, given a parametric rule $(N_1, C_1) >_E (N_2, C_2)$ we can now check that the reachability expression $\tau^*_{C_1}$ provided by FAST, and encoded as explained above, corresponds to the solutions of $\exists \boldsymbol{p_2} \;.\; \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2})$ using the following additional SMT query:

$$\forall \boldsymbol{p_1}, \boldsymbol{p_1'} \;.\; C_1(\boldsymbol{p_1}) \implies (\exists \boldsymbol{p_2} \;.\; \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge \tilde{E}(\boldsymbol{p_1'}, \boldsymbol{p_2}) \iff \tau^*_{C_1}(\boldsymbol{p_1}, \boldsymbol{p_1'})) \tag{7.11}$$

(and similarly for $\tau^*_{C_2}$).

Once the equivalence Equation (7.11) above has been validated by a solver, it is, in practice, way more efficient to use the formula $\exists \boldsymbol{p_2} \;.\; \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge \tilde{E}(\boldsymbol{p_1'}, \boldsymbol{p_2})$ inside the core requirements, rather than the formula $\tau^*_{C_1}(\boldsymbol{p_1}, \boldsymbol{p_1'})$ given by FAST, since the latter introduces many new acceleration variables.

7.6 Decidability

Even if our method may not terminate, since FAST is only a semi-decision procedure we can prove that checking the correctness of parametric E-abstraction is decidable.

Theorem 7.12 (Checking Parametric E-abstraction is Decidable)**.** *Given two nets N_1, N_2 and constraints C_1, C_2 expressed as Presburger formulas. The problem of deciding whether the statement $(N_1, C_1) \approxeq_E (N_2, C_2)$ holds is decidable.*

Proof. We proved in Theorems 7.9 and 7.10 that the statement $(N_1, C_1) \approxeq_E (N_2, C_2)$ holds if and only if (Core 0) is valid for both nets (N_1, C_1) and (N_2, C_2) and core requirements

(Core 1), (Core 2), and (Core 3) are valid (in both ways). Furthermore, checking the truth of Presburger formulas is decidable [PJ91].

We are left to prove that we can construct these formulas. The crux relies on the computation of predicates $\tau^*_{C_1}$ and $\tau^*_{C_2}$. We proved in Theorem 7.11 a necessary condition to have a correct equivalence, that is, $R_\tau(N_1, C_1)$ and $R_\tau(N_2, C_2)$ must be Presburger-definable. The problem of deciding if the reachability set of a general Petri net from an initial Presburger set of markings is Presburger (equivalently semilinear [GS66]) is decidable [Hau90; Lam90]. Then, if either $R_\tau(N_1, C_1)$ or $R_\tau(N_2, C_2)$ is not Presburger-definable we can assert that the equivalence does not hold; without constructing the core requirements. Otherwise, the net is flat [Ler13]; and computing $\tau^*_{C_1}$ and $\tau^*_{C_2}$ is also decidable [FL02].

Hence, we proposed a theoretical procedure to answer the problem of deciding a parametric E-equivalence holds, where all steps are decidable. □

7.7 Checking the State Space Partition

We finally propose to check whether a statement provides a state space partition—that is not entailed by the parametric E-equivalence (refer to Sect. 4.5.3)—since the relation is symmetric—by verifying two additional core requirements ((Core 4) and (Core 5)) on the reduced net N_2. We already proved in Theorem 4.8 that any equivalence admitting a well-formed Token Flow Graph forms a partition. It is important to emphasize that the state space partition is not a prerequisite for solving reachability problems mentioned in this thesis. Nevertheless, it is a requirement for some model counting methods, for which polyhedral reduction were initially developed [BLD18; BLD19].

$$\forall \boldsymbol{p_2}, \boldsymbol{p'_2} \,.\, C_2(\boldsymbol{p_2}) \wedge \tau(\boldsymbol{p_2}, \boldsymbol{p'_2}) \implies \mathrm{EQ}(\boldsymbol{p_2}, \boldsymbol{p'_2}) \tag{Core 4}$$

$$\forall \boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p'_2} \,.\, C_2(\boldsymbol{p_2}) \wedge C_2(\boldsymbol{p'_2}) \wedge \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p_2}) \wedge \tilde{E}(\boldsymbol{p_1}, \boldsymbol{p'_2}) \implies \mathrm{EQ}(\boldsymbol{p_2}, \boldsymbol{p'_2}) \tag{Core 5}$$

Given a marking m'_2 of the reduced net N_2, we define $\mathrm{Inv}_E(m'_2)$ as the set of markings of the initial net N_1 related to m'_2.

$$\mathrm{Inv}_E(m'_2) \triangleq \{m'_1 \mid m'_1 \equiv_E m'_2\} \tag{7.12}$$

Theorem 7.13 (Checking State Space Partition)**.** *Given* $(N_1, C_1) \approxeq_E (N_2, C_2)$. *The family of sets* $S \triangleq \{\mathrm{Inv}_E(m'_2) \mid m'_2 \in R(N_2, C_2)\}$ *is a partition of* $R(N_1, C_1)$ *if and only (Core 4) and (Core 5) are valid.*

Proof. The set S is a partition as a consequence of the following points:

No empty set in S. For any marking m_2' in $R(N_2, C_2)$ there exists some marking m_2 and sequence σ such that $m_2 \models C_2$ and $m_2 \stackrel{\sigma}{\Rightarrow} m_2'$. By condition (S1) of the parametric E-abstraction, there is some marking m_1 such that $m_1 \langle C_1 E C_2 \rangle m_2$. From Theorem 7.1, we have $(N_1, m_1) \equiv_E (N_2, m_2)$. Now, by condition (A2) of the E-abstraction (Definition 3.2), there is some m_1' such that $m_1' \equiv_E m_2'$. Thus, $\mathrm{Inv}_E(m_2')$ is not empty. This implies $\emptyset \notin S$.

The union $\cup_{A \in S} A$ is equal to $R(N_1, C_1)$. We prove both inclusions separately.

- Take a marking m_1' in $R(N_1, C_1)$. As previously, we still have some markings $m_1 \models C_1$ and $m_2 \models C_2$ such that $(N_1, m_1) \equiv_E (N_2, m_2)$ and $m_1' \in R(N_1, m_1)$ (by condition (S1) and Theorem 7.1). By condition (A2) of the E-abstraction, there is some marking m_2' such that $m_2' \in R(N_2, m_2)$ and $m_1' \equiv_E m_2'$. Hence, there is some set $A \in S$ such that $m_1' \in A$ and so $R(N_1, C_1) \subseteq \cup_{A \in S} A$.
- Now take a set A in S and a marking $m_1' \in A$. By construction, there is some marking m_2' in $R(N_2, C_2)$ such that $m_1' \equiv_E m_2'$. By condition (S1) and Theorem 7.1, there is $(N_1, m_1) \equiv_E (N_2, m_2)$ such that $m_1 \models C_1$, $m_2 \models C_2$ and $m_2' \in R(N_2, m_2)$. By condition (A2) of Definition 3.2 we have $m_1' \in R(N_1, m_1)$. Hence, $m_1' \in R(N_1, C_1)$ and so $\cup_{A \in S} A \subseteq R(N_1, C_1)$.

Pairwise disjoint. Take two different markings m_2' and m_2'' in $R(N_2, C_2)$. Since (N_2, C_2) is a coherent net, we can find some initial and intermediate markings such that $m_2^{(1)} \to m_2^{(2)} \stackrel{\epsilon}{\Rightarrow} m_2'$ and $m_2^{(3)} \to m_2^{(4)} \stackrel{\epsilon}{\Rightarrow} m_2''$ with $m_2^{(i)} \models C_2$ for all i in 1..4. And since (Core 4) is valid, we have $m_2' = m_2^{(2)}$ and $m_2'' = m_2^{(4)}$ (firing silent transitions from a coherent state do not change the marking). Hence, we get $m_2' \models C_2$ (i) and $m_2'' \models C_2$ (ii).

Now, we prove by contradiction that $\mathrm{Inv}_E(m_2') \cap \mathrm{Inv}_E(m_2'') = \emptyset$. Suppose that $\mathrm{Inv}_E(m_2') \cap \mathrm{Inv}_E(m_2'')$ is not empty and take a marking m_1 from it. Hence, $m_1 \equiv_E m_2'$ and $m_1 \equiv_E m_2''$. From (i) and (ii), and the hypothesis $m_2' \not\equiv m_2''$, we contradict the validity of (Core 5).

We are left to prove that the validity of (Core 4) and (Core 5) is a necessary condition to obtain such partition. Assume that S is a partition of $R(N_1, C_1)$.

- Suppose that (Core 4) is not valid. Then, there is a pair of different markings m_2, m_2' of N_2 such that $m_2 \models C_2$ and $m_2 \stackrel{\epsilon}{\Rightarrow} m_2'$. From condition (S1) there is some marking m_1 such that $m_1 \equiv_E m_2$, and by condition (S2) we also have $m_1 \equiv_E m_2'$. Then, there are two sets A and A' in S such that $m_1 \in A$ and $m_1 \in A'$, which contradicts that sets in S are pairwise disjoint.
- Suppose that (Core 5) is not valid. Then, there are some markings m_1 of N_1 and m_2, m_2' of N_2 such that $m_2 \models C_2$, $m_2' \models C_2$, $m_1 \equiv_E m_2$, $m_1 \equiv_E m_2'$ and $m_2 \not\equiv m_2'$. By

construction of S, we can find some sets A, A' in S, such that $m_1 \in A$ and $m_1 \in A'$, which also contradicts that sets in S are pairwise disjoint.

□

7.8 Generalizing Equivalence Rules

Before looking at our implementation, we discuss some results related to the *genericity* and *generalizability* of our reduction rules. We consider several "dimensions" in which a rule can be generalized. A first dimension is related to the parametricity of the initial marking, which is considered by our use of a parametric equivalence, $\approxeq$ instead of $\equiv$, see Theorem 7.1. Next, we show that we can infer an infinite number of equivalences from a single reduction rule using compositionality, transitivity, and structural modifications involving labels. Therefore, each reduction rule can be interpreted as a schema for several equivalence rules.

Definition 7.4 (Transition Operations)**.** *Given a Petri net* $N \triangleq (P, T, \mathrm{Pre}, \mathrm{Post})$ *and its labeling function* $l : T \to \Sigma \cup \{\tau\}$*, we define two operations:* T^-*, for removing, and* T^+*, for duplicating transitions. Let* a *and* b *be labels in* Σ*.*

- $T^-(a)$ *is a net* $(P, T', \mathrm{Pre}', \mathrm{Post}')$*, where* $T' \triangleq T \setminus l^{-1}(a)$*, and* Pre' *(resp.* Post'*) is the projection of* Pre *(resp.* Post*) to the domain* T'*.*
- $T^+(a, b)$ *is a net* $(P, T', \mathrm{Pre}', \mathrm{Post}')$*, where* T' *is a subset of* $T \times \{0, 1\}$ *defined by* $T' \triangleq T \times \{0\} \cup l^{-1}(a) \times \{1\}$*. Additionally, we define* $\mathrm{Pre}'(t, i) \triangleq \mathrm{Pre}(t)$ *and* $\mathrm{Post}'(t, i) \triangleq \mathrm{Post}(t)$ *for all* $t \in T$ *and* $i \in \{0, 1\}$*. Finally, the labeling function* l' *is defined with* $l'(t, 0) \triangleq l(t)$ *and* $l'(t, 1) \triangleq b$ *for all* $t \in T$*.*

The operation $T^-(a)$ removes transitions labeled by a, while $T^+(a, b)$ duplicates all transitions labeled by a and labels the copies with b. We illustrated T^+ in the nets of rule [MAGIC], in Fig. 7.2, page 160 where the "blue dashed" transition c' can be interpreted as the result of applying operation $T^+(c, c')$. Note that these operations only involve labeled transitions. Silent transitions are kept untouched.

Theorem 7.14 (Preservation by Transition Operations)**.** *Assume that* $(N_1, C_1) \approxeq_E (N_2, C_2)$ *is a parametric* E*-abstraction equivalence, and that* a *and* b *are labels in* Σ*. Then,*

- $T_i^-(a)$ *and* $T_i^+(a, b)$ *satisfy the coherency constraint* C_i*, for* $i = 1, 2$*;*
- $(T_1^-(a), C_1) \approxeq_E (T_2^-(a), C_2)$*;*
- $(T_1^+(a, b), C_1) \approxeq_E (T_2^+(a, b), C_2)$*.*

where T_i^-*,* T_i^+ *is (respectively) the operation* T^-*,* T^+ *on* N_i*.*

Proof. We assume that $(N_1, C_1) \approx_E (N_2, C_2)$ (i) holds, which implies that N_1 satisfies the coherency constraint C_1 (resp., N_2 satisfies C_2). For each operation T^-, T^+, we show that the transformed nets N_1' and N_2' still satisfy the coherency constraints and that the conditions (S1), (S2), (S3) of Definition 7.3 still hold. Conditions (S1) and (S2) do not involve labeled transitions, so they immediately hold in N_1' and N_2'. (S3) is proven by considering each operation separately.

- Case $T^-(a)$: N_1' (resp. N_2') is N_1 (resp. N_2) without transitions labeled by a. Assume that $(N_1', m_1) \stackrel{\sigma}{\Rightarrow} (N_1', m_1')$ holds (hence, $a \notin \sigma$). From (i), for all markings m_2, m_2', such that $m_1 \langle C_1 E C_2 \rangle m_2 \wedge m_1' \equiv_E m_2'$, we have $(N_2, m_2) \stackrel{\sigma}{\Rightarrow} (N_2, m_2')$. Hence, $(N_2', m_2) \stackrel{\sigma}{\Rightarrow} (N_2', m_2')$ holds since $a \notin \sigma$.
- Case $T^+(a, b)$: N_1' (resp. N_2') is N_1 (resp. N_2) with transitions labeled by a duplicated and duplicates are labeled by b. Assume that $(N_1', m_1) \stackrel{\sigma}{\Rightarrow} (N_1', m_1')$ holds. Let σ_a be $\sigma\{b \leftarrow a\}$. Then, we have $(N_1, m_1) \stackrel{\sigma_a}{\Longrightarrow} (N_1, m_1')$. From (i), for all markings m_2, m_2', such that $m_1 \langle C_1 E C_2 \rangle m_2 \wedge m_1' \equiv_E m_2'$, we have $(N_2, m_2) \stackrel{\sigma_a}{\Longrightarrow} (N_2, m_2')$. Then, $(N_2', m_2) \stackrel{\sigma_a}{\Longrightarrow} (N_2', m_2')$ holds since transitions of N_2 are included in those of N_2'. In N_2', each transition labeled by a is identical to a twin transition labeled by b. Hence, any such transition can be freely replaced by its twin. Therefore, $(N_2', m_2) \stackrel{\sigma}{\Rightarrow} (N_2', m_2')$ also holds. This concludes the case.

The proof that net N_1' (resp. N_2') still satisfies the coherency constraint C_1 (resp. C_2) is also done by considering each operation separately, and is actually very similar to the above cases (we omit the details). The three conditions (S1), (S2), (S3) hold on N_1' and N_2', thus $(N_1', C_1) \approx_E (N_2', C_2)$ is shown. □

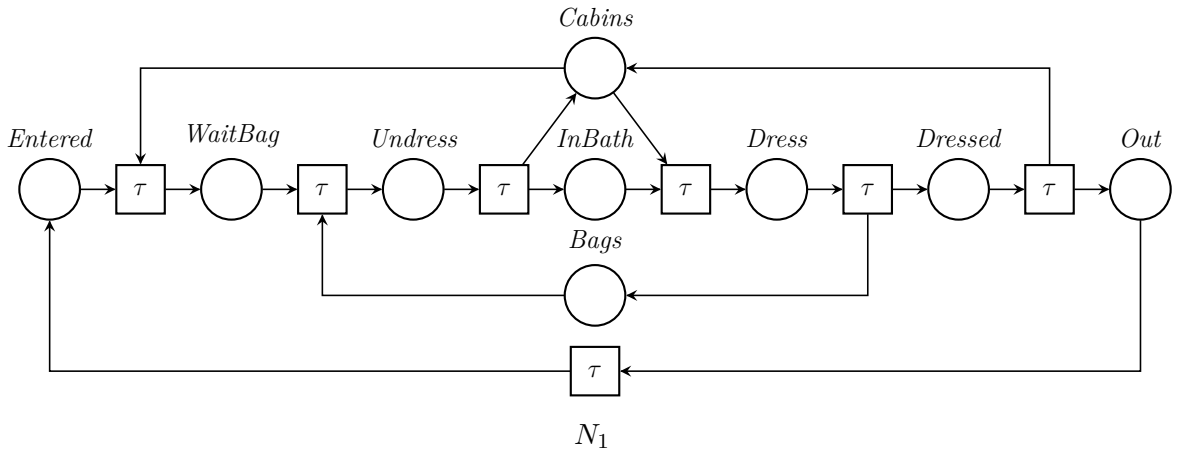
Finally, we recall previous results from Chapter 3, which states that equivalence rules can be combined using synchronous composition (Theorem 3.8), relabeling (Theorems 3.9), and chaining (Theorem 3.6).

Theorem (E-equivalence is a Congruence)**.** *Assume that* $(N_1, m_1) \equiv_E (N_2, m_2)$ *and* $(N_2, m_2) \equiv_{E'} (N_3, m_3)$ *are two compatible equivalence statements, and that* M *is a Petri net such that* $N_1 \parallel M$ *and* $N_2 \parallel M$ *are defined, then*

- $(N_1, m_1) \parallel (M, m) \equiv_E (N_2, m_2) \parallel (M, m)$.
- $(N_1, m_1) \equiv_{\exists P_2 \setminus (P_1 \cup P_3).E \wedge E'} (N_3, m_3)$.
- $(N_1[a/b], m_1) \equiv_E (N_2[a/b], m_2)$ *for any* $a \in \Sigma$ *and* $b \in \Sigma \cup \{\tau\}$.

7.9 Experimental Validation

We have implemented our automated procedure in a new tool called Reductron. The tool is open-source, under the GPLv3 license, and is freely available on GitHub [Ama23d]. The repository contains a subdirectory, `rules`, that provides examples of equivalence rules that can be checked



$$C_1 \triangleq Cabins = 10 \wedge Out = 20 \wedge Bags = 15 \wedge$$
$$Entered + WaitingBag + Undress + Dress + Inbath + Dressed = 0$$

$$E \triangleq \begin{cases} Cabins + Dress + Dressed + Undress + WaitBag = 10 \\ Dress + Dressed + Entered + InBath + Out + Undress + WaitBag = 20 \\ Bags + Dress + InBath + Undress = 15 \end{cases}$$

Fig. 7.8 A Petri net modeling users in a swimming pool, see e.g. [BF99].

using our approach. Each test contains two Petri nets, one for N_1 (called `initial.net`) and another for N_2 (called `reduced.net`), defined using the syntax of Tina [BRV04; LAA23]. These nets also include declarations for constraints, C_1 and C_2, and for the equation system E. Our list contains examples of rules that are implemented in Tedd and SMPT, such as rule [CONCAT] depicted in Fig. 7.1, but also some examples of unsound equivalences rules. For instance, we provide example [FAKECONCAT], which corresponds to the example of Fig. 7.1 with transition d added.

When a rule is unsound, an interesting feature of Reductron is to return which core requirement failed. For instance, with [FAKECONCAT], we learn that (N_1, C_1) is not coherent because of d (we cannot reach a coherent marking after firing d using only silent transitions). We can also detect many cases with an error in the specification of either C or E.

We performed some experimentation using z3 [MB08] (version 4.8) as our target SMT solver and FAST (version 2.1). Our repository's examples can be solved in a few seconds. Although we focus on automatically verifying reduction rules, we have also tested our tool on moderate-sized nets, such as the swimming pool example in Fig. 7.8. In this context, we use the fact that an equivalence of the form $(N, C) \cong_E (\emptyset, \text{True})$, between N and a net containing an empty set of places, entails that the reachability set of (N, C) must be equal to the solution set of E. In this case, also, the results are almost immediate.

These excellent results depend mainly on the continuous improvements made by SMT solvers. Indeed, we generate very large LIA formulas, with sometimes hundreds of quantified variables and a moderate amount of quantifier alternation (formulas of the form $\forall\exists\forall$). For instance, experiments performed with older versions of z3 (such as 4.4.1, October 2015) exhibit significantly degraded performances.

7.10 Discussion

This chapter concludes the theoretical contributions of this thesis and aims to improve the safety of our polyhedral reduction framework using automated reasoning techniques instead of relying only on "manual theorem proving". But the result we find the most interesting is the fact that it enhances our understanding of the theoretical underpinnings of polyhedral equivalence and its close relation to the notion of flat nets. It also underlines the importance of coherency constraints, which takes a central role in our definition of a parametric version of polyhedral equivalence. We also hope that it helps better understand how to construct new reduction rules in the future.

There is still ample room to study polyhedral reduction. For instance, we are interested in characterizing Petri nets that are *fully reducible*, but where E is a "convex" predicate (to ensure that the equivalence defines a partition of the state space). This defines an interesting and non-trivial subset of flat nets.

Finally, we exhibited a concrete use case for the problem of deciding whether the state space of a given Petri net is Presburger-definable. This result can be found in two different works [Hau90; Lam90], with proofs that do not easily translate into practical algorithms. We believe that it would be worthwhile to revisit this problem.

This work has been published in:

- N. Amat, S. Dal Zilio, and D. Le Botlan. "Automated Polyhedral Abstraction Proving". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. vol. 13929. Lecture Notes in Computer Science. Springer, 2023. DOI: 10.1007/978-3-031-33620-1_18

The tool related to this chapter is:

- Reductron https://github.com/nicolasAmat/Reductron

Chapter 8

Tools and Reproducibility

The Experimental Counterpart

> Talk is cheap. Show me the code.
>
> Linus Torvalds

This chapter is the experimental counterpart of the previous chapters, which are more theoretical in nature. We give a thorough presentation of our tools, the benchmark suite from the Model Checking Contest used in our experiments, and explain how to reproduce our experiments using the accompanying artifact. It is also an opportunity to take stock of our three participations in the Model Checking Contest.

8.1 Experimental Benchmark

For most of our experiments we used the extensive, and independently managed, set of models and formulas collected from the 2023 edition of the Model Checking Contest (MCC) [Kor+23].

The Model Checking Contest was a key player during my PhD thesis. It is an annual and international competition for model checking tools. The contest is divided in different examinations: state space generation, computation of global properties, computation of queries regarding the upper bounds of markings, evaluation of reachability formulas, evaluation of CTL formulas, and evaluation of LTL formulas. The objective of the contest is to provide a common benchmark, increasing each year with new submissions, and compare model checking approaches in an open and independent way.

Participating in the reachability examination has been a motivation from the start; and the resulting benchmark was an incredible help in carrying out experiments with models from both academia and industry.

Models. The benchmark is built from a collection of 132 models. Most of the models are parametrized, and therefore there can be several *instances* for the same model. This amounts to 1 678 different instances of Petri nets whose size varies widely. The larger models contain up to 10^5 places, 10^6 transitions, and few million arcs. Most nets are ordinary (non-zero weights on all arcs are equal to 1), but a significant number of instances (about 380) use weighted arcs.

The MCC benchmark is composed of both (standard) Petri nets and colored Petri nets (see Sect. 1.6). However, the MCC provides equivalent Petri nets for colored specifications, except for some instances that cannot be unfolded due to an explosion in the size of the resulting net. Then, the total number of standard Petri net instances amounts to 1 426. Since this thesis only focuses on standard Petri nets, our benchmark is built on this subset of instances.

Overall, the collection provides numerous examples with various structural and behavioral characteristics, covering a large variety of use cases. We display in Table 8.1 (page 193) a classification of the models along their application domain, provided by the MCC model board in [Ama+23].

Formulas. Each year, 32 reachability formulas are randomly generated using the tool Citili[1]. Formulas are divided into two categories of equal sizes: (1) *cardinality queries*, 16 formulas that are linear predicates over the places (as defined in Sect. 1.3); and *fireability queries*, 16 formulas that are Boolean combinations of fireability predicates (quasi-liveness). Note that fireability formulas can be converted into formulas that only deal with places, by expanding the pre-conditions of the transitions.

In our experiments, we focus on the *cardinality* category, and we kept the most recent formulas from the 2023 edition. This amounts to 22 816 pairs consisting of a model instance and a cardinality formula; what we call *queries* in the thesis.

Input formats. The format for nets is the Petri Net Markup Language (PNML) [Hil+10], an XML-based language. Reachability formulas are expressed using the MCC property language [JP20], which is also XML-based. Cardinality formulas only represent a subclass of the properties described in Sect. 1.3. Basically, the property language only supports the encoding of Boolean combinations of inequalities of the form $\alpha \leqslant \beta$, where α and β are either sums of places, or integer constants. Hence, it is not possible to mix sums of variables and integer constants in the literals α, β or use negative coefficients. As a consequence, it is not possible to express conditions like $p < q$ (since it is equivalent to $p \leqslant q + 1$), or $p + q \leqslant r + 1$. However, most tools participating in the contest, such as ITS-Tools, TAPAAL and LoLA, support the full fragment of quantifier-free Presburger formulas.

[1] https://github.com/mcc-petrinets/citili

8.2 Tools for Computing Polyhedral Reductions

In our approach, the computation of structural reductions is delegated to a separate tool. We propose two possibilities for this task.

First, the tool Reduce, which is a new addition to the Tina model checking toolbox [BRV04; LAA23] since version 3.7. It is currently used by the Tedd model checker that competes in the Model Checking Contest (MCC), on the state space examination.

We can also rely on a new open-source tool (and a Rust crate) [Cha22], called Shrink, developed by Louis Chauvet, an intern I supervised. This is a highly customizable tool, and also a library, that we hope can be reused and improved in other contexts.

Both tools implement the set of reduction rules described in [BLD18; BLD19]. A particularity of Reduce is to provide additional (and more sophisticated) reductions, for instance the detection of nets with the PR-R property [Huj+20a; Huj+20b], see Sect. 1.5.4, meaning nets such that the set of reachable markings equals the set of potentially reachable ones.

8.3 SMPT: Satisfiability Modulo Petri Nets

SMPT [Ama20b] is a model checker for reachability problems in Petri nets. The tool name is an acronym that stands for *Satisfiability Modulo Petri Net*. This choice underlines the fact that, for most of the new features we implemented, it acts as a front-end to an SMT solver; but also that it adds specific knowledge from Petri net theory, such as invariants, use of structural properties, etc.

SMPT started as a portfolio of methods to experiment with symbolic model checking and was designed to be easily extended. Some distinctive features are its ability to benefit from polyhedral reductions and to generate verdict certificates. In Sect. 8.7 we show that the tool is quite mature and performed well compared to other state-of-the-art tools in the Model Checking Contest.

General Presentation

SMPT is an open-source model checker designed to answer reachability queries (in the form of linear predicates). It does not impose any restrictions on the marking of places or the weight on the arcs and can in particular handle unbounded nets.

The design of SMPT reflects the two main phases during its development process. The tool was initially developed as a testbed for symbolic model checking algorithms that can take advantage of polyhedral reductions, see Chapters 3 and 5. This explains why it includes many "reference" implementations of fundamental reachability algorithms, tailored for Petri nets, such as Bounded Model Checking (BMC) or k-induction. It also includes our implementation of Property Directed Reachability (PDR) from Chapter 2. One of our goals is to efficiently compare different algorithms, on a level playing field, with the ability to switch on or off

optimizations. This motivates our choice to build a tool that is highly customizable and easily extensible.

In a second phase, since 2021, we worked to make SMPT more mature, with the goal of improving its interoperability, and with the addition of new verification methods that handle problems where symbolic methods are not the best suited. Following we discuss the portfolio approach implemented in SMPT.

Other tools perform similar tasks. We provide a brief comparison of SMPT with two other model checkers in Sect. 8.7, namely ITS-Tools [Thi15] and TAPAAL [Dav+12]. These tools have in common their participation in the MCC and the use of symbolic techniques. They also share common input formats for nets and formulas. We can offer two reasons for users to use SMPT instead of—or more logically in addition to—these tools. First, SMPT takes advantage of our *polyhedral reduction*, to accelerate the verification of reachability properties. This approach can be extremely effective in some cases where other methods do not scale. Another interesting feature of SMPT is the ability to return a *verdict certificate*. When a property is invariant, we can return a "proof" that can be checked independently by an SMT solver.

Design and Implementation

SMPT is open-source, under the GNU GPL v3 license, and is freely available on GitHub [Ama20b]. It is a Python project of about 4 500 lines of code, and is fully typed using the static type checker mypy. The code is heavily documented (5 000 lines) and we provide many tracing and debugging options that can help understand its inner workings. The project is packaged in libraries and provides abstract classes to help with future extensions. Following we describe each library.

The ptio library defines the main data structures of the model checker, for Petri nets (`pt.py`), reachability formulas (`formula.py`), and reduction equations (`system.py`). It also provides the corresponding parsers, for different formats.

The interface library includes interfaces to external tools and solvers. For example, we provide an integrated interface to z3 [MB08; Bjø] built around the SMT-LIB format [BFT17]. We can also interface with MiniZinc [Net+07], a solver based on constraint programming techniques, and with a random state space explorer, Walk, distributed with the Tina toolbox [BRV04; LAA23]. New tools can be added by implementing the abstract class `Solver` (`solver.py`).

The exec library provides a concurrent "jobs scheduler" that helps run multiple verification tasks in parallel and manage their interactions.

The checker library is the core of our tool. It includes a portfolio of methods intended to be executed in parallel. All methods implement an abstract class (`abstractchecker.py`) which describes the abstract method `prove`. We currently support the following eight methods:

1. **Induction**: (Sect. 1.5.3): a basic method that checks if a property is an inductive invariant. This property is "easy" to check, even though interesting properties are seldom inductive. It is also useful to check verdict certificates.
2. **BMC** (Sect. 1.5.2): Bounded Model Checking [Bie+99] is an iterative method to explore the state space of systems by unrolling their transitions. This method is only useful for finding counter-examples.
3. **k-Induction** (Sect. 1.5.3): is an extension of BMC that can also prove invariants [SSS00].
4. **PDR** (Chapter 2): Property Directed Reachability [Bra11; Bra12], also known as IC3, is a method to strengthen a property that is not inductive, into an inductive one. This method can return a verdict certificate. As mentioned, we provide three different methods of increasing complexity (one for coverability and two for general reachability).
5. **State Equation** (Sect. 1.5.4): is a method for checking that a property is true for all "potentially reachable markings" (solution of the state equation). This is a semi-decision method, found in many portfolio tools, that can easily check for invariants. We implement a refined version [Thi15; Thi20] that can over-approximate the result with the help of trap constraints [EM00] and other structural information, such as NUPN specifications [Gar19].
6. **Random Walk** (Sect. 1.5.1): relies on simulation tools to quickly find counter-examples. It is also found in many tools that participate in the MCC [Kor+21b]. We currently use Walk, distributed with the Tina toolbox [BRV04; LAA23].
7. **Constraint Programming**: is a method specific to SMPT in the case where nets are "fully reducible" (the reduced net has only one marking). In this case, reachable markings are exactly the solution of the reduction equations (E) and verdicts are computed by solving linear system of equations.
8. **Enumeration**: performs an exhaustive exploration of the state space and relies on the Tina model checker [BRV04; LAA23]. It can be used as a fail-safe, or to check the reliability of our results.

Commands, Basic Usage and Installation

SMPT requires Python version 3.7 or higher. It also requires the sexpdata Python library to parse some outputs from z3. The easiest method for experimenting with the tool is to directly run the `smpt` module as a script, using a command such as `python3 -m smpt`. Our repository includes a script to simplify the installation of the tool and all its dependencies. It is also possible to find disk images with a running installation in the MCC website and in artifacts archived on Zenodo [ADH22a; AD22; ADL23a]. As usual, option `--help` returns an abridged description of all the available options. We list some of them below, grouped by usage.

Input formats. We accept Petri nets described using the Petri Net Markup Language (PNML) [Hil+10] and can also support colored Petri nets (using option `--colored`) by using the external unfolder `mcc` [Dal20]. For methods that rely on polyhedral reductions, it is possible to automatically compute the reduction (`--auto-reduce`) or to provide a pre-computed version (with option `--reduced-net <path>`). It is also possible to save a copy of the reduced net with the option `--save-reduced-net <path>`. Finally, one can use the option `--project` to enable our projection method defined in Chapter 5.

Verification methods. We support the verification of three predefined classes of reachability properties: *deadlock detection* (`--deadlock`), which is self-descriptive; *quasi-liveness* (`--quasi-liveness t`), to check if it is possible to fire transition `t`; and *reachability* (`--reachability p`), to check if there is a reachable marking where place `p` is marked (it has at least one token). It is also possible to check the reachability of several places, at once, by passing a comma-separated list of names, `--reachability p1,...,pn`; and similarly for quasi-liveness. Finally, SMPT supports properties expressed using the MCC property language [JP20], in XML format. Several properties can be checked at once.

Output format. Results are printed in the format required by the MCC, which is a single line of text of the form `FORMULA <id> (TRUE/FALSE)` for each answered query. There are also options to output more information: `--debug` to print the SMT-LIB input/output code exchanged with the SMT solver; `--show-techniques`, to return the methods that successfully computed a verdict; `--show-time`, to print the execution time per property; `--show-reduction-ratio`, to get the reduction ratio; `--show-model`, to print the counter-example if it exists; `--check-proof`, to check verdict certificates (when we have one); `--export-proof`, to export verdict certificates (inductive invariants, traces leading to counter-examples, etc.).

Tweaking options We provide a set of options to control the behavior of our verification jobs scheduler. We can add a timeout, globally (`--global-timeout <int>`) or per property (`--timeout <int>`). We can also restrict the choice of verification methods (`--methods <method_1> .... <method_n>`). Finally, option `--mcc` puts the tool in "competition mode".

8.4 Kong: The Koncurrent Places Grinder

Kong [Ama20a], the *Koncurrent places Grinder*, is a formal verification tool for Petri nets that can take advantage of polyhedral reductions to accelerate the verification of specific reachability properties: checking whether a given marking is reachable; computing the concurrency relation; of finding the dead places of a net.

Commands, Basic Usage and Installation

Kong is an open-source tool, under the GNU GPL v3 license, made freely available on GitHub [Ama20a]. The project has about 1 000 lines of code.

Dependencies. Kong is written in Python and requires a version 3.5 or higher. It also requires the graphviz Python library in order to output a graphical description of Token Flow Graphs (optional). Kong is intended to be as understandable as possible; the code is heavily documented, and we provide many tracing and debugging options that can help understand its inner workings.

We support two different tools to compute polyhedral reductions, Reduce and Shrink, that both use the same input and output formats. Kong runs Reduce if the executable is in the current `PATH` environment variable, but automatically switches to Shrink otherwise. It is still possible to enforce the use of Shrink by using the `--shrink` option. As with SMPT, it is also possible to directly provide a pre-computed result of structural reductions with the option `--reduced-net`.

Kong is a command-line tool organized into subcommands that expose its different features. The tool provides several options that are described in the documentation using `--help`. We give a brief description of some of them in the following.

Concurrent and Dead Places. The main subcommands of Kong are `conc` and `dead` for, respectively, computing the concurrent relation and the list of dead places in a net. When computing a concurrency matrix, Kong relies on an external tool to compute the concurrency matrix of the reduced net. This is currently done using Cæsar.BDD, part of the CADP toolbox [BG21; INR], which is the state-of-the-art tool for the concurrent places problem.

Kong takes as inputs ordinary, safe Petri nets defined using either the Petri Net Markup Language (PNML) [Hil+10], or the Nested-Unit Petri Net (NUPN) format [Gar19]. (The file format is automatically detected from the file extension.) The use of a NUPN decomposition, which provides information about the concurrent structure of the net, can bring a significant performance improvement. The tool was designed to be fully compatible with Petri net instances used in the MCC. For instance, we can make use of NUPN information added to a PNML model using its tool-specific extension mechanism.

Kong can be executed as a Python script or converted into a standalone executable using `cx_Freeze`. Each subcommand only requires the path to the input Petri net (with a `.pnml` or `.nupn` extension). Hence, a typical call to Kong is of the form `./kong.py conc model.pnml`. We also provide two main options to limit the exploration performed by Cæsar.BDD: `--bdd-timeout` to set a time limit and `--bdd-iterations` to limit the number of iterations.

Our output format for the concurrency matrix is the same as the one of Cæsar.BDD. We can output our results using a compressed format, based on a run-length encoding (RLE) of the rows of C. For the sake of readability, it is possible to disable this encoding using option `--no-rle`. It is also possible to print the place ordering with option `--place-names`.

A call to `kong.py conc` delegates the computation of the concurrent relation on the reduced net to the tool Cæsar.BDD. It can also take as input a precomputed concurrency matrix of the reduced net, using option `--reduced-matrix`. Likewise, the `dead` subcommand provides option `--reduced-vector` if we have a precomputed list of dead places for the reduced net. It is also possible to output the matrix computed by Cæsar.BDD with option `--show-reduced-matrix` (resp. `--show-reduced-vector` if we use subcommand `dead`).

Marking Reachability. The `reach` subcommand provides a procedure to check if a given marking is reachable. Like previously, this command relies on an external tool to check if a marking is reachable in the reduced net. To this end, we use Sift, which is an explicit-state model checker for Petri nets from the Tina toolbox [BRV04; LAA23], that can check reachability properties on the fly.

The tool takes as input a Petri net—not necessarily safe, ordinary or bounded—described either in the PNML or the NET format. (NET is the specification format of the Tina toolbox). The target marking is defined using a simple textual format, as a space-separated list of place identifiers with their multiplicities, of the form `p*k`, where `p` is a place and `k` is a positive integer. By default, places that are not listed contain no tokens. The path to the file describing the target marking is given using option `--marking`. Finally, the option `--show-projected-marking` permits to output the projected marking on the reduced net, to be checked by Sift.

Architecture of Kong

Our tool is basically composed of three modules: `kong.py` the front-end program in charge of parsing command-line options; `pt.py` a Petri net parser; and `tfg.py` the data structure and computational module based on Token Flow Graphs. We illustrate the architecture of Kong in Fig. 8.1, where we describe the different steps involved during a typical computation.

The first step is to reduce the input Petri net, say (N, m), using the tool Reduce (or Shrink). Reduce outputs a reduced net (N', m') and a system of linear equations E. By construction, the result of this first stage is guaranteed to be a polyhedral equivalence. Kong provides an option, `--save-reduced-net`, to save the reduced net into a specific file. Additionally, we can print the reduction equations with the option `--show-equations`. Then we build a Token Flow Graph, $[\![E]\!]$ from the set of linear equations in E; a graphical version of the TFG can be displayed using option `--draw-graph`.

At this stage, we must distinguish two possible cases. First, the net could be fully reduced, meaning the resulting net is “empty”; it has no remaining places. In this case, the set of

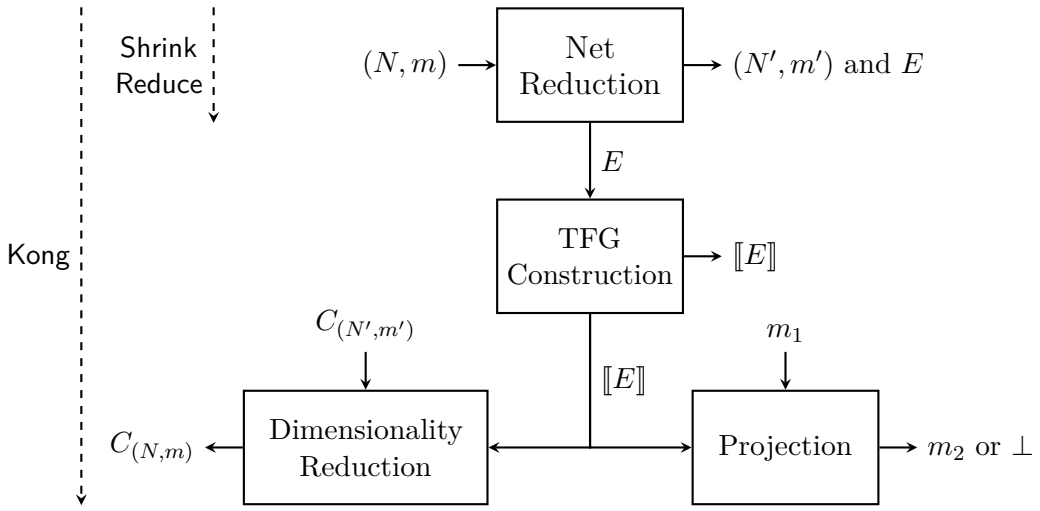


Fig. 8.1 Architecture of Kong.

markings of (N, m) is exactly the solution of the linear system E. Hence, the TFG is enough to compute the concurrency matrix using an algorithm that we call *dimensionality reduction* or to decide if a given marking is reachable. Otherwise, we have a non-trivial reduced net, in which case we need to obtain the concurrency matrix of (N', m') from an external tool or to check the reachability of the *projection* of our marking of interest.

8.5 Octant and Reductron: Two Hidden Tools

We now present two other tools, Octant [Ama23c] and Reductron [Ama23d], that are part of our toolchain but not dedicated to being used by end users (and so we do not detail their usage). Both tools are under GPLv3 license and are freely available on GitHub.

Octant is a preprocessor that, given a polyhedral equivalence $(N_1, m_1) \equiv_E (N_2, m_2)$, projects an initial reachability property into a simpler one, to be checked on the reduced net N_2. Octant implements the projection procedure described in Chapter 5. The tool is written in OCaml (about 1 500 lines of code) and is named after the Octant map projection[2] proposed by Leonardo da Vinci in 1508; the first known example of a polyhedral map projection.

Reductron is the tool that permits to prove that some parametric polyhedral equivalence, say $(N_1, C_1) \approxeq_E (N_2, C_2)$, is correct. The tool is written in Python (about 1 000 lines of code) and relies on the tool FAST [Bar+03; Bar+08] and the SMT solver z3 [MB08; Bjø].

[2] https://en.wikipedia.org/wiki/Octant_projection

8.6 Experimental Environment and Reproducibility

As mentioned at the beginning of this chapter, the experiments of this thesis were realized on the benchmark of the 2023 edition of the MCC (see Sect. 8.1). We used the latest versions of our tools, presented in the previous sections. The experimental results may differ slightly from those published, since we are using the latest versions of the binaries[3] and, more importantly, published papers use the formulas of previous MCC editions. Nonetheless, the results appear to be qualitatively the same between the different benchmarks, which is reassuring.

All experiments were run on the same node of our computing platform, with the following characteristics.

OS: Ubuntu 20.04.6 LTS (Focal Fossa)
CPU: 2× Intel Xeon-P 8352V (2×36 cores @3.50 GhZ)
Memory: 512 GB (@3200 MHz)

We provide an artifact to replicate the experiments that is made available on the Zenodo platform [Ama23b]. We have chosen to use a Docker image in order to be easily usable on both amd64 and Apple Silicon architectures. Please refer to the `README` provided with the artifact to build the `Dockerfile` file and replicate the experiments.

8.7 Three Years of Participation in the MCC

I conclude this chapter with a short experience report on our three participation, with SMPT, in the reachability category of the Model Checking Contest; see for instance the evolution of tool performances in Fig. 8.2. We also comment on the results obtained by SMPT, ITS-Tools [Thi15], and TAPAAL [Dav+12] at the 2023 edition of the MCC [Kor+23].

Year-by-year Review

Our first participation was in 2021. At this time, SMPT was only composed of two methods: Bounded Model Checking (BMC) and an initial implementation of PDR that used only the state-based generalization described in Sect. 2.3.2. As a result, it was not able to manage true invariants that are not coverability properties. Of course, SMPT was integrating polyhedral reduction, using the E-transform formula described in Chapter 3. This prototype version of SMPT was able to solve more than 50% of the queries, and already performed better than exhaustive approaches using decision diagrams, like the one in GreatSPN [Amp+16; ADG22].

The following year, in 2022, we added a competition mode to SMPT, which includes a basic strategy to orchestrate the different verification methods. Our simple, two step strategy was to start by trying: (1) random walk state space exploration (in order to eliminate queries

[3] Except for ITS-Tools and TAPAAL, for which we used the binaries submitted to the 2022 edition of the MCC [Kor+22] for compatibility with our `libc`.

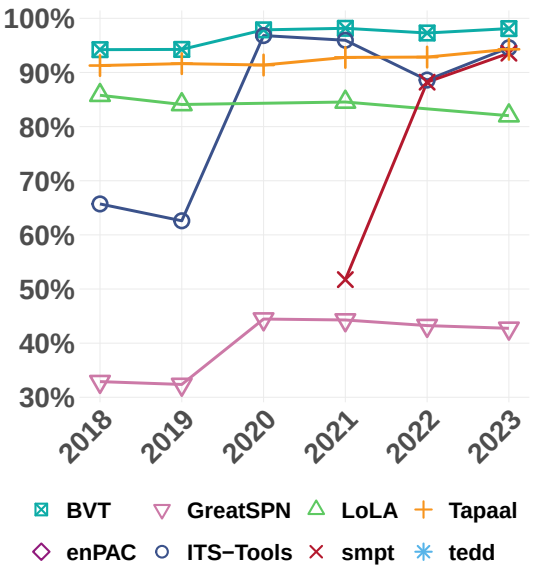


Fig. 8.2 Evolution of tool performance. (copied from [Ama+23])

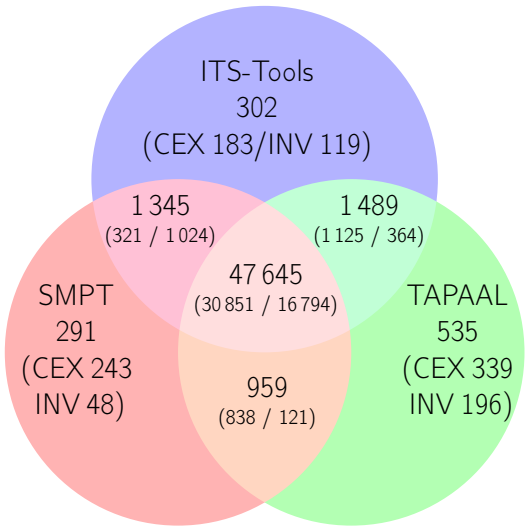


Fig. 8.3 Comparison of tools on all computed queries in 2023.

decided by an easily found counter-example), and (2) the state equation method (to catch invariants implied by the structural invariants of the net), in parallel, with a timeout of 120 s, on all formulas. The second step was to run more demanding methods as long as possible: BMC, k-induction, PDR, etc. These improvements made it possible to obtain the bronze medal (behind TAPAAL and ITS-Tools) by computing 88.6% of the queries. In comparison, TAPAAL computed 92.8% of the queries in the benchmark. SMPT obtained a 100% confidence score in 2022 (meaning the tool never returned an erroneous verdict), a first proof that our tool had become quite mature.

Finally, in 2023, we added support for our formula projection method, which is implemented in our tool Octant; see Chapter 5. With this latest version, we were able to compute 93.6% of all queries in the benchmark; a substantial increase of 5.5% when compared with our results from 2022. This is quite a good result, considering that the ratios of solved queries for ITS-Tools and TAPAAL in 2023 are respectively 94.6% and 94.3%. SMPT still obtained the bronze medal (ahead of LoLA [Wol18] and GreatSPN), but failed to reach the 100% confidence level due to an error with a single query, a mistake when parsing the output of a routine function used to simplify formulas.

Detailed Results

We display the results of SMPT, ITS-Tools and TAPAAL during the 2023 edition in a Venn diagram (Fig. 8.3) where we make a distinction between CEX (false AG properties or true EF

ones) and INV properties (true AG properties or false EF ones). There is a total of 52 566 answered queries (with almost 65% CEX). We observe that a vast majority of these queries (47 645) are computed by all tools, and can be considered "easy". Conversely, we have 4 921 difficult queries, solved by only one or two tools.

Overall, we observe that SMPT performs well compared to other state-of-the-art tools in the Model Checking Contest and that tools are quite complementary. In particular, we observe that SMPT is a sensible choice when checking invariants (INV queries), a result that we would like to credit to our enhanced model checking procedures.

- **Biology and chemistry**: Angiogenesis, CircadianClock, Diffusion2D, DNAWalker, EGFr, ERK, GPPP, MAPK, MAPKbis, PaceMaker, PhaseVariation, ViralEpidemic;
- **business process and automation**: BugTracking, BusinessProcesses, CryptoMiner, FamilyReunion, FMS, HealthRecord, HospitalTriage, HouseConstruction, IBM (4 models), Kanban, Medical, ProductionCell, ParamProductionCell, RobotManipulation, UtilityControlRoom;
- **distributed memory and related algorithms**: CANConstruction, CANInsertWithFailure, LeafsetExtension, MultiCrashLeafsetExtension, QuasiCertifProtocol, SatelliteMemory, SharedMemory, StigmergyCommit;
- **elections or consensus**: Election2020, HirschbergSinclair, NeoElection, Raft, StigmergyElection;
- **games**: DLCRound, DLCShifumi, NQueens, Solitaire, Sudoku:
- **hardware**: ARMCacheCoherence, ASLink, DiscoveryGPU, GPUForwardProgress, NoC3x3, Ring, SafeBus, TokenRing, UtahNoC, Vasy2003;
- **operating systems or middleware**: PolyORBLF, PolyORBNT, SimpleLoadBalancer, SmallOperatingSystem;
- **IoT, cloud, reconfiguration**: CloudDeployment, CloudOpsManagement, CloudReconfiguration, Planning, SmartHome;
- **mutual exclusion**: Anderson, DatabaseWithMutex, Dekker, DoubleLock, EisenbergMcGuire, FunctionPointer, GlobalResAllocation, LamportFastMutEx, Peterson, Philosophers, PhilosophersDyn, ResAllocation, RwMutex, SwimmingPool, Szymanski, TwoPhaseLocking;
- **network protocols**: CSRepetitions, Echo, HexagonalGrid, HypercubeGrid, HypertorusGrid, IOTPpurchase, NeighborGrid, PermAdmissibility, SquareGrid, TCPcondis, TriangularGrid, VehicularWifi;
- **security**: DES, ShieldIIPs, ShieldIIPt, ShieldPPPs, ShieldPPPt, ShieldRVs, ShieldRVt;
- **synchronisations and message passing**: ClientsAndServers, DBSingleClientW, DLCflexbar, FlexibleBarrier, MultiwaySync, RingSingleMessageInMbox, SemanticWebServices, ServersAndClients, SieveSingleMsgMbox;
- **academic and synthetic models**: DoubleExponent, Eratosthenes, DrinkVendingMachine, JoinFreeModules, Murphy, PGCD, Referendum, RefineWMG, RERS (4 models);
- **transportation systems**: AirplaneLD, AutoFlight, AutonomousCar, BART, BridgeAndVehicles, CircularTrains, EnergyBus, Parking, Railroad.

Table 8.1 List of models in the MCC benchmark (copied from [Ama+23]) divided according to their application domain.

This work has been published in:

- N. Amat and L. Chauvet. "Kong: a Tool to Squash Concurrent Places". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. vol. 13288. Springer, 2022. DOI: 10.1007/978-3-031-06653-5_6
- N. Amat and S. Dal Zilio. "SMPT: A Testbed for Reachabilty Methods in Generalized Petri Nets". In: *Formal Methods (FM)*. vol. 14000. Lecture Notes in Computer Science. Springer, 2023. DOI: 10.1007/978-3-031-27481-7_25
- N. Amat, P. Bouvier, and H. Garavel. "A Toolchain to Compute Concurrent Places of Petri Nets". In: *Transactions on Petri Nets and Other Models of Concurrency XVII*. Lecture Notes in Computer Science 14150 (2024), pp. 1–26. DOI: 10.1007/978-3-662-68191-6_1

A conference artifact is available on Zenodo:

- N. Amat and S. Dal Zilio. *Artifact for FM 2023 Paper: SMPT: A Testbed for Reachability Methods in Generalized Petri Nets*. Zenodo, 2022. DOI: 10.5281/zenodo.7341426

The tools related to this chapter are:

- SMPT https://github.com/nicolasAmat/SMPT
- Kong https://github.com/nicolasAmat/Kong
- Octant https://github.com/nicolasAmat/Octant
- Reductron https://github.com/nicolasAmat/Reductron

Epilogue

Contributions and Perspectives

> We can only see a short distance ahead, but we can see plenty there that needs to be done.
>
> Alan Turing

My initial goal, that was set to me at the beginning of my PhD thesis, was to study how to combine structural reductions with the model checking of reachability properties, if possible with the use of symbolic methods. As we reach the conclusion of this manuscript, it is time to take stock and look at the different results that I have obtained trying to achieve this goal.

My work led me to address different research problems. A cornerstone of my results is the definition of a new notion of *polyhedral equivalence* between nets, see Chapter 3, whereas the original approach used in [BLD18] relied on a single, monolithic reduction system. This provides an elegant way to separate two different aspects of our approach: (1) a theoretical one—what are the properties, between the state space of two nets, needed to apply our methods—; and (2) a practical one—how can we automatically compute a useful equivalence in practice, and how do we exploit it for model checking. The concurrent study, and the interactions between these two aspects produced interesting results. This dual concern also explains our choice of the term "framework" in the title of this thesis—instead of a more mundane expression, like "polyhedral method" for example—as a way to stress the fact that we provide both theoretical tools, to understand our approach, and a structured set of software tools to apply it on real examples.

On the theoretical side, it led us to the question of extending our approach with more general linear constraints. In this respect, the use of the QF-LIA theory, then the move to Presburger arithmetic, appears as a natural choice. One of the unexpected results of this endeavor was to unearth a deep connection between polyhedral equivalence and the notion of flat nets [Bar+05]. This gives a better understanding of what we can achieve with our polyhedral method, which can be roughly described as: simplifying a net by trying to find, and then abstract away, its "flat" sub-parts. This connection plays a key role in our method for automatically checking polyhedral equivalence (Chapter 7), which is one of our main contributions.

We also obtained results concerning the practical aspects of our approach. We proposed a new data stucture (the TFG of Chapter 4) that captures quite precisely the "shape" of the equation system, E, obtained using structural reductions. TFGs have been used for two of our main contributions: to optimize the computation of the concurrency relation (see Chapter 6), and when defining our formula projection method (see Chapter 5).

But there is also a third protagonist in this story, in the form of the Model Checking Contest. Participating in the MCC had a significant impact on my work. It obviously affected the experimental evaluation of my methods, via the use of its benchmark, but it has also been an ideal playground for performing open and reproducible science.

Before describing possible paths for future works, we describe some of our contributions in more details. To get a clearer picture, we describe in Fig. 8.4 the dependencies between chapters and their relations to our contributions.

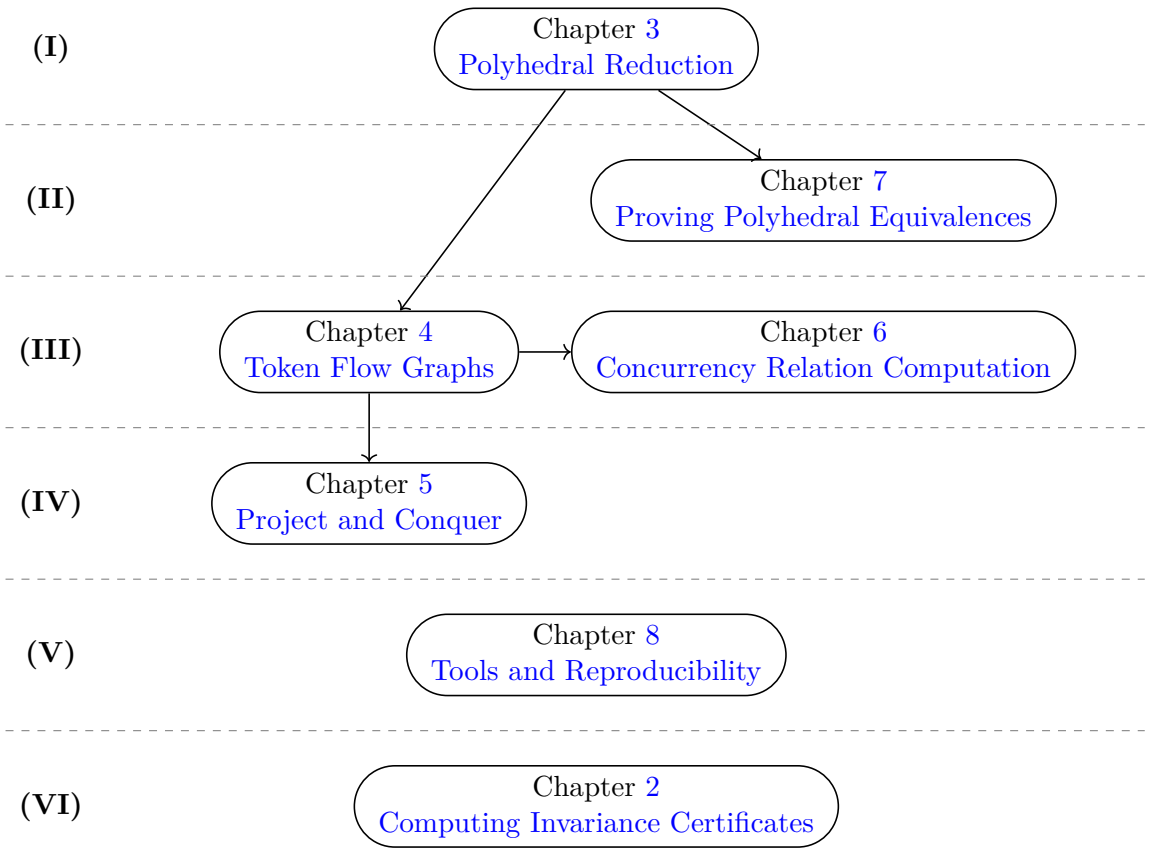


Fig. 8.4 Chapter and contribution dependency graph.

(I) Polyhedral Abstraction Equivalence. To adapt our approach with the theory of SMT solving, we have defined an abstraction based on Boolean combinations of linear constraints between integer variables (representing the marking of places). This resulted in a new relation,

denoted $N \equiv_E N'$. We called this equivalence *polyhedral* in reference to "polyhedral models" used in program optimization and static analysis [Fea96; BJT99]. Indeed, like in these works, we proposed an algebraic representation of the relation between a model and its state space which relies on the sets of solutions to predicates of linear constraints.

One of our main results is that, given a relation $N \equiv_E N'$, we can derive a formula $\tilde{E}$ such that F is reachable in N if and only if $\tilde{E} \wedge F$ is reachable in N'. This is interesting if the net N' has fewer places than N, in which case we may expect that checking the reachability of $\tilde{E} \wedge F$ on N' is more efficient than checking F in N.

A difference with previous works on structural reductions, e.g. [Ber87], is that our approach is not tailored to a particular class of properties—such as the absence of deadlocks—but can be applied to more general problems, expressed as linear formulas over the marking of places. In particular, we can apply more aggressive reduction techniques than, say, with *slicing* [Rak12; Llo+17; KKG18], *cone of influence* [CGP99], or other methods [GRV08; KBJ21] that seek to remove or gather together places that are not relevant to the property we want to check (and so cannot contribute to its truth value). We do not share this restriction in our approach, since we reduce nets beforehand and can therefore reduce places that occur in the initial property. We could argue that approaches similar to slicing only simplify a model with respect to a formula, whereas we simplify both the model and the formula using our new method. This is more efficient when we need to check several properties on the same model and, in any case, nothing prevents us from applying slicing techniques on the result of our reduction.

(II) Automated Proving. Polyhedral equivalence may be subtle, and there is no proof method associated with it. To solve this problem, we propose a procedure to automatically prove *polyhedral equivalence* between pairs of parametric Petri nets. This contribution is motivated by our goal to increase our confidence on the tools that we implement. In this context, we use the term *parametric* to stress the fact that we manipulate semilinear sets of markings, meaning sets that can be defined using a Presburger arithmetic formula C. In particular, we reason about parametric nets (N, C), instead of marked nets (N, m_0), with the intended meaning that all markings satisfying C are potential initial markings of N. We define an extended notion of polyhedral equivalence between parametric nets, denoted $(N, C) \approxeq_E (N', C')$, whereas our standard definition is between marked nets only. We show that, given a valid equivalence statement $(N, C) \approxeq_E (N', C')$, it is possible to derive a Presburger formula, in a constructive way, whose satisfaction implies that the equivalence holds. Our approach relies on an encoding into a set of SMT formulas using the LIA theory—which means that we may need quantifiers alternation in this case. The difficulty, in this context, arises from the fact that we need to handle infinite-state systems. For completeness, we exploit a connection with the class of Petri nets with Presburger-definable reachability sets.

(III) Token Flow Graphs and the Concurrency Relation. Token Flow Graphs (TFG) capture the particular structure of constraints occurring in the linear system, E, generated using structural reductions. To demonstrate the versatility of this structure, we have applied it in Chapter 6 to a specific problem, that is to compute the *concurrency relation* of a net, i.e., enumerating all pairs of places that can be marked together in some reachable marking. This problem turns out to be useful for the decomposition into NUPNs. We described TFGs and showed how to leverage this data structure in order to accelerate the computation of the concurrency relation of a net. We use the term acceleration to stress the "multiplicative effect" of TFGs. Indeed, we propose a framework that, starting from a tool for solving the concurrent places problem, provide an augmented version of this tool that takes advantage of reductions. The augmented tool can compute the concurrency relation for the initial instance N, by computing it on a reduced version N', and then reconstructing a correct solution for the initial instance. In each case, our approach takes the form of an "inverse transform" that relies only on E and that does not involve expensive preprocessing on the reduced net. This foreshadows our next contribution, where we tried to come up with a similar architecture when checking reachability formulas.

(IV) Formula Projection. Model checking with our polyhedral approach entails checking formulas of the form $E \wedge F$. A complication arises from the fact that formula $E \wedge F$ may include variables (places) that no longer occur in the reduced net N', and therefore act as existentially quantified variables. This can complicate some symbolic verification techniques, such as k-induction, and impede the use of explicit, enumerative approaches. Indeed, in the later case, it means that we need to solve an integer linear problem for each new state, instead of just evaluating a closed formula. To overcome this problem, we proposed a new method, Chapter 5, for projecting the formula $E \wedge F$ into an equivalent one, F', that only refers to the places of N'. Then, F' can be checked on N' using any off-the-shelf verification tool.

Our projection can be defined as a semi-procedure for quantifier elimination in Presburger arithmetic, tailored for the specific kind of constraints we handle in E. Whereas quantifier elimination has an exponential complexity in general for existential formulas, our construction has linear complexity and can only decrease the size of a formula. It also always terminates and returns a result that is guaranteed to be sound. Which means that it under-approximates the set of reachable models and, therefore, a witness of F' in N' necessarily corresponds to a witness of F in N. Additionally, our approach includes a simple condition on F that is enough to detect when our result is exact, meaning that if F' is unreachable in N', then F is unreachable in N.

(V) Software and Competition. Our approach and algorithms have been implemented in four open-source (under GPLv3 license) tools that we described in Chapter 8: SMPT for checking reachability properties; Kong for accelerating the computation of concurrent places;

Octant for eliminating quantifiers; and Reductron for automatically proving the correctness of polyhedral equivalences. We gave experimental results about their effectiveness, both for bounded and unbounded nets, using a large benchmark provided by the Model Checking Contest. We paid attention to the reproducibility of our results, and provide an accompanying artifact that covers all our experiments.

(VI) PDR and Certificates of Invariance. The last contribution in our list, and the subject of Chapter 2, is not directly related to our polyhedral framework. Our work led us to study the adaptation of several symbolic model checking algorithm for Petri nets, such as Bounded Model Checking (BMC) and k-induction. We propose a new algorithm as well, which is a semi-decision procedure based on the Property Directed Reachability (PDR) method. A distinctive feature of our extension of PDR to Petri nets is the ability to generate "certificate of invariance", in the form of an inductive Presbuger invariant (see Theorem 1.2), when we find that a property holds on all the reachable markings. We actually defined three different versions, that vary depending on the method used for abstracting possible witnesses, and that are able to handle problems of increasing difficulty. In this work, we have sought improvements in terms of both "performance" and "expressiveness". We also targeted what we consider to be a difficult, and less studied area of research: procedures that can be applied when a property is an invariant and when the net is unbounded, or its state space cannot be fully explored.

Perspectives

There is much to be humble about the reachability problem, which seems quite simple at first glance, but which has and will continue to pique the interest of researchers from the theoretical computer science community.

I am convinced that further research is needed in order to develop new algorithms to address reachability queries that are out of reach with current methods. In this context, my adaptation of the PDR method could be improved; for instance by adding new heuristics or new "acceleration methods" to better block groups of related witnesses.

I also believe that portfolio approaches will continue to be the most effective strategy in practice, considering the theoretical difficulty of the problem. But new research is also necessary in this context, to improve the strategies used to select, tune, and combine all the available methods in a portfolio.

Another direction for future works would be to investigate the use of TFG with other verification problems. For instance computing the *max-marking* of a net, which is the maximal number of tokens in a reachable state. This is a first step towards tackling "optimization" problems over the reachable states of a net. Another related problem is the *generalized mutual exclusion constraints* [GDS92], which extends reachability queries by allowing the use of non-trivial coefficients (other than 1 or -1) in atomic formulas. In the same line of research, it

would be interesting to improve the precision of our formula projection procedure when the result is not complete. Even though we have shown that such cases rarely occur in practice.

I am also interested in pursuing problems that are close, but not directly related to polyhedral reductions. For instance, finding a "workable" algorithm to decide if the reachability set of a net is Presbuger-definable. This will would be helpful in our automated proving procedure.

My theoretical works build on the close relation between Presburger arithmetic and Petri nets theory, such as the development of an optimized quantifier elimination procedure. This explains my motivation to explore current topics in Presburger arithmetic. Such subjects fit logically within a broader research project on which I want to embark. The advances made in SMT solvers in recent years have led to significant advances in verification tools. Nevertheless, sometimes solvers remain too general and are not adapted to specific problems. In this context, I plan to develop more specific solvers and decision procedures that take better into account the underlying models. These results could have applications in several domains: for model checking obviously; but also for solving problems occurring in planification and schedulability analysis; or for checking the correctness of autonomous critical systems.

References

[Aal15] W. M. P. van der Aalst. "Business process management as the "Killer App" for Petri nets". In: *Software & Systems Modeling* 14.2 (2015), pp. 685–691. DOI: 10.1007/s10270-014-0424-2.

[Age74] T. Agerwala. *Complete model for representing the coordination of asynchronous processes*. Research report. Johns Hopkins University, 1974.

[Ama23a] N. Amat. *A QF-LIA Benchmark Suite from Polyhedral Reductions of Petri Nets*. Research report. LAAS-CNRS, 2023.

[Ama23b] N. Amat. *Artifact for PhD thesis: "A polyhedral framework for reachability problems in Petri nets"*. Zenodo, 2023. DOI: 10.5281/zenodo.8349546.

[Ama20a] N. Amat. *Kong: The Koncurrent Places Grinder. A tool to accelerate the computation of the concurrency relation of a Petri net using polyhedral reduction.* 2020. URL: https://github.com/nicolasAmat/Kong (visited on 10/10/2023).

[Ama23c] N. Amat. *Octant: The Reachability Formula Projector. A tool to project Petri net reachability properties on reduced nets using polyhedral reduction.* 2023. URL: https://github.com/nicolasAmat/Octant (visited on 10/10/2023).

[Ama23d] N. Amat. *Reductron: The Polyhedral Abstraction Prover. A tool to automatically prove the correctness of polyhedral equivalences for Petri nets.* 2023. URL: https://github.com/nicolasAmat/Reductron (visited on 10/10/2023).

[Ama20b] N. Amat. *SMPT: The Satisfiability Modulo Petri Nets Model Checker. An SMT-based model checker for Petri nets focused on reachability problems that takes advantage of polyhedral reduction.* 2020. URL: https://github.com/nicolasAmat/SMPT (visited on 10/10/2023).

[Ama23e] N. Amat. *uSMPT: an educational project, targeting Master and PhD students to showcase the application of SMT methods in system verification, by developing a Petri net model checker for the reachability problem.* 2023. URL: https://github.com/nicolasAmat/uSMPT (visited on 10/10/2023).

[Ama+23] N. Amat, E. Amparore, B. Berthomieu, P. Bouvier, S. Dal Zilio, P. G. Jensen, L. Jezequel, F. Kordon, S. Li, E. Paviot-Adet, J. Srba, Y. Thierry-Mieg, and K. Wolf. "Behind the Scene of the Model Checking Contest, Analysis of Results from 2018 to 2023". Submitted. 2023.

[ABD22] N. Amat, B. Berthomieu, and S. Dal Zilio. "A Polyhedral Abstraction for Petri Nets and its Application to SMT-Based Model Checking". In: *Fundamenta Informaticae* 187.2-4 (2022), pp. 103–138. DOI: 10.3233/FI-222134.

[ABD21] N. Amat, B. Berthomieu, and S. Dal Zilio. "On the Combination of Polyhedral Abstraction and SMT-Based Model Checking for Petri Nets". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 12734. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-76983-3_9.

[ABG24] N. Amat, P. Bouvier, and H. Garavel. "A Toolchain to Compute Concurrent Places of Petri Nets". In: *Transactions on Petri Nets and Other Models of Concurrency XVII*. Lecture Notes in Computer Science 14150 (2024), pp. 1–26. DOI: 10.1007/978-3-662-68191-6_1.

[AC22] N. Amat and L. Chauvet. "Kong: a Tool to Squash Concurrent Places". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 13288. Springer, 2022. DOI: 10.1007/978-3-031-06653-5_6.

[AD22] N. Amat and S. Dal Zilio. *Artifact for FM 2023 Paper: SMPT: A Testbed for Reachability Methods in Generalized Petri Nets*. Zenodo, 2022. DOI: 10.5281/zenodo.7341426.

[AD23] N. Amat and S. Dal Zilio. "SMPT: A Testbed for Reachabilty Methods in Generalized Petri Nets". In: *Formal Methods (FM)*. Vol. 14000. Lecture Notes in Computer Science. Springer, 2023. DOI: 10.1007/978-3-031-27481-7_25.

[ADH22a] N. Amat, S. Dal Zilio, and T. Hujsa. *Artifact for TACAS 2022 Paper: Property Directed Reachability for Generalized Petri Nets*. Zenodo, 2022. DOI: 10.5281/zenodo.5863379.

[ADH23a] N. Amat, S. Dal Zilio, and T. Hujsa. *Model entitled "CryptoMiner" proposed for the Model Checking Contest*. 2023. URL: https://mcc.lip6.fr/2023/pdf/CryptoMiner-form.pdf (visited on 10/10/2023).

[ADH23b] N. Amat, S. Dal Zilio, and T. Hujsa. *Model entitled "Murphy" proposed for the Model Checking Contest*. 2023. URL: https://mcc.lip6.fr/2023/pdf/Murphy-form.pdf (visited on 10/10/2023).

[ADH23c] N. Amat, S. Dal Zilio, and T. Hujsa. *Model entitled "PGCD" proposed for the Model Checking Contest*. 2023. URL: https://mcc.lip6.fr/2023/pdf/PGCD-form.pdf (visited on 10/10/2023).

[ADH22b] N. Amat, S. Dal Zilio, and T. Hujsa. "Property Directed Reachability for Generalized Petri Nets". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 13243. Lecture Notes in Computer Science. Springer, 2022. DOI: 10.1007/978-3-030-99524-9_28.

[ADL21] N. Amat, S. Dal Zilio, and D. Le Botlan. "Accelerating the Computation of Dead and Concurrent Places Using Reductions". In: *Model Checking Software (SPIN)*. Vol. 12864. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-84629-9_3.

[ADL23a] N. Amat, S. Dal Zilio, and D. Le Botlan. *Artifact for VMCAI 2024 Paper "Project and Conquer: Fast Quantifier Elimination for Checking Petri Net Reachability"*. Zenodo, 2023. DOI: 10.5281/zenodo.10061156.

[ADL23b] N. Amat, S. Dal Zilio, and D. Le Botlan. "Automated Polyhedral Abstraction Proving". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 13929. Lecture Notes in Computer Science. Springer, 2023. DOI: 10.1007/978-3-031-33620-1_18.

[ADL23c] N. Amat, S. Dal Zilio, and D. Le Botlan. "Leveraging polyhedral reductions for solving Petri net reachability problems". In: *International Journal on Software Tools for Technology Transfer* 25.1 (2023), pp. 95–114. DOI: 10.1007/s10009-022-00694-8.

[ADL24] N. Amat, S. Dal Zilio, and D. Le Botlan. "Project and Conquer: Fast Quantifier Elimination for Checking Petri Nets Reachability". In: *Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Lecture Notes in Computer Science. Springer, 2024. DOI: 10.1007/978-3-031-50524-9_5.

[Amp+19] E. Amparore, B. Berthomieu, G. Ciardo, S. Dal Zilio, F. Gallà, L. M. Hillah, F. Hulin-Hubard, P. G. Jensen, L. Jezequel, F. Kordon, D. Le Botlan, T. Liebke, J. Meijer, A. Miner, E. Paviot-Adet, J. Srba, Y. Thierry-Mieg, T. van Dijk, and K. Wolf. "Presentation of the 9th Edition of the Model Checking Contest". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 11429. Lecture Notes in Computer Science. Springer, 2019. DOI: 10.1007/978-3-662-58381-4_9.

[Amp+16] E. G. Amparore, G. Balbo, M. Beccuti, S. Donatelli, and G. Franceschinis. "30 Years of GreatSPN". In: *Principles of Performance and Reliability Modeling and Evaluation*. Springer, 2016, pp. 227–254. DOI: 10.1007/978-3-319-30599-8_9.

[ADC20] E. G. Amparore, S. Donatelli, and G. Ciardo. "Variable order metrics for decision diagrams in system verification". In: *International Journal on Software Tools for Technology Transfer* 22.5 (2020), pp. 541–562. DOI: 10.1007/s10009-019-00522-6.

[ADG22] E. G. Amparore, S. Donatelli, and F. Gallà. "starMC: an automata based CTL* model checker". In: *PeerJ Computer Science* 8.e823 (2022). DOI: 10.7717/peerj-cs.823.

[ADS11] D. Angeli, P. De Leenheer, and E. D. Sontag. "Persistence Results for Chemical Reaction Networks with Time-Dependent Kinetics and No Global Conservation Laws". In: *SIAM Journal on Applied Mathematics* 71.1 (2011), pp. 128–146. DOI: 10.1137/090779401.

[AK76] T. Araki and T. Kasami. "Some decision problems related to the reachability problem for Petri nets". In: *Theoretical Computer Science* 3.1 (1976), pp. 85–104. DOI: 10.1016/0304-3975(76)90067-0.

[AMP06] A. Armando, J. Mantovani, and L. Platania. "Bounded Model Checking of Software Using SMT Solvers Instead of SAT Solvers". In: *Model Checking Software (SPIN)*. Vol. 3925. Lecture Notes in Computer Science. Springer, 2006. DOI: 10.1007/11691617_9.

[Arn02] A. Arnold. "Nivat's processes and their synchronization". In: *Theoretical Computer Science* 281.1 (2002), pp. 31–36. DOI: 10.1016/S0304-3975(02)00006-3.

[AS80] B. Aspvall and Y. Shiloach. "A Polynomial Time Algorithm for Solving Systems of Linear Inequalities with Two Variables Per Inequality". In: *SIAM Journal on Computing* 9.4 (1980), pp. 827–845. DOI: 10.1137/0209063.

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[Bal+10] P. Baldan, N. Cocco, A. Marin, and M. Simeoni. "Petri nets for modelling metabolic pathways: a survey". In: *Natural Computing* 9.4 (2010), pp. 955–989. DOI: 10.1007/s11047-010-9180-6.

[Bar+08] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. "FAST: acceleration from theory to practice". In: *International Journal on Software Tools for Technology Transfer* 10.5 (2008), pp. 401–424. DOI: 10.1007/s10009-008-0064-3.

[Bar+03] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. "FAST: Fast Acceleration of Symbolic Transition Systems". In: *Computer Aided Verification (CAV)*. Vol. 2725. Lecture Notes in Computer Science. Springer, 2003. DOI: 10.1007/978-3-540-45069-6_12.

[Bar+05] S. Bardin, A. Finkel, J. Leroux, and P. Schnoebelen. "Flat Acceleration in Symbolic Model Checking". In: *Automated Technology for Verification and Analysis (ATVA)*. Vol. 3707. Lecture Notes in Computer Science. Springer, 2005. DOI: 10.1007/11562948_35.

[BFT17] C. Barrett, P. Fontaine, and C. Tinelli. *The SMT-LIB Standard: Version 2.6*. Standard. University of Iowa, 2017.

[Bar+22] M. Barth, D. Dietsch, M. Heizmann, and A. Podelski. *Ultimate Eliminator at SMT-COMP 2022*. Research report. University of Freiburg, 2022.

[Bas98] T. Basten. "In terms of nets : system design with Petri nets and process algebra". PhD thesis. Technische Universiteit Eindhoven, 1998. DOI: 10.6100/IR516117.

[BF99] B. Bérard and L. Fribourg. "Reachability Analysis of (Timed) Petri Nets Using Real Arithmetic". In: *Concurrency Theory (CONCUR)*. Vol. 1664. Lecture Notes in Computer Science. Springer, 1999. DOI: 10.1007/3-540-48320-9_14.

[BPS01] J. A. Bergstra, A. Ponse, and S. A. Smolka. *Handbook of process algebra*. Elsevier, 2001. DOI: 10.1016/B978-0-444-82830-9.X5017-6.

[Ber87] G. Berthelot. "Transformations and Decompositions of Nets". In: *Petri Nets: Central Models and Their Properties (ACPN)*. Vol. 254. Lecture Notes in Computer Science. Springer, 1987. DOI: 10.1007/978-3-540-47919-2_13.

[BL85] G. Berthelot and Lri-Iie. "Checking properties of nets using transformations". In: *Advances in Petri Nets (APN)*. Vol. 222. Lecture Notes in Computer Science. Springer, 1985. DOI: 10.1007/BFb0016204.

[BRV04] B. Berthomieu, P.-O. Ribet, and F. Vernadat. "The tool TINA – Construction of abstract state spaces for Petri nets and time Petri nets". In: *International Journal of Production Research* 42.14 (2004), pp. 2741–2756. DOI: 10.1080/00207540412331312688.

[BLD19] B. Berthomieu, D. Le Botlan, and S. Dal Zilio. "Counting Petri net markings from reduction equations". In: *International Journal on Software Tools for Technology Transfer* 22.2 (2019), pp. 163–181. DOI: 10.1007/s10009-019-00519-1.

[BLD18] B. Berthomieu, D. Le Botlan, and S. Dal Zilio. "Petri net Reductions for Counting Markings". In: *Model Checking Software (SPIN)*. Vol. 10869. Lecture Notes in Computer Science. Springer, 2018. DOI: 10.1007/978-3-319-94111-0_4.

[BJT99] F. Besson, T. Jensen, and J.-P. Talpin. "Polyhedral Analysis for Synchronous Languages". In: *Static Analysis (SAS)*. Vol. 1694. Lecture Notes in Computer Science. Springer, 1999. DOI: 10.1007/3-540-48294-6_4.

[BDK96] E. Best, R. Devillers, and M. Koutny. "Petri nets, process algebras and concurrent programming languages". In: *Lectures on Petri Nets II: Applications (ACPN)*. Vol. 1492. Lecture Notes in Computer Science. Springer, 1996. DOI: 10.1007/3-540-65307-4_46.

[Bie+99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. "Symbolic Model Checking without BDDs". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 1579. Lecture Notes in Computer Science. Springer, 1999. DOI: 10.1007/3-540-49059-0_14.

[Bil+21] A. Bilgram, P. G. Jensen, T. Pedersen, J. Srba, and P. H. Taankvist. "Improvements in Unfolding of Colored Petri Nets". In: *Reachability Problems (RP)*. Vol. 13035. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-89716-1_5.

[Bjø] N. Bjørner. *The Z3 Theorem Prover*. URL: https://github.com/Z3Prover/z3/ (visited on 10/10/2023).

[BHO21] M. Blondin, C. Haase, and P. Offtermatt. "Directed Reachability for Infinite-State Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 12652. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-72013-1_1.

[Boi98] B. Boigelot. "Symbolic methods for exploring infinite state spaces". PhD thesis. ULiège-Université de Liège, 1998.

[BW94] B. Boigelot and P. Wolper. "Symbolic verification with periodic sets". In: *Computer Aided Verification (CAV)*. Vol. 818. Lecture Notes in Computer Science. Springer, 1994. DOI: 10.1007/3-540-58179-0_43.

[Boj+11] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. "Two-variable logic on data words". In: *ACM Transactions on Computational Logic* 12.4 (2011), pp. 1–26. DOI: 10.1145/1970398.1970403.

[BB87] T. Bolognesi and E. Brinksma. "Introduction to the ISO specification language LOTOS". In: *Computer Networks and ISDN Systems* 14.1 (1987), pp. 25–59. DOI: 10.1016/0169-7552(87)90085-7.

[Bøn+19] F. M. Bønneland, J. Dyhr, P. G. Jensen, M. Johannsen, and J. Srba. "Stubborn versus structural reductions for Petri nets". In: *Journal of Logical and Algebraic Methods in Programming* 102 (2019), pp. 46–63. DOI: 10.1016/j.jlamp.2018.09.002.

[BT76] I. Borosh and L. B. Treybig. "Bounds on Positive Integral Solutions of Linear Diophantine Equations". In: *Proceedings of the American Mathematical Society* 55.2 (1976), pp. 299–304. DOI: 10.2307/2041711.

[BE12] A. Bouajjani and M. Emmi. "Analysis of recursively parallel programs". In: *ACM SIGPLAN Notices* 47.1 (2012), pp. 203–214. DOI: 10.1145/2103621.2103681.

[BG21] P. Bouvier and H. Garavel. "Efficient Algorithms for Three Reachability Problems in Safe Petri Nets". In: *Application and Theory of Petri Nets and Concurrency (Petri Nets)*. Vol. 12734. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-76983-3_17.

[BGP20] P. Bouvier, H. Garavel, and H. Ponce-de-León. "Automatic Decomposition of Petri Nets into Automata Networks – A Synthetic Account". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 12152. Lecture Notes in Computer Science. Springer, 2020. DOI: 10.1007/978-3-030-51831-8_1.

[Bra11] A. R. Bradley. "SAT-Based Model Checking without Unrolling". In: *Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Vol. 6538. Lecture Notes in Computer Science. Springer, 2011. DOI: 10.1007/978-3-642-18275-4_7.

[Bra12] A. R. Bradley. "Understanding IC3". In: *Theory and Applications of Satisfiability Testing (SAT)*. Vol. 7317. Lecture Notes in Computer Science. Springer, 2012. DOI: 10.1007/978-3-642-31612-8_1.

[Bur+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.-J. Hwang. "Symbolic model checking: 1020 states and beyond". In: *Information and Computation* 98.2 (1992), pp. 142–170. DOI: 10.1016/0890-5401(92)90017-A.

[BKY00] F. Burns, A. Koelmans, and A. Yakovlev. "WCET Analysis of Superscalar Processors Using Simulation With Coloured Petri Nets". In: *Real-Time Systems* 18.2 (2000), pp. 275–288. DOI: 10.1023/A:1008101416758.

[CJL17] F. Cassez, P. G. Jensen, and K. G. Larsen. "Refinement of Trace Abstraction for Real-Time Programs". In: *Reachability Problems (RP)*. Vol. 10506. Lecture Notes in Computer Science. Springer, 2017. DOI: 10.1007/978-3-319-67089-8_4.

[Cha22] L. Chauvet. *PNets: a Rust library for manipulating and reducing Petri nets*. 2022. URL: https://github.com/Fomys/pnets (visited on 10/10/2023).

[CEP95] A. Cheng, J. Esparza, and J. Palsberg. "Complexity results for 1-safe nets". In: *Theoretical Computer Science* 147.1 (1995), pp. 117–136. DOI: 10.1016/0304-3975(94)00231-7.

[Chi+93] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. "Stochastic well-formed colored nets and symmetric modeling applications". In: *Transactions on Computers* 42.11 (1993), pp. 1343–1360. DOI: 10.1109/12.247838.

[Chi+91] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. "On Well-Formed Coloured Nets and Their Symbolic Reachability Graph". In: *High-level Petri Nets*. Springer, 1991. DOI: 10.1007/978-3-642-84524-6_13.

[Cim+16] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. "Infinite-state invariant checking with IC3 and predicate abstraction". In: *Formal Methods in System Design* 49.3 (2016), pp. 190–218. DOI: 10.1007/s10703-016-0257-4.

[Cim+14] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. "IC3 Modulo Theories via Implicit Predicate Abstraction". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 8413. Lecture Notes in Computer Science. Springer, 2014. DOI: 10.1007/978-3-642-54862-8_4.

[CGP99] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[Cla+01] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. "Bounded Model Checking Using Satisfiability Solving". In: *Formal Methods in System Design* 19.1 (2001), pp. 7–34. DOI: 10.1023/A:1011276507260.

[Cla+00] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. "Counterexample-Guided Abstraction Refinement". In: *Computer Aided Verification (CAV)*. Vol. 1855. Lecture Notes in Computer Science. Springer, 2000. DOI: 10.1007/10722167_15.

[CE81] E. M. Clarke and E. A. Emerson. "Design and synthesis of synchronization skeletons using branching time temporal logic". In: *Logic of Program*. Vol. 131. Lecture Notes in Computer Science. Springer. 1981. DOI: 10.1007/BFb0025774.

[CL98] E. Cohen and L. Lamport. "Reduction in TLA". In: *Concurrency Theory (CONCUR)*. Vol. 1466. Lecture Notes in Computer Science. Springer, 1998. DOI: 10.1007/BFb0055631.

[Com+71] F. Commoner, A. W. Holt, S. Even, and A. Pnueli. "Marked directed graphs". In: *Journal of Computer and System Sciences* 5.5 (1971), pp. 511–523. DOI: 10.1016/S0022-0000(71)80013-2.

[Coo72] D. C. Cooper. "Theorem Proving in Arithmetic without Multiplication". In: *Machine Intelligence* 7 (1972), pp. 91–100.

[Cos+10] N. Coste, H. Garavel, H. Hermanns, F. Lang, R. Mateescu, and W. Serwe. "Ten Years of Performance Evaluation for Concurrent Systems Using CADP". In: *Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA)*. Vol. 6416. Lecture Notes in Computer Science. Springer. 2010. DOI: 10.1007/978-3-642-16561-0_18.

[CL07] H. Costelha and P. Lima. "Modelling, analysis and execution of robotic tasks using petri nets". In: *Intelligent Robots and Systems (IROS)*. IEEE, 2007. DOI: 10.1109/IROS.2007.4399365.

[CH78] P. Cousot and N. Halbwachs. "Automatic discovery of linear restraints among variables of a program". In: *Principles of Programming Languages (POPL)*. ACM, 1978. DOI: 10.1145/512760.512770.

[Cze+20] W. Czerwiński, S. Lasota, R. Lazić, J. Leroux, and F. Mazowiecki. "The Reachability Problem for Petri Nets is Not Elementary". In: *Journal of the ACM* 68.1 (2020), pp. 1–28. DOI: 10.1145/3422822.

[CO22] W. Czerwiński and Ł. Orlikowski. "Reachability in Vector Addition Systems is Ackermann-complete". In: *Foundations of Computer Science (FOCS)*. IEEE, 2022. DOI: 10.1109/FOCS52979.2021.00120.

[Dal20] S. Dal Zilio. "MCC: A Tool for Unfolding Colored Petri Nets in PNML Format". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 12152. Lecture Notes in Computer Science. Springer, 2020. DOI: 10.1007/978-3-030-51831-8_23.

[Dav+12] A. David, L. Jacobsen, M. Jacobsen, K. Y. Jørgensen, M. H. Møller, and J. Srba. "TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 7214. Lecture Notes in Computer Science. Springer, 2012. DOI: 10.1007/978-3-642-28756-5_36.

[Dia09] M. Diaz. *Petri Nets: Fundamental Models, Verification and Applications*. Wiley-ISTE, 2009.

[DL20] A. Dixon and R. Lazić. "KReach: A Tool for Reachability in Petri Nets". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 12078. Lecture Notes in Computer Science. Springer, 2020. DOI: 10.1007/978-3-030-45190-5_22.

[DS97] A. Dolzmann and T. Sturm. "REDLOG: computer algebra meets computer logic". In: *ACM SIGMA Bulletin* 31.2 (1997), pp. 2–9. DOI: 10.1145/261320.261324.

[DFS98] C. Dufourd, A. Finkel, and P. Schnoebelen. "Reset nets between decidability and undecidability". In: *Automata, Languages and Programming (ICALP)*. Vol. 1443. Lecture Notes in Computer Science. Springer, 1998. DOI: 10.1007/BFb0055044.

[EC80] E. A. Emerson and E. M. Clarke. "Characterizing correctness properties of parallel programs using fixpoints". In: *Automata, Languages and Programming (ICALP)*. Vol. 85. Lecture Notes in Computer Science. Springer, 1980. DOI: 10.1007/3-540-10003-2_69.

[Esp98] J. Esparza. "Decidability and complexity of Petri net problems — An introduction". In: *Lectures on Petri Nets I: Basic Models (ACPN)*. Lecture Notes in Computer Science. Springer, 1998. DOI: 10.1007/3-540-65306-6_20.

[Esp+14] J. Esparza, R. Ledesma-Garza, R. Majumdar, P. Meyer, and F. Niksic. "An SMT-Based Approach to Coverability Analysis". In: *Computer Aided Verification (CAV)*. Vol. 8559. Lecture Notes in Computer Science. 2014. DOI: 10.1007/978-3-319-08867-9_40.

[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.

[EN94] J. Esparza and M. Nielsen. "Decidability issues for Petri nets". In: *BRICS Report Series* 1.8 (1994). DOI: 10.7146/brics.v1i8.21662.

[ES01] J. Esparza and C. Schröter. "Net Reductions for LTL Model-Checking". In: *Correct Hardware Design and Verification Methods (CHARME)*. Vol. 2144. Lecture Notes in Computer Science. Springer. 2001. DOI: 10.1007/3-540-44798-9_25.

[EHP05] S. Evangelista, S. Haddad, and J. Pradat-Peyre. "Syntactical Colored Petri Nets Reductions". In: *Automated Technology for Verification and Analysis (ATVA)*. Vol. 3707. Lecture Notes in Computer Science. Springer, 2005. DOI: 10.1007/11562948_17.

[Fea96] P. Feautrier. "Automatic parallelization in the polytope model". In: *The Data Parallel Programming Model*. Vol. 1132. Lecture Notes in Computer Science. Springer, 1996. DOI: 10.1007/3-540-61736-1_44.

[FL11] P. Feautrier and C. Lengauer. "Polyhedron Model". In: *Encyclopedia of Parallel Computing* (2011), pp. 1581–1592. DOI: 10.1007/978-0-387-09766-4_502.

[Fin92] G. Findlow. "Obtaining deadlock-preserving skeletons for coloured nets". In: *Application and Theory of Petri Nets (ICATPN)*. Vol. 616. Lecture Notes in Computer Science. Springer, 1992. DOI: 10.1007/3-540-55676-1_10.

[Fin91] A. Finkel. "The minimal coverability graph for Petri nets". In: *Advances in Petri Nets (ICATPN)*. Vol. 674. Lecture Notes in Computer Science. Springer, 1991. DOI: 10.1007/3-540-56689-9_45.

[FHK21] A. Finkel, S. Haddad, and I. Khmelnitsky. "Commodification of accelerations for the Karp and Miller Construction". In: *Discrete Event Dynamic Systems* 31.2 (2021), pp. 251–270. DOI: 10.1007/s10626-020-00331-z.

[FL02] A. Finkel and J. Leroux. "How to Compose Presburger-Accelerations: Applications to Broadcast Protocols". In: *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Vol. 2556. Lecture Notes in Computer Science. Springer, 2002. DOI: 10.1007/3-540-36206-1_14.

[FR98] M. J. Fischer and M. O. Rabin. "Super-Exponential Complexity of Presburger Arithmetic". In: *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, 1998. DOI: 10.1007/978-3-7091-9459-1_5.

[GRV08] P. Ganty, J.-F. Raskin, and L. Van Begin. "From Many Places to Few: Automatic Abstraction Refinement for Petri Nets". In: *Fundamenta Informaticae* 88.3 (2008), pp. 275–305.

[Gar19] H. Garavel. "Nested-unit Petri nets". In: *Journal of Logical and Algebraic Methods in Programming* 104 (2019), pp. 60–85. DOI: 10.1016/j.jlamp.2018.11.005.

[Gar15] H. Garavel. "Nested-Unit Petri Nets: A Structural Means to Increase Efficiency and Scalability of Verification on Elementary Nets". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 9115. Lecture Notes in Computer Science. Springer, 2015. DOI: 10.1007/978-3-319-19488-2_9.

[Gar20] H. Garavel. *Proposal for Adding Useful Features to Petri-Net Model Checkers*. Research report 03087421. Inria Grenoble, 2020.

[Gar12] H. Garavel. "Three Decades of Success Stories in Formal Methods". In: *Formal Methods for Industrial Critical Systems (FMICS)*. 2012.

[GLS17] H. Garavel, F. Lang, and W. Serwe. "From LOTOS to LNT". In: *ModelEd, TestEd, TrustEd*. Vol. 10500. Lecture Notes in Computer Science. Springer, 2017. DOI: 10.1007/978-3-319-68270-9_1.

[GS04] H. Garavel and W. Serwe. "State Space Reduction for Process Algebra Specifications". In: *Algebraic Methodology and Software Technology (AMAST)*. Vol. 3116. Lecture Notes in Computer Science. Springer, 2004. DOI: 10.1007/978-3-540-27815-3_16.

[GS06] H. Garavel and W. Serwe. "State space reduction for process algebra specifications". In: *Theoretical Computer Science* 351.2 (2006), pp. 131–145. DOI: 10.1016/j.tcs.2005.09.064.

[GS90] H. Garavel and J. Sifakis. "Compilation and verification of LOTOS specifications." In: *Protocol Specification, Testing and Verification (PSTV)*. North-Holland, 1990.

[GS92] S. M. German and A. P. Sistla. "Reasoning about systems with many processes". In: *Journal of the ACM* 39.3 (1992), pp. 675–735. DOI: 10.1145/146637.146681.

[GS66] S. Ginsburg and E. Spanier. "Semigroups, Presburger formulas, and languages". In: *Pacific journal of Mathematics* 16.2 (1966), pp. 285–296. DOI: 10.2140/pjm.1966.16.285.

[GDS92] A. Giua, F. DiCesare, and M. Silva. "Generalized mutual exclusion contraints on nets with uncontrollable transitions". In: *Systems, Man, and Cybernetics (SMC)*. IEEE, 1992. DOI: 10.1109/ICSMC.1992.271666.

[GW94] P. Godefroid and P. Wolper. "A Partial Approach to Model Checking". In: *Information and Computation* 110.2 (1994), pp. 305–326. DOI: 10.1006/inco.1994.1035.

[Gre78] S. A. Greibach. "Remarks on blind and partially blind one-way multicounter machines". In: *Theoretical Computer Science* 7.3 (1978), pp. 311–324. DOI: 10.1016/0304-3975(78)90020-8.

[Haa18] C. Haase. "A survival guide to Presburger arithmetic". In: *ACM SIGLOG News* 5.3 (2018), pp. 67–82. DOI: 10.1145/3242953.3242964.

[Hac76] M. H. T. Hack. "Decidability Questions for Petri Nets". PhD thesis. Massachusetts Institute of Technology, 1976.

[HP06] S. Haddad and J.-F. Pradat-Peyre. "New Efficient Petri Nets Reductions for Parallel Programs Verification". In: *Parallel Processing Letters* 16.01 (2006), pp. 101–116. DOI: 10.1142/S0129626406002502.

[Haj14] Á. Hajdu. "Extensions to the CEGAR approach on Petri nets". In: *Acta Cybernetica* 21.3 (2014), pp. 401–417. DOI: 10.14232/actacyb.21.3.2014.8.

[Ham+06] A. Hamez, L. Hillah, F. Kordon, A. Linard, E. Paviot-Adet, X. Renault, and Y. Thierry-Mieg. "New features in CPN-AMI 3: focusing on the analysis of complex distributed systems". In: *Application of Concurrency to System Design (ACSD)*. IEEE, 2006. DOI: 10.1109/ACSD.2006.15.

[Hau90] D. Hauschildt. "Semilinearity of the reachability set is decidable for Petri nets". PhD thesis. University of Hamburg, Germany, 1990.

[HHP09] M. Heizmann, J. Hoenicke, and A. Podelski. "Refinement of Trace Abstraction". In: *Static Analysis (SAS)*. Vol. 5673. Lecture Notes in Computer Science. Springer, 2009. DOI: 10.1007/978-3-642-03237-0_7.

[Hel01] K. Heljanko. "Bounded Reachability Checking with Process Semantics". In: *Concurrency Theory (CONCUR)*. Vol. 2154. Lecture Notes in Computer Science. Springer, 2001. DOI: 10.1007/3-540-44685-0_15.

[Hen+23] E. G. Henriksen, A. M. Khorsid, E. Nielsen, T. Risager, J. Srba, A. M. Stück, and A. S. Sørensen. "Potency-Based Heuristic Search with Randomness for Explicit Model Checking". In: *Model Checking Software (SPIN)*. Vol. 13872. Lecture Notes in Computer Science. Springer, 2023. DOI: 10.1007/978-3-031-32157-3_10.

[Hil+10] L.-M. Hillah, F. Kordon, L. Petrucci, and N. Treves. "PNML Framework: An Extendable Reference Implementation of the Petri Net Markup Languag". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 6128. Lecture Notes in Computer Science. Springer, 2010. DOI: 10.1007/978-3-642-13675-7_20.

[Hir94] Y. Hirshfeld. "Petri nets and the equivalence problem". In: *Computer Science Logic (CSL)*. Vol. 832. Lecture Notes in Computer Science. Springer, 1994. DOI: 10.1007/BFb0049331.

[Hla+21] P.-E. Hladik, F. Ingrand, S. Dal Zilio, and R. Tekin. "Hippo: A formal-model execution engine to control and verify critical real-time systems". In: *Journal of Systems and Software* 181 (2021). DOI: 10.1016/j.jss.2021.111033.

[Hoa78] C. A. R. Hoare. "Communicating sequential processes". In: *Communications of the ACM* 21.8 (1978), pp. 666–677. DOI: 10.1145/359576.359585.

[HB12] K. Hoder and N. Bjørner. "Generalized Property Directed Reachability". In: *Theory and Applications of Satisfiability Testing (SAT)*. Vol. 7317. Lecture Notes in Computer Science. Springer, 2012. DOI: 10.1007/978-3-642-31612-8_13.

[HK10] A. J. Hoffman and J. B. Kruskal. "Integral Boundary Points of Convex Polyhedra". In: *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 49–76. DOI: 10.1007/978-3-540-68279-0_3.

[HP79] J. Hopcroft and J.-J. Pansiot. "On the reachability problem for 5-dimensional vector addition systems". In: *Theoretical Computer Science* 8.2 (1979), pp. 135–159. DOI: 10.1016/0304-3975(79)90041-0.

[Hub+85] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen. "Towards reachability trees for high-level Petri nets". In: *Advances in Petri Nets*. Vol. 188. Lecture Notes in Computer Science. Springer, 1985. DOI: 10.1007/3-540-15204-0_13.

[Huj+20a] T. Hujsa, B. Berthomieu, S. D. Zilio, and D. L. Botlan. *Checking marking reachability with the state equation in Petri net subclasses*. 2020. arXiv: 2006.05600 [cs.LO].

[Huj+20b] T. Hujsa, B. Berthomieu, S. D. Zilio, and D. L. Botlan. *On the Petri Nets with a Single Shared Place and Beyond*. 2020. arXiv: 2005.04818 [cs.DS].

[HLL92] T. Huynh, C. Lassez, and J.-L. Lassez. "Practical issues on the projection of polyhedral sets". In: *Annals of Mathematics and Artificial Intelligence* 6.4 (1992), pp. 295–315. DOI: 10.1007/BF01535523.

[IR93] J.-M. Ilié and O. Rojas. "On well-formed nets and optimizations in enabling tests". In: *Application and Theory of Petri Nets (ICATPN)*. Vol. 691. Lecture Notes in Computer Science. Springer, 1993.

[Imb93] J.-L. Imbert. "Fourier's elimination: Which to choose?" In: *Principles and Practice of Constraint Programming (PPCP)*. 1993.

[INR] INRIA. *CADP*. URL: https://cadp.inria.fr/ (visited on 10/10/2023).

[ISO89] ISO/IEC. *LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. Standard 8807. International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, 1989. URL: https://www.iso.org/standard/16258.html.

[Jan94] P. Jančar. "Decidability questions for bisimilarity of Petri nets and some related problems". In: *Symposium on Theoretical Aspects of Computer Science (STACS)*. Vol. 775. Lecture Notes in Computer Science. Springer, 1994. DOI: 10.1007/3-540-57785-8_173.

[Jan84] R. Janicki. “Nets, sequential components and concurrency relations”. In: *Theoretical Computer Science* 29.1-2 (1984), pp. 87–121. DOI: 10.1016/0304-3975(84)90014-8.

[JM09] B. Jeannet and A. Miné. “Apron: A Library of Numerical Abstract Domains for Static Analysis”. In: *Computer Aided Verification (CAV)*. Vol. 5643. Lecture Notes in Computer Science. Springer, 2009. DOI: 10.1007/978-3-642-02658-4_52.

[Jen+16] J. F. Jensen, T. Nielsen, L. K. Oestergaard, and J. Srba. “TAPAAL and Reachability Analysis of P/T Nets”. In: *Transactions on Petri Nets and Other Models of Concurrency XI*. Lecture Notes in Computer Science 9930 (2016), pp. 307–318. DOI: 10.1007/978-3-662-53401-4_16.

[Jen87] K. Jensen. “Coloured Petri Nets”. In: *Petri Nets: Central Models and Their Properties (ACPN)*. Vol. 254. Lecture Notes in Computer Science. Springer, 1987. DOI: 10.1007/978-3-540-47919-2_10.

[Jen81] K. Jensen. “Coloured petri nets and the invariant-method”. In: *Theoretical Computer Science* 14.3 (1981), pp. 317–336. DOI: https://doi.org/10.1016/0304-3975(81)90049-9.

[Jen96] K. Jensen. “Condensed state spaces for symmetrical Coloured Petri nets”. In: *Formal Methods in System Design* 9.1 (1996), pp. 7–40. DOI: 10.1007/BF00625967.

[Jen83] K. Jensen. “High-Level Petri Nets”. In: *Applications and Theory of Petri Nets*. Vol. 66. Lecture Notes in Computer Science. Springer, 1983. DOI: 10.1007/978-3-642-69028-0_12.

[JP20] L. Jezequel and E. Paviot-Adet. *Model Checking Contest: The Property Language Manual*. Standard. 2020.

[Kah74] G. Kahn. “The Semantics of a Simple Language for Parallel Programming”. In: *Information Processing (IFIP)*. North-Holland, 1974.

[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), pp. 1–29. DOI: 10.1145/2629608.

[KBJ21] J. Kang, Y. Bai, and L. Jiao. “Abstraction-Based Incremental Inductive Coverability for Petri Nets”. In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 12734. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-76983-3_19.

[Kar12] A. Karatkevich. “Conditions of SM-Coverability of Petri Nets”. In: *Boolean Problems*. 2012. DOI: 10.13140/2.1.2162.1762.

[KM69] R. M. Karp and R. E. Miller. “Parallel program schemata”. In: *Journal of Computer and System Sciences* 3.2 (1969), pp. 147–195. DOI: 10.1016/S0022-0000(69)80011-5.

[KKG18] Y. I. Khan, A. Konios, and N. Guelfi. “A Survey of Petri Nets Slicing”. In: *ACM Computing Surveys* 51.5 (2018), pp. 1–32. DOI: 10.1145/3241736.

[Klo+13] J. Kloos, R. Majumdar, F. Niksic, and R. Piskac. “Incremental, Inductive Coverability”. In: *Computer Aided Verification (CAV)*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013. DOI: 10.1007/978-3-642-39799-8_10.

[Kor+21a] F. Kordon, P. Bouvier, H. Garavel, L. M. Hillah, F. Hulin-Hubard, N. Amat, E. Amparore, B. Berthomieu, S. Biswal, D. Donatelli, F. Galla, S. Dal Zilio, P. G. Jensen, L. Jezequel, C. He, D. Le Botlan, S. Li, E. Paviot-Adet, J. Srba, Y. Thierry-Mieg, A. Walner, and K. Wolf. *Complete Results for the 2021 Edition of the Model Checking Contest*. 2021. URL: http://mcc.lip6.fr/2021/results.php (visited on 10/10/2023).

[Kor+22] F. Kordon, P. Bouvier, H. Garavel, F. Hulin-Hubard, N. Amat, E. Amparore, B. Berthomieu, D. Donatelli, S. Dal Zilio, P. G. Jensen, L. Jezequel, C. He, S. Li, E. Paviot-Adet, J. Srba, and Y. Thierry-Mieg. *Complete Results for the 2022 Edition of the Model Checking Contest*. 2022. URL: http://mcc.lip6.fr/2022/results.php (visited on 10/10/2023).

[Kor+23] F. Kordon, P. Bouvier, H. Garavel, F. Hulin-Hubard, N. Amat, E. Amparore, B. Berthomieu, D. Donatelli, S. Dal Zilio, P. G. Jensen, L. Jezequel, E. Paviot-Adet, J. Srba, and Y. Thierry-Mieg. *Complete Results for the 2023 Edition of the Model Checking Contest*. 2023. URL: https://mcc.lip6.fr/2023/results.php (visited on 10/10/2023).

[Kor15] F. Kordon. *Model entitled "SmallOperatingSystem" from the Model Checking Contest benchmark*. 2015. URL: https://mcc.lip6.fr/2023/pdf/SmallOperatingSystem-form.pdf (visited on 10/10/2023).

[Kor+21b] F. Kordon, L. M. Hillah, F. Hulin-Hubard, L. Jezequel, and E. Paviot-Adet. "Study of the efficiency of model checking techniques using results of the MCC from 2015 To 2019". In: *International Journal on Software Tools for Technology Transfer* 23.6 (2021), pp. 931–952. DOI: 10.1007/s10009-021-00615-1.

[KLP06] F. Kordon, A. Linard, and E. Paviot-Adet. "Optimized Colored Nets Unfolding". In: *Formal Techniques for Networked and Distributed Systems (FORTE)*. Vol. 4229. Lecture Notes in Computer Science. Springer, 2006. DOI: 10.1007/11888116_25.

[Kos82] S. R. Kosaraju. "Decidability of Reachability in Vector Addition Systems". In: *Symposium on Theory of Computing (STOC)*. ACM, 1982. DOI: 10.1145/800070.802201.

[Kov92] A. V. Kovalyov. "Concurrency relations and the safety problem for Petri nets". In: *Application and Theory of Petri Nets (ICATPN)*. Vol. 616. Lecture Notes in Computer Science. Springer, 1992. DOI: 10.1007/3-540-55676-1_17.

[Kov00] A. Kovalyov. "A Polynomial Algorithm to Compute the Concurrency Relation of a Regular STG". In: *Hardware Design and Petri Nets*. Springer, 2000, pp. 107–126. DOI: 10.1007/978-1-4757-3143-9_6.

[KE96] A. Kovalyov and J. Esparza. "A Polynomial Algorithm to Compute the Concurrency Relation of Free-choice Signal Transition Graphs". In: *Workshop on Discrete Event Systems (WODES)*. IEEE, 1996.

[KS08] D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View*. Springer, 2008.

[LAA23] LAAS-CNRS. *Tina Toolbox*. 2023. URL: http://projects.laas.fr/tina (visited on 10/10/2023).

[Lam92] J. L. Lambert. "A structure to decide reachability in Petri nets". In: *Theoretical Computer Science* 99.1 (1992), pp. 79–104. DOI: 10.1016/0304-3975(92)90173-d.

[Lam90] J. L. Lambert. *Vector addition systems and semi-linearity*. Université Paris-Nord. Centre Scientifique et Polytechnique [CSP], 1990.

[LS07] A. Lasaruk and T. Sturm. "Weak quantifier elimination for the full linear theory of the integers: A uniform generalization of Presburger arithmetic". In: *Applicable Algebra in Engineering, Communication and Computing* 18.6 (2007), pp. 545–574. DOI: 10.1007/s00200-007-0053-x.

[LAG15] H. Leroux, D. Andreu, and K. Godary-Dejean. “Handling Exceptions in Petri Net-Based Digital Architecture: From Formalism to Implementation on FPGAs”. In: *IEEE Transactions on Industrial Informatics* 11.4 (2015), pp. 897–906. DOI: 10.1109/TII.2015.2435696.

[Ler21] J. Leroux. “Flat Petri Nets (Invited Talk)”. In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 12734. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-76983-3_2.

[Ler13] J. Leroux. “Presburger Vector Addition Systems”. In: *Logic in Computer Science (LICS)*. IEEE, 2013. DOI: 10.1109/LICS.2013.7.

[Ler09] J. Leroux. “The General Vector Addition System Reachability Problem by Presburger Inductive Invariants”. In: *Logic in Computer Science (LICS)*. IEEE, 2009. DOI: 10.1109/LICS.2009.10.

[Ler22] J. Leroux. “The Reachability Problem for Petri Nets is Not Primitive Recursive”. In: *Foundations of Computer Science (FOCS)*. IEEE, 2022. DOI: 10.1109/FOCS52979.2021.00121.

[Ler11] J. Leroux. “Vector addition system reachability problem: a short self-contained proof”. In: *Principles of Programming Languages (POPL)*. ACM, 2011. DOI: 10.1145/1926385.1926421.

[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.

[LP09] J. Leroux and G. Point. “TaPAS: the Talence Presburger arithmetic suite”. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 5505. Lecture Notes in Computer Science. Springer, 2009. DOI: 10.1007/978-3-642-00768-2_18.

[LS19] J. Leroux and S. Schmitz. “Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension”. In: *Logic in Computer Science (LICS)*. IEEE, 2019. DOI: 10.1109/LICS.2019.8785796.

[LS05] J. Leroux and G. Sutre. “Flat Counter Automata Almost Everywhere!” In: *Automated Technology for Verification and Analysis (ATVA)*. Vol. 3707. Lecture Notes in Computer Science. Springer, 2005. DOI: 10.1007/11562948_36.

[Lip76] R. Lipton. *The Reachability Problem Requires Exponential Space*. Research report 63. Department of Computer Science, Yale University, 1976.

[Lip75] R. J. Lipton. “Reduction: a method of proving properties of parallel programs”. In: *Communications of the ACM* 18.12 (1975), pp. 717–721. DOI: 10.1145/361227.361234.

[LHY12] F. Liu, M. Heiner, and M. Yang. “An efficient method for unfolding colored Petri nets”. In: *Winter Simulation Conference (WSC)*. IEEE, 2012.

[Llo+17] M. Llorens, J. Oliver, J. Silva, and S. Tamarit. “An Integrated Environment for Petri Net Slicing”. In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 10258. Lecture Notes in Computer Science. Springer, 2017. DOI: 10.1007/978-3-319-57861-3_8.

[LAV91] J. C. Lloret, P. Azéma, and F. Vernadat. “Compositional design and verification of communication protocols, using labelled Petri nets”. In: *Computer-Aided Verification (CAV)*. Vol. 531. Lecture Notes in Computer Science. Springer, 1991. DOI: 10.1007/BFb0023723.

[LT88] N. A. Lynch and M. R. Tuttle. *An Introduction to Input/Output Automata*. Research report. Laboratory for Computer Science, Massachusetts Institute of Technology, 1988.

[Mäk01] M. Mäkelä. "Optimising Enabling Tests and Unfoldings of Algebraic System Nets". In: *Applications and Theory of Petri Nets (ICATPN)*. Vol. 2075. Lecture Notes in Computer Science. Springer, 2001. DOI: 10.1007/3-540-45740-2_17.

[May81] E. W. Mayr. "An Algorithm for the General Petri Net Reachability Problem". In: *Symposium on Theory of Computing (STOC)*. ACM, 1981. DOI: 10.1145/800076.802477.

[McM93] K. L. McMillan. *Symbolic Model Checking*. Springer, 1993. DOI: 10.1007/978-1-4615-3190-6.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. Springer, 1980. DOI: 10.1007/3-540-10235-3.

[MPW92] R. Milner, J. Parrow, and D. Walker. "A calculus of mobile processes, I". In: *Information and Computation* 100.1 (1992), pp. 1–40. DOI: 10.1016/0890-5401(92)90008-4.

[Min06] A. Miné. "The octagon abstract domain". In: *Higher-order and Symbolic Computation* 19.1 (2006), pp. 31–100. DOI: 10.1007/s10990-006-8609-1.

[Mon10] D. Monniaux. "Quantifier Elimination by Lazy Model Enumeration". In: *Computer Aided Verification (CAV)*. Vol. 6174. Lecture Notes in Computer Science. Springer, 2010. DOI: 10.1007/978-3-642-14295-6_51.

[MB08] L. de Moura and N. Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008. DOI: 10.1007/978-3-540-78800-3_24.

[Mur77] T. Murata. "State equation, controllability, and maximal matchings of Petri nets". In: *Transactions on Automatic Control* 22.3 (1977), pp. 412–416. DOI: 10.1109/TAC.1977.1101509.

[MK80] T. Murata and J. Koh. "Reduction and expansion of live and safe marked graphs". In: *Transactions on Circuits and Systems* 27.1 (1980), pp. 68–71. DOI: 10.1109/TCS.1980.1084711.

[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.

[Net+07] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. "MiniZinc: Towards a Standard CP Modelling Language". In: *Principles and Practice of Constraint Programming (CP)*. Vol. 4741. Lecture Notes in Computer Science. Springer, 2007. DOI: 10.1007/978-3-540-74970-7_38.

[Opp78] D. C. Oppen. "A 222pn 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.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.

[PTG22] F. Pommereau, C. Thomas, and C. Gaucherel. “Petri Nets Semantics of Reaction Rules (RR) A Language for Ecosystems Modelling”. In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 13288. Lecture Notes in Computer Science. Springer, 2022. DOI: 10.1007/978-3-031-06653-5_10.

[Pre29] M. Presburger. “Uber die vollstandigkeiteines gewissen systems der arithmetik ganzer zahlen, in welchen die addition als einzige operation hervortritt”. In: *Comptes-Rendus du ler Congres des Mathematiciens des Pays Slavs*. 1929.

[PJ91] M. Presburger and D. Jacquette. “On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation”. In: *History and Philosophy of Logic* 12.2 (1991), pp. 225–233. DOI: 10.1080/01445340910883718 7.

[Pug91] W. Pugh. “The Omega Test: a fast and practical integer programming algorithm for dependence analysis”. In: *Supercomputing*. ACM, 1991. DOI: 10.1145/125826.125848.

[QS82] J.-P. Queille and J. Sifakis. “Specification and verification of concurrent systems in CESAR”. In: *International Symposium on Programming (Programming)*. Vol. 137. Lecture Notes in Computer Science. Springer, 1982. DOI: 10.1007/3-540-11494-7_22.

[Rac78] C. Rackoff. “The covering and boundedness problems for vector addition systems”. In: *Theoretical Computer Science* 6.2 (1978), pp. 223–231. DOI: 10.1016/0304-3975(78)90036-1.

[Rak12] A. Rakow. “Safety Slicing Petri Nets”. In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 7347. Lecture Notes in Computer Science. Springer, 2012. DOI: 10.1007/978-3-642-31131-4_15.

[Rei12] W. Reisig. *Petri nets: An Introduction*. Springer, 2012. DOI: 10.1007/978-3-642-69968-9.

[Sch00] K. Schmidt. “How to calculate symmetries of Petri nets”. In: *Acta Informatica* 36.7 (2000), pp. 545–590. DOI: 10.1007/s002360050002.

[Sch03] K. Schmidt. “Using Petri Net Invariants in State Space Construction”. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 2619. Lecture Notes in Computer Science. Springer, 2003. DOI: 10.1007/3-540-36577-X_35.

[Sch16] S. Schmitz. “The complexity of reachability in vector addition systems”. In: *ACM SIGLOG News* 3.1 (2016), pp. 4–21. DOI: 10.1145/2893582.2893585.

[SY95] A. Semenov and A. Yakovlev. “Combining partial orders and symbolic traversal for efficient verification of asynchronous circuits”. In: *ASP-DAC’95/CHDL’95/VLSI’95 with EDA Technofair*. IEEE, 1995. DOI: 10.1109/ASPDAC.1995.486371.

[SJJ91] Y. Shanlin, Z. Jie, and G. Jun. “Inverse petri nets: properties and applications”. In: *IFAC Proceedings Volumes* 24.14 (1991), pp. 91–95. DOI: 10.1016/S1474-6670(17)69330-3.

[SSS00] M. Sheeran, S. Singh, and G. Stålmarck. “Checking Safety Properties Using Induction and a SAT-Solver”. In: *Formal Methods in Computer-Aided Design (FMCAD)*. Vol. 1954. Lecture Notes in Computer Science. Springer, 2000. DOI: 10.1007/3-540-40922-X_8.

[STC96] M. Silva, E. Terue, and J. M. Colom. "Linear algebraic and linear programming techniques for the analysis of place/transition net systems". In: *Lectures on Petri Nets I: Basic Models (ACPN)*. Vol. 1491. Lecture Notes in Computer Science. Springer, 1996. DOI: 10.1007/3-540-65306-6_19.

[Sta91] P. H. Starke. "Reachability Analysis of Petri Nets Using Symmetries". In: *Systems Analysis Modelling Simulation* 8.4-5 (1991), pp. 293–303. DOI: 10.5555/115220.115224.

[Thi20] Y. Thierry-Mieg. "Structural Reductions Revisited". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 12152. Lecture Notes in Computer Science. Springer, 2020. DOI: 10.1007/978-3-030-51831-8_15.

[Thi21] Y. Thierry-Mieg. "Symbolic and Structural Model-Checking". In: *Fundamenta Informaticae* 183.3-4 (2021), pp. 319–342. DOI: 10.3233/FI-2021-2090.

[Thi15] Y. Thierry-Mieg. "Symbolic Model-Checking Using ITS-Tools". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 9035. Lecture Notes in Computer Science. Springer, 2015. DOI: 10.1007/978-3-662-46681-0_20.

[Thi+09] Y. Thierry-Mieg, D. Poitrenaud, A. Hamez, and F. Kordon. "Hierarchical Set Decision Diagrams and Regular Models". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 5505. Lecture Notes in Computer Science. Springer, 2009. DOI: 10.1007/978-3-642-00768-2_1.

[Val88] A. Valmari. "Error detection by reduced reachability graph generation". In: *Proceedings of the 9th European Workshop on Application and Theory of Petri Nets*. 1988.

[Vau87] J. Vautherin. "Parallel systems specifications with coloured Petri nets and algebraic specifications". In: *Advances in Petri Nets (APN)*. Vol. 266. Lecture Notes in Computer Science. Springer, 1987. DOI: 10.1007/3-540-18086-9_31.

[Ver10] S. Verdoolaege. "isl: An Integer Set Library for the Polyhedral Model". In: *Mathematical Software (ICMS)*. Vol. 6327. Lecture Notes in Computer Science. Springer, 2010. DOI: 10.1007/978-3-642-15582-6_49.

[WW22] S. Wallner and K. Wolf. "Skeleton Abstraction for Universal Temporal Properties". In: *Fundamenta Informaticae* 187.2–4 (2022), pp. 245–272. DOI: 10.3233/FI-222138.

[Wei84] M. Weiser. "Program Slicing". In: *Transactions on Software Engineering* SE-10.4 (1984), pp. 352–357. DOI: 10.1109/TSE.1984.5010248.

[Wim13] H. Wimmel. *Sara: Structures for Automated Reachability Analysis*. 2013. URL: https://github.com/nlohmann/service-technology.org/tree/master/sara (visited on 10/10/2023).

[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.

[Wiś+14] R. Wiśniewski, A. Karatkevich, M. Adamski, and D. Kur. "Application of comparability graphs in decomposition of Petri nets". In: *Human System Interactions (HSI)*. IEEE, 2014. DOI: 10.1109/HSI.2014.6860478.

[WWJ19] R. Wiśniewski, M. Wiśniewska, and M. Jarnut. "C-Exact Hypergraphs in Concurrency and Sequentiality Analyses of Cyber-Physical Systems Specified by Safe Petri Nets". In: *IEEE Access* 7 (2019), pp. 13510–13522. DOI: 10.1109/ACCESS.2019.2893284.

[Wol07] K. Wolf. "Generating Petri Net State Spaces". In: *Petri Nets and Other Models of Concurrency (ICATPN)*. Vol. 4546. Lecture Notes in Computer Science. Springer, 2007. DOI: 10.1007/978-3-540-73094-1_5.

[Wol18] K. Wolf. "Petri Net Model Checking with LoLA 2". In: *Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 10877. Lecture Notes in Computer Science. Springer, 2018. DOI: 10.1007/978-3-319-91268-4_18.