Improving Explicit Model Checking for Petri Nets

Dissertation

to obtain the academic degree of

Doktor-Ingenieur (Dr.-Ing.)

of the Faculty of Computer Science and Electrical Engineering

at the University of Rostock

submitted by

Dipl.-Inf. Torsten Liebke

born on October 9, 1985 in Rostock

Rostock, December 11, 2020

https://doi.org/10.18453/rosdok_id00003271

First reviewer: Prof. Dr. rer. nat. habil. Karsten Wolf
University of Rostock, Germany

Second reviewer: Prof. Jirí Srba, PhD
University of Aalborg, Denmark

Date of submission: December 11, 2020

Date of defense: September 15, 2021

Acknowledgments

First of all, my sincerest thanks go to my supervisor Karsten Wolf for giving me the opportunity to work on my dissertation. I also thank him for his trust and support throughout all stages of this work in so many different ways. This thesis would have not been possible without him. The endless discussions on model checking shaped this work.
In addition, I would like to thank the second reviewer, who was unknown at the time of printing. Second, it is my pleasure to thank Jirí Srba for agreeing to review my thesis and for hosting me Aalborg a couple of years back. The ongoing competition between TAPAAL and LoLA in the yearly model checking contest has been a driving force in my work.
Third, I am grateful to Christian Rosenke for always helping out, reading drafts of this thesis and taking the time to give me insightful comments.
I also want to thank all my friends for distracting me when I get stuck at work.
A very special thanks goes to my partner Angelika for her tremendous support and motivation during this time.
Finally, I want to say a big thank-you to my parents, my brother, and my grandparents, who supported me throughout my whole life and made this possible in the first place.
Thank you all!

Abstract

Model checking is the automated verification that systematically checks if a given behavioral property holds for a given model of a system. For modeling, analyzing, and verifying systems with mathematical rigor, formal methods are used, which can be considered as applied mathematics. We use Petri nets and temporal logic as formalisms to describe a system and its behavior in a mathematically precise and unambiguous manner.

Explicit state space verification explores all possible states of a system one by one to prove or disprove the given property. The problem is that the state space usually grows exponentially fast with the size of the model. This is called the state explosion problem and handling it, is a main challenge in model checking today. Although the number of states can be enormous, modern techniques have accomplished the successful verification of large industrial systems due to the greatly reduced size of state spaces. Nevertheless, these techniques are often not powerful enough for some very large and complex systems that we see today. Thus, it is still important to come up with refined or even new methods to reduce the number of states that have to be explored.

The contributions of this thesis are concerned with the improvement of model checking efficiency both in theory and in practice.

At first, we present two new reduction techniques that can be used in a portfolio setting, which means that several methods run in parallel and the fastest successful method determines the runtime. In the category of scalable models, there is usually a parameter to scale over the structure or over the resources of the model. To verify a property in the latter category, it is sometimes sufficient to consider the model with very limited resources. We propose a technique utilizing this idea that considerably reduces the state space. The second reduction technique that we introduce is based on the counterexample guided abstraction refinement (CEGAR) method, which, so far, could only be used for reachability problems. We demonstrate that our CEGAR approach works for a whole class of temporal logic formulas.

For our second contribution, we provide certain supplementary strength re-

ductions, that is, techniques that complement the model checking process. More precisely, we introduce formula simplifications based on structural methods and propose quick checks for the fast verification of certain necessary or sufficient conditions of the input formula. Furthermore, we introduce a new and faster algorithm to compute conflicting transitions.
The third and final contribution of this thesis is all about stubborn sets, a dialect of partial order reduction. In fact, in LTL model checking, the formula is represented by an automaton. We use the rather unexplored idea to utilize all available information from this automaton in order to compute smaller stubborn sets. In addition, we propose specialized stubborn sets for simple and frequently occurring CTL formulas.
When possible, we implemented the proposed algorithms as a proof-of-concept in our explicit model checker LoLA and validated them with the benchmark provided by the annual model checking contest. This includes all but two methods, when implementation was hampered by an internal incompatibility with the architecture of LoLA. Our experiments show that all implemented techniques increase the model checking efficiency.

Zusammenfassung

Model Checking ist die automatisierte systematische Überprüfung, ob eine gegebene Verhaltenseigenschaft für ein gegebenes Modell eines Systems erfüllt ist. Zur Modellierung, Analyse und Verifikation von Systemen mit mathematischer Rigorosität werden formale Methoden verwendet, die als angewandte Mathematik betrachtet werden können. Wir verwenden Petrinetze und temporale Logik als Formalismen, um ein System und sein Verhalten mathematisch präzise und eindeutig zu beschreiben.

Explizite Zustandsraumverifikation untersucht nacheinander alle möglichen Zustände eines Systems, um die gegebene Eigenschaft zu beweisen oder zu widerlegen. Das Problem ist, dass der Zustandsraum in der Regel exponentiell schnell mit der Größe des Modells wächst. Dies wird als das Zustandsexplosionsproblem bezeichnet und der Umgang damit ist heutzutage eine der größten Herausforderungen beim Model Checking. Obwohl die Anzahl der Zustände riesig sein kann, haben moderne Techniken die Verifikation großer industrieller Systeme ermöglicht. Diese Techniken reduzieren die Größe des Zustandsraums erheblich und ermöglichen so die erfolgreiche Verifikation größerer Systeme. Dennoch sind sie für einige sehr große und komplexe Systeme, die wir heutzutage sehen, oft nicht leistungsfähig genug. Daher ist es nach wie vor wichtig, verbesserte oder sogar neue Methoden zu entwickeln, um die Anzahl der zu untersuchenden Zustände zu reduzieren.

Die Beiträge dieser Arbeit beschäftigten sich mit der Verbesserung der Effizienz des Model Checkings sowohl in der Theorie als auch in der Praxis. Zunächst stellen wir zwei neue Reduktionstechniken vor, die in einer Portfolioumgebung eingesetzt werden können, d.h. dass mehrere Methoden parallel laufen und die schnellste erfolgreiche Methode die Laufzeit bestimmt. In der Kategorie der skalierbaren Modelle gibt es normalerweise einen Parameter, der über die Struktur oder über die Ressourcen des Modells skaliert werden kann. Um eine Eigenschaft in dieser Kategorie zu verifizieren, ist es manchmal ausreichend, das Modell mit sehr begrenzten Ressourcen zu betrachten. Wir schlagen eine Technik vor, die diese Idee nutzt und den Zustandsraum erheblich reduziert. Die zweite Reduktionstechnik, die wir

vorstellen, basiert auf der CEGAR-Methode (Counterexample Guided Abstraction Refinement), die bisher nur bei Erreichbarkeitsproblemen eingesetzt werden konnte. Wir zeigen, dass unser CEGAR-Ansatz für eine ganze Klasse von Formeln der temporalen Logik funktioniert.
Unser zweiter Beitrag stellt zusätzliche Stärkereduzierung zur Verfügung, d.h. Techniken, die das Model Checking ergänzen. Genauer gesagt führen wir Formelvereinfachungen ein, die auf strukturellen Methoden basieren, und schlagen Quick Checks zur schnellen Überprüfung bestimmter notwendiger oder hinreichender Bedingungen der Eingabeformel vor. Darüber hinaus führen wir einen neuen und schnelleren Algorithmus zur Berechnung von Konflikt-Transitionen ein.
Im dritten und letzten Beitrag dieser Arbeit geht es um sture Mengen, einem Dialekt der Partial Order Reduction. Beim LTL Model Checking wird die Eingabeformel durch einen Automaten dargestellt. Wir verwenden die eher unerforschte Idee, alle verfügbaren Informationen aus diesem Automaten zu nutzen, um kleinere sture Mengen zu berechnen. Darüber hinaus schlagen wir spezielle sture Mengen für einfache und häufig vorkommende CTL Formeln vor.
Wenn möglich, haben wir die vorgestellten Algorithmen als Proof-of-Concept in unserem expliziten Model Checker LoLA implementiert und dann mit dem Benchmark des jährlichen Model Checking Contests validiert. Dies schließt alle Methoden bis auf zwei ein, bei denen die Implementierung durch eine interne Inkompatibilität mit der Architektur von LoLA erschwert wurde. Unsere Experimente zeigen, dass alle implementierten Techniken die Effizienz des Model Checkings erhöhen.

Contents

Part I

Introduction and preliminaries

In this part, we work out the area this thesis is concerned with. We first motivate the need for model checking, which means to verify systems with respect to a given specification. To this end, we present different formal frameworks to model systems and introduce temporal logic as a formal method for specifications. To do model checking, we combine the formal system models and the specification. The question is whether a given specification holds in a given system.
The remainder of this part is organized as follows. In the next Chapter 1, we motivate the research area we are interested in, state our research goal, and present the contributions we made in this area. We continue in Chapter 2 and Chapter 3 with the presentation of formal frameworks for systems and specifications. Finally, in Chapter 4 we describe the model checking algorithms we use for verification.

Chapter 1

About this thesis

Model checking [19, 21, 39, 104] is an established formal method to verify aspects of a given system. Over the years many different methods have been developed, which allow the use of model checking for industrial systems. Even today model checking is still a diverse and active field of research. In addition, there are sophisticated verification tools [34, 122, 142] available, which are still being actively developed. A general introduction to model checking and a description of the involved methods is given by Clarke et al. in [24] and by Baier and Katoen in [15]. In this chapter, we identify performance enhancement of model checking as an important and interesting research problem that we are going to address in this thesis.
The remainder of this chapter is organized as follows. In the next Section 1.1, we motivate the topic and introduce the abstract model checking process. Section 1.2 formulates the research goal of this thesis. We continue in Section 1.3 with a summary of our contributions to the research goal. Section 1.4 concludes this chapter with an outline of this thesis.

1.1 Motivation

Reactive and distributed systems, in the following only called systems, are part of nearly every aspect of today's life – they are ubiquitous. Such systems are used in software and hardware designs, including e-commerce, medical instruments, and hardware circuits, further they are used in protocols, business processes, biochemical behavior, and many other processes (as further reference see the list of models in the model checking contest (MCC) [68]). At the same time, such systems are becoming more and more complex, and thus the error-proneness is increasing. In 2017, software testing company Tricents analyzed 606 software failures at 314 companies [125]. The report

revealed that these software bugs caused USD 1.7 trillion in financial losses and affected 3.6 billion people.
The only possibility to guarantee the correctness of a system is to use formal methods. Because a formal investigation of the original system is usually impossible, it has to be translated into an *abstract model* that can be used to mathematically prove its correctness. The abstract model has to cover all important aspects of the original system that are relevant to verify a given specification. In this thesis, systems are represented by *Petri nets* [100, 101, 106]. Petri nets are an established formal method for modeling and verifying asynchronous, concurrent and distributed systems as the MCC shows and they are used in various industries [146]. The wide use of Petri nets has two reasons. First, they have a straightforward graphical representation enabling the modeling of various systems. Second, Petri nets have a strong mathematical foundation which allows a rigorous analyses and verification process.
Although formal methods are expensive, time-consuming, and require experts, they are the only way to actually verify aspects of a given system, which testing can never thoroughly achieve. Therefore, we have to cope somehow with the drawback that formal methods are usually not scaling well with the increasing size of the system under investigation.
Model checking is the fully automated verification of the question whether a given system meets a given specification. For this, the specification has to be *formalized* in some *logical formalism*, e.g., propositional or temporal logic [20, 40, 103]. Temporal logic, for instance, can rigorously assert how the behavior of the system evolves over time.
The mathematical representation of the system and the specification, namely the model and the logical formula, are both the input to the model checker. The model checker then tries to prove that the model meets the logical formula. The result is either "yes" (positive), "no" (negative), or "unknown".
If the verification fails, the model checker produces an error trace which basically represents a counterexample for the investigated specification. Usually, a simulation of the counterexample helps the operator of the model checker to redesign the system, refine the model, or rephrase the logical formula. After the input is modified, the model checking algorithm is applied again. This basic model checking procedure is illustrated in Figure 1.1.
The model checker can return "unknown" in several cases. First, the amount of time or memory used for the verification was not enough. Second, an incomplete method was used that provides a definitive result in some cases and an indefinite result in other cases. For example, over- or under-approximation techniques are in general incomplete methods [23].
In today's model checkers, the verification problem, as such, is generally re-

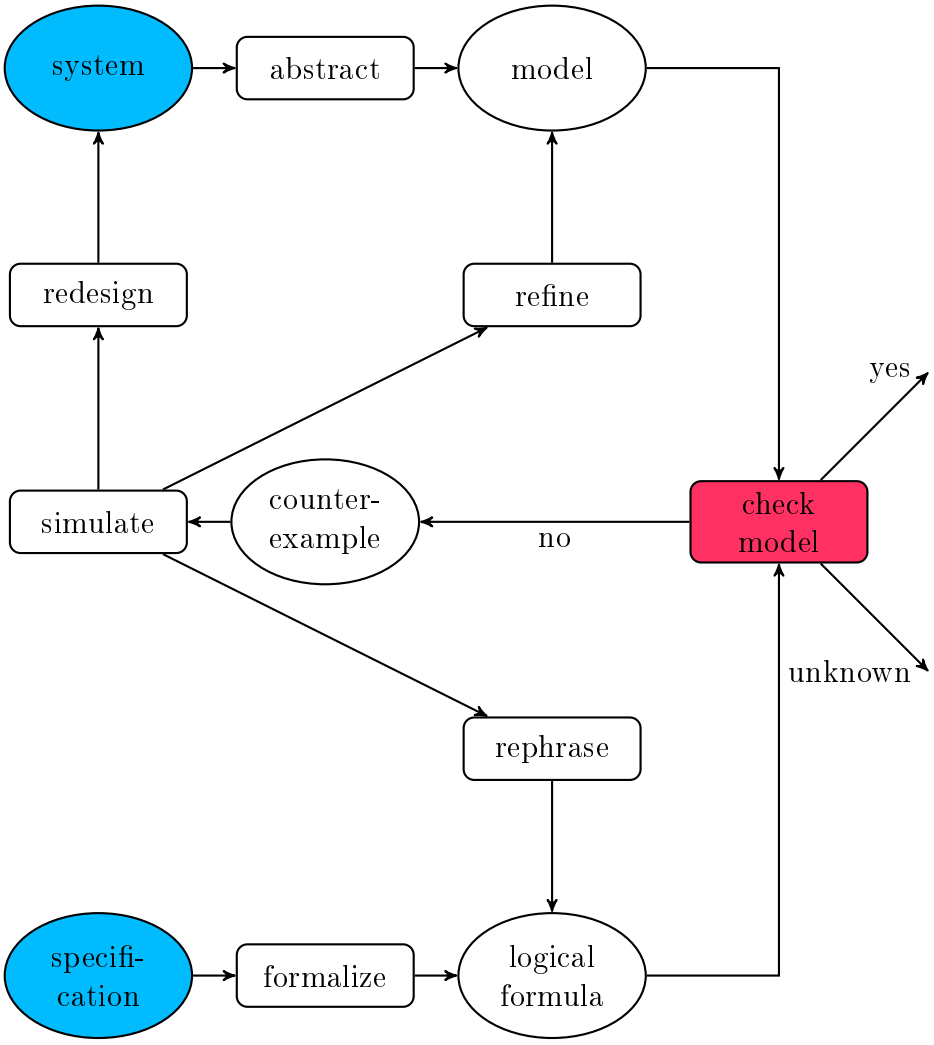


Figure 1.1: Model checking process.

duced to a graph search problem. In this, the nodes of the graph represent the states of the system and the edges represent possible transitions between states. The goal is then to find undesired states, i.e., bugs. The behavior that the state space usually grows exponentially with the size of the model is called the *state explosion problem* [26] and handling it, is a main challenge in model checking today. But even though the number of global states can be enormous, modern techniques have accomplished the verification of large industrial systems. These techniques are among others *compact representation* [12], *symmetry* [110], *abstraction* [85], *model reduction* [4, 5, 8, 92], and *partial order reduction* [49, 97, 127].
Although these techniques greatly reduce the size of the state space, and

thus enable the successful verification of bigger systems, they are simply not powerful enough for the large and complex systems, we often see today. Thus, it is important to come up with new methods or to improve and tune existing ones.
Next to developing new methods on a theoretical level, it is important to implement these methods in a state of the art model checker, too. One such tool is LoLA [142], developed by Karsten Wolf. LoLA is still maintained and further developed by our research group. The successful use in case studies [33, 44, 61, 116, 118], as well as the many victories and podium places at the annual MCC [66–68] show that LoLA is a very powerful tool. In order to solve larger verification problems, it is important to continuously develop and improve LoLA and model checking efficiency in general.

1.2 Research goal

While the scientific findings and their practical applications have made remarkable progress over the years, performance of model checking in general is still insufficient. There are still large industrial systems which are far from approachable by verification. With the globalization and the digitization of the world, systems become more and more complex and harder to verify. This leads us to the central research question that we investigate in this thesis:

> How can model checking efficiency be improved in order to verify larger systems?

This question can be specified to the precise challenges:

1. *Model simplification.* How can a given model be reduced, changed, or extended such that it has a smaller state space?
2. *Specification simplification.* How can a given specification be reduced to an equivalent but simpler one?
3. *State space reduction.* How can a given state space be pruned without leaving out relevant areas?
4. *State space exploration.* How can state space exploration be accelerated?
5. *Alternative reduction techniques.* Which other techniques can be used to speed up model checking?
6. *Performance enhancement of LoLA.* How can the model checking performance of LoLA be increased?

As a research goal of this thesis, we want to investigate the challenges described above. There are many other challenges in this area, however, they will not be the focus of this thesis.

1.3 Contributions

We made several contributions to handle the specific challenges and to reach our research goal. Some of the contributions touch several challenges at once. The results presented here are published in the journal articles [67,82], (submitted [81]) and in the workshop and conference papers [68,76–80]. This thesis summarizes and extends these results by making the following four main contributions.

Contribution 1: reduction techniques

In general, there is a whole range of different approaches available for the verification of systems. To benefit from the wide range of different methods [144], it is worth using a portfolio approach [145]. This means that several methods run in parallel and the fastest one determines the result. As a first contribution, we present two new *alternative reduction techniques* that can be used in a portfolio setting. Furthermore, we present a collection of necessary or sufficient conditions, called quick checks, for a range of specifications.

– **Verification with under-approximation** (published in [80] (and submitted [81])). Scalable models usually scale either over the structure or the resources (players, components, ...) of the model. To verify specifications for resource scaling models, it is sufficient to consider the model with limited resources. As a consequence, the initial state and thus the *model is simplified* and the *state space is reduced*. We propose a sufficient test utilizing an under-approximation [89] to simplify the initial state in the aforementioned way.

– **CEGAR for finite-single-paths formulas** (published in [78] and [82]). Counterexample guided abstraction refinement (CEGAR) [23] is a structural approach that can be used in addition to the actual state space search. This method is especially good for the verification of negative results, meaning that the specification does not hold in the system. CEGAR was only known to work for reachability analysis [140]. This thesis contributes a CEGAR approach for temporal logic formulas to *reduce the state space*.

In particular, in earlier work [78], we proposed a verification technique for computational tree logic (CTL) [20] formula $\mathbf{E}(\varphi \mathbf{U} \psi)$. For this, we use the *Petri net state equation* [91] with CEGAR. As a side product, we showed that $(\mathbf{EX})^k \varphi$ formulas can be solved with CEGAR, too. We use these two approaches as building bricks to solve the entire class of finite-single-path formulas [82]. Such formulas can be verified by a limited linear path through the state space providing either a witness or a counterexample.

– **Quick checks** (published in [79]). For the verification of some complex specifications C, it is enough to verify simpler specifications that serve as a necessary or sufficient condition for the validity of C. We introduce such necessary or sufficient conditions, called quick checks, for a range of specifications. These quick checks serve as *alternative reduction techniques* and are based on structural methods [144] which require less space than the actual verification.

Contribution 2: supplementary strength reduction

This contribution is dedicated to supplementary strength reduction techniques. In particular, we introduce a method to speed up *state space exploration*. Furthermore, we present a collection of techniques to *simplify the specification*. These techniques support the actual verification.

– **Accelerating the state space computation** (published in [77]). Many challenges in model checking reduce to *searching the state space*. Therefore, this is a time critical operation deserving elaborate speed up efforts in every detail. To compute the state space requires for every state to compute its successor. This is usually done by testing every action of the model for its activation in the given state. To speed up this computation it is sufficient to focus the activation test to a limited set of actions. We introduce a new algorithm to compute these information.

– **Simplifying the specification** (published in [79]). *Specification simplification* can drastically decrease the computation power needed for model checking, since a simpler specification is usually easier to verify [7]. We introduce embedded place invariants and traps to find properties of the specification which are invariantly true or false. Such properties can be verified without model checking since they will always stay true or false. If such a property is part of a more complex specification, only the other parts have to be verified using model checking. To further simplify the specification, tautologies can

be used. Many of them are well known in the literature [71], but not all of them are commonly known.

Contribution 3: partial order reduction

This contribution is solely concerned with *state space reduction*. To this end, we use one of the most powerful reduction techniques, namely partial order reduction (POR) [49, 97, 127]. POR is based on the observation that concurrent and independent running processes contribute extensively to the state explosion problem, while having only little influence on the preservation of the specification of individual processes. This thesis extends the reduction efficiency for partial order reduction in the area of temporal logic in the following two aspects.

- **Automata-based POR** (published in [76]). In conventional linear time temporal logic (LTL) [103] model checking [134] with POR the state space is first reduced, and then, together with an automaton B representing the (negated) formula, a product automaton is built. The actual verification takes then place in the product automaton. We use a rather unexplored idea to first build the product automaton with the original state space, and then reduce the product automaton with the additional information available from B. With the additional information available from B, we propose a new Automata-based POR, which is a generalization and extension of the ideas presented in [64] and [75]. With our new method, we are able to weaken or drop several requirements involved in conventional POR and are able to reach a better reduction efficiency.

- **POR for special CTL formulas** (published in [79]). In the CTL [20] category of recent model checking contests [66–68], less problems have been solved than in the reachability and LTL categories. Hence, improving CTL model checking technology deserves particular attention. We propose to relieve a generic explicit CTL model checker [139]. This is done by designing specialized routines that cover a large set of simple and frequently occurring formula types. The generic CTL model checker is then only applied to formulas that do not fall into any special case. A verification technique dedicated to just one class C of simple CTL queries may use a better partial order reduction: we only need to preserve C rather than whole CTL.

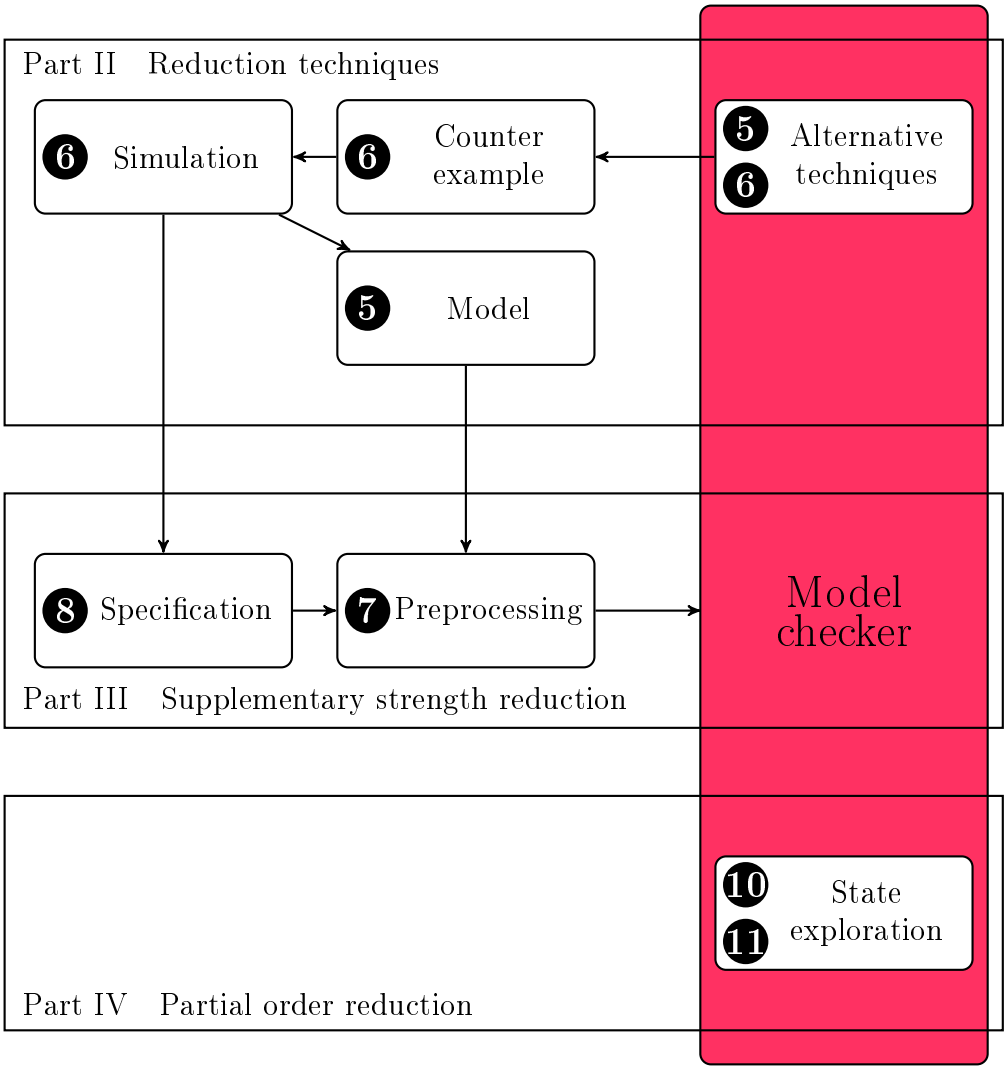


Figure 1.2: Interrelation of the results and its chapters in the context of the model checking procedure.

Contribution 4: Performance enhancement of LoLA

As a proof-of-concept, most algorithms presented in this thesis are implemented in the free open source tool LoLA [142]. Experimental validations using the MCC as benchmark show in all cases a substantial increase in the *model checking efficiency*. The verification with under-approximation applied to resource scaling models is able to solve 12.8 % of all queries. The new algorithm for accelerating the state space computation is two orders of magnitude faster than the old algorithm. Simplifying the specification

could solve 15.1 % of all CTL formulas directly in the initial state. In addition, 1.2 % of the CTL formulas where solved by quick checks. The use of POR in special CTL formulas has resulted in over 50 % of previously unanswered queries being successfully solved. LoLA is available for download at `http://service-technology.org/tools/`.

1.4 Outline

The aforementioned list of *Contributions 1 to 3* sketches an outline for the remainder of this thesis which is illustrated in Figure 1.2. *Contribution 4: Performance enhancement of LoLA* is presented directly in the corresponding chapters after the introduction of the theory.

Part I continues with a review of formal methods and model checking algorithms from the literature. In particular, Chapter 2 and Chapter 3 introduce the formalisms and specifications that we are using for model checking. Chapter 4 presents explicit model checking and describes the algorithms we apply.

Part II is dedicated to Contribution 1: reduction techniques. It introduces reduction techniques for a portfolio approach. In Chapter 5, we present a verification technique for models whose size scales over the existing resources. The technique is based on an under-approximation. Chapter 6 introduces the first extension of a structural CEGAR method to temporal logic. We describe how the Petri net state equation in combination with CEGAR can be used to solve finite-single-path formulas. Furthermore, we introduce quick checks based on structural methods that can be run in parallel to the actual verification.

Part III presents Contribution 2: supplementary strength reduction. Chapter 7 introduces a new technique to speed up the computation of enabledness and disabledness information that we use to build the state space. Chapter 8 introduces a range of specification simplifications.

Part IV is concerned with Contribution 3: partial order reduction. Chapter 9 recalls the partial order reduction from the literature. We describe the involved principles and state which combination of principles preserve which property classes. Chapter 10 introduces a new Automata-based partial order reduction for LTL. In Chapter 11, we present stubborn sets for simple and frequently occurring CTL formulas.

Part V concludes this thesis. Chapter 12 addresses some final selected topics that we believe deserve some remarks. It summarizes this thesis, compares the identified research challenges with the contributions we made and discusses the compatibility and combined use of the introduced techniques. Finally, we identify open problems and future work.

Chapter 2

Systems

A *dynamic process* is a series of activities that interact to produce a result and changes and progresses constantly. A *distributed system* is a cohesive structure of interacting or interrelated processes or components that are at different locations. The components communicate and coordinate their actions by passing messages to one another. Every System is described by its boundaries, defined by its structure and purpose, and expressed through its functioning. Systems and dynamic processes occur everywhere. They are ubiquitous in nature, society, technology, and other areas. Biochemical reactions are examples for dynamic processes in nature. A voting protocol can be seen as a system in the society. And a hardware circuit is a system in the technology field.
To verify such systems and dynamic processes it is not practicable to use them directly since they tend to be very large. Therefore, abstract views, called *formal models*, of these systems and dynamic processes are built. Formal models abstract away uninteresting parts and focus solely on the things that should be verified. As a consequence, formal models are smaller than the original system or dynamic process.
In the thesis, we will be primarily concerned with distributed systems and their behavior over time. A distributed system, compared to a sequential one, involves more than one process. These processes can usually be tracked in terms of quantities that change over time. This brings us to the main abstraction tool in system modeling, which is the concept of state. A state represents all quantities of a system at a certain instant of time. It is basically a snapshot of the system.
Depending on the domain of the quantities of a state, systems can be distinguished. In *discrete* systems all quantities of the system range over countable domains. As an example, consider the arrival times of trains at a station, e.g., 10:12, 10:24, 11:48. Whereas in *continuous* systems all quantities range

over dense domains. An example would be the amount of water that flows over a dam. *Hybrid* systems are comprised of discrete and continuous quantities. States that contain time as a specific variable are *real-time* system. If the timescale of a real-time system is discrete, as well as all other quantities, then the real-time system belongs to the category of discrete systems. On the other hand, if the timescale is dense, as well as all other quantities, then it belongs to the continuous systems. Otherwise, it falls in the category of hybrid systems.
Discrete systems have typically a countable number of states. Whereas continuous and hybrid systems usually have an uncountable number of states. This makes them not suitable for explicit state space verification (states are generated and evaluated one by one), because explicit models depend on the enumeration of states. Since one of our research goals is the performance enhancement of the model checker LoLA and because LoLA is an explicit model checker, we disregard continues and hybrid systems and focus solely upon discrete systems in the remainder.
In this chapter, we work out the area of systems this thesis is concerned with. We introduce basic formalisms for modeling the structure and behavior of a system. More information regarding systems in the field of model checking can be found in [15, 24].
The rest of this chapter is organized as follows. Before we give a formal introduction into systems, we briefly recall some basic mathematical concepts in Section 2.1. We continue in Section 2.2 with the introduction of *labeled transition systems* which are closely related to graphs. The main formalism of this thesis, the *place/transition nets*, are introduced in Section 2.3. Arguing over place/transition nets can be traced back to labeled transition systems.

2.1 Basic mathematical notions

In this section, we recall basic notion of mathematics and computer science and introduce concepts that we use in the remainder of this thesis. As reference for the following definitions see [57, 73].

Sets.
Throughout this thesis, $\mathbb{N}$ is the set of natural numbers (including 0) and $\mathbb{N}_1$ is the set of positive natural numbers (excluding 0). The set of integers is denoted by $\mathbb{Z}$ and the Boolean domain with the elements $\{\mathbf{true}, \mathbf{false}\}$ is denoted by $\mathbb{B}$. In figures and algorithms, we use $\{\mathbf{tt}, \mathbf{ff}\}$ instead of $\{\mathbf{true}, \mathbf{false}\}$. Let M be a finite set. For the cardinality of M, we write $|M|$ to denote the number of elements $x \in M$ that occur in M. The power set of M is denoted by 2^M. It is the set of all subsets of M.

Relations and mappings.
Let X and Y be some sets. A binary relation rel over X and Y is a subset of the Cartesian product $X \times Y$, where the Cartesian product $X \times Y := \{(x, y) | x \in X, y \in Y\}$, with its elements called ordered pairs. The inverse of rel is the binary relation $rel^{-1} = \{(y, x) | (x, y) \in rel\} \subseteq Y \times X$.
A mapping f, also called function, is a right-unique relation over X and Y, that is, with $(x, y), (x, z) \in f \implies y = z$. To distinguish functions f from other relations we write $f : x \to y$.

Multiset.
Let X be some set. Then, a mapping $f : X \to \mathbb{N}_1$ can be interpreted as a *multiset* over X. The number $f(x)$ is the multiplicity of $x \in X$ in f and states, how often an element x occurs in f. We may denote a concrete finite multiset f by enumerating each element x that occurs in f. For example, for a multiset f with $f(x_1) = 2$ and $f(x_2) = 1$, for all $x \in X \setminus \{x_1, x_2\}$ we write $f(x_1, x_1, x_2)$.

In this thesis, we use multisets to describe states of a system.

Alphabet, word, sequence, language.
An *alphabet* W is a finite non-empty set of elements. A finite *word* w over W, also called *sequence*, is a finite string of elements of W. The empty word is denoted by ε and non-empty sequences are written as the concatenation of their elements. We denote the length of w by $|w|$, i.e., if $w = w_1 \dots w_n$, then $|w| = n$. Further, we denote with

- W^n the set of all words with length n over W;
- $W^* := \cup_{i \in \mathbb{N}} W^i$ the set of all words over W;
- $W^+ := W^* \setminus \{\varepsilon\}$ the set of all non-empty words over W.

A *language* over W is a subset L of W^*.

Many challenges in the model checking domain can be traced back to graphs and graph algorithms. Therefore, we recall also basic graph notions from the literature in this section. As reference for the following definition see [10].

Graph, subgraph.
A directed graph $G = (V, E)$ consists of a finite set V of *vertices (or nodes)* and a set $E \subseteq V \times V$ of directed *edges (or arcs)*. A graph $G' = (V', E')$ is a *subgraph* of G, written as $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$.

In this thesis, we are only concerned with directed graphs and therefore, we skip undirected graphs.

2.2 Labeled transition system

In Chapter 1, we introduced the concept of state. That is, a state represents a snapshot of the system at a particular instant of time. To describe the dynamics of a system requires the next building brick, the concept of a state change. State changes are assumed to occur instantaneously at discrete points of time. Therefore, it possible to abstract away the explicit notion of continuous time with gradual change.
State changes in a system occur due to *events* or *actions*. The mathematical association of original state, occurred event, and new state are captured by a *transition relation*. The evolution of a system is then defined as a finite or infinite sequence of states, where each state is obtained from the previous state by an event as determined in the transition relation. The most general notion to describe discrete systems, using the concepts of state and event, is a labeled transition system. As reference for the following definitions see [50].

Definition 2.2.1 (Labeled transition system (LTS)).
A labeled transition system $TS = (S, A, \rightarrow, s_0)$ *consists of:*

- *a finite set* S *of* states*;*
- *a finite set* A *of* actions*;*
- *a relation* $\rightarrow \subseteq S \times A \times S$ *of* transitions*, also called* events*;*
- *an* initial state $s_0 \in S$.

An example for a labeled transition system with $TS = (S, A, \rightarrow, s_0)$ can be seen in Figure 2.1. TS consists of

- $S = \{s_0, s_1, s_2\}$;
- $A = \{a, b, c, d\}$;
- $\rightarrow = \{(s_0, a, s_1), (s_1, b, s_0), (s_1, c, s_2), (s_2, d, s_0)\}$;
- initial state s_0.

An LTS models the behavior of systems. It also represents a class of automata with an initial state but no end state. The transition relation $\rightarrow$ denotes possible state changes. If (s, a, s') is an element of the transition relation, then the system can move from state s to s' by performing action a. In the remainder of the thesis, we denote $(s, a, s') \in \rightarrow$ as $s \xrightarrow{a} s'$. If $s \xrightarrow{a} s'$ for some action a, then s' is an immediate *successor* of s, and s is an immediate *predecessor* of s'. In an *unlabeled transition system* the transition relation is defined without actions, $\rightarrow \subseteq S \times S$.

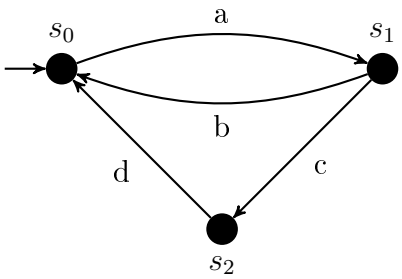


Figure 2.1: An example labeled transition system.

Definition 2.2.2 (Terminal state).
Let $TS = (S, A, \rightarrow, s_0)$ be an LTS. A state $s \in S$ is a terminal *state if and only if there is no $s' \in S$ such that $s \xrightarrow{a} s'$ with $a \in A$.*

Terminal states, thus, have no successor states. In Chapter 6, we use terminal states for the verification of some specifications. We continue with the formalization of the behavior of an LTS.

Definition 2.2.3 (Path).
Let $TS = (S, A, \rightarrow, s_0)$ be an LTS and $s, s' \in S$ two states. A finite path *of length n from s to s' is sequence of n actions $s_1 \xrightarrow{a_1} s'_1 s_2 \xrightarrow{a_2} s'_2 \ldots s_n \xrightarrow{a_n} s'_n$ such that $s = s_1, s' = s'_n$ and $s'_i = s_{i+1}$ for $i = 1, \ldots, n$, denoted by $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots s_n \xrightarrow{a_n} s_{n+1}$. If $n = 0$, then the path is* empty *and $s = s' = s_1$. An* infinite path *is a sequence $s_1, s_2, s_3, \ldots$ such that $s_i \xrightarrow{a_1} s_{i+1}$ for each $i \in \mathbb{N}$, yields the infinite path $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \ldots$.*

In some cases, we require that a finite path ending in a terminal state s is infinite. In such a cases, s will be repeated indefinitely with $s \xrightarrow{\tau} s$ where τ is special action denoting an invisible internal action.

Definition 2.2.4 (Maximal path).
A maximal path is either a finite path that ends in a terminal state, or an infinite path.

Paths(s) denotes the set of maximal paths.

Definition 2.2.5 (Acyclic, cyclic).
Let $TS = (S, A, \rightarrow, s_0)$ be an LTS. A path $\pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots s_n \xrightarrow{a_n} s_{n+1}$ in TS is acyclic *if $s_i \neq s_j$ for all $i \neq j$ with $i, j \in \{1, \ldots, n+1\}$. Otherwise π is cyclic. If TS contains no cyclic path starting from s_0, then TS is acyclic.*

Using paths, we are able define the notion of *reachable*.

Definition 2.2.6 (Reachable).
Let $TS = (S, A, \rightarrow, s_0)$ be an LTS, ε the empty sequence of actions, w a sequence of actions, and a an action. Further, let A^* be the set of all possible strings using A, including the empty set. We define the reachability relation *$\rightarrow^* \subseteq S \times A^* \times S$ by the inductive scheme using the following axiom and rule:*

$$\frac{}{s \xrightarrow{\varepsilon}{}^* s} \qquad \frac{s_1 \xrightarrow{w}{}^* s_2 \quad s_2 \xrightarrow{a}{}^* s_3}{s_1 \xrightarrow{wa}{}^* s_3}$$

We say that s' is reachable *from s denoted by $s \rightarrow^* s'$ if and only if there is a sequence w of actions holding $s \xrightarrow{w}{}^* s'$. For better readability, we often write $s \xrightarrow{w} s'$ instead of $s \xrightarrow{w}{}^* s'$.*

For the verification of some specifications, e.g., liveness [109], certain structures can be used.

Definition 2.2.7 (SCC, TSCC).
Let $TS = (S, A, \rightarrow, s_0)$ be an LTS. A strongly connected component (SCC) *is a maximal set of states $\mathcal{M}$ of TS such that $s, s' \in \mathcal{M}$ implies $s \rightarrow^* s'$. A* terminal strongly connected component (TSCC) *is an SCC from which no other SCC is reachable.*

The example LTS from Figure 2.1 has only one TSCC which consists of all states. Every state is reachable from any other state.
We are also able to analyze the behavior and the structure of labeled transition systems using graph terms.

Lemma 2.2.1 (LTS as graph)**.** *For every LTS there is a directed graph where nodes are interpreted as states, and edges are interpreted as state changes triggered by actions in the edge labels.*

The graph terms from Section 2.1 can be, thus, lifted to labeled transition systems. In the remainder of this thesis, we will use this to apply graph algorithms on labeled transition systems.

2.3 Place/transition nets

The main formalism used in this thesis for modeling the structure and the behavior of a system are *place/transition nets*. Place/transition nets are the same as classical (low-level) *Petri nets* [101], and also known as vector addition systems [63]. Place/transition nets are an established formalism for modeling, analyzing, and verifying distributed systems and their features. The global state of a distributed system is described in a place/transition

net as a collection of local states. Examples for such systems are hardware circuits, resources, protocols, control flow, and many more. As reference for an introduction of Petri nets the reader is referred to [100, 106].

Definition 2.3.1 (Place/transition net).
A place/transition net (P/T net) $N = (P, T, F, W, m_0)$ *consists of*

- *a finite set of* places P;
- *a finite set of* transitions T *such that* $P \cap T = \emptyset$;
- *a set of* arcs $F \subseteq (P \times T) \cup (T \times P)$;
- *a* weight function $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ *such that* $W(x, y) = 0$ *if and only if* $(x, y) \notin F$;
- *an* initial marking $m_0 : P \to \mathbb{N}$.

A P/T net can be seen as a (finite) bipartite directed graph with the node set unifying *places* and *transitions*. Weights W and markings such as m_0 reflect to the dynamic of P/T nets which is to be introduced later. The graphically representation of a place is a circle and the one of a transition is a box. Weighted arcs are graphically represented as weight-labeled edges. If an arc weight is one, then it is usually omitted in the graphical representation.
The structural environment of a place or transition is covered by the notion of *preset* and *post-set* of a node.

Definition 2.3.2 (Preset, post-set).
For a node $x \in P \cup T$ *of a P/T net* N, ${}^\bullet x = \{y \mid (y, x) \in F\}$ *is the* preset *of* x *and* $x^\bullet = \{y \mid (x, y) \in F\}$ *is the* post-set *of* x. *This extends to sets of nodes: if* $X \subseteq P \cup T$, *then* ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ *and* $X^\bullet = \bigcup_{x \in X} x^\bullet$ *are the pre- respectively post-set of* X.

As an example for presets and post-sets consider Figure 2.2. The transition t_2 has the preset ${}^\bullet t_2 = \{p_1, p_2, p_3\}$ and the post-set $t_2{}^\bullet = \{p_4, p_5\}$. The post-set of the set $\{p_1, p_2\}$ is $\{p_1, p_2\}^\bullet = p_1{}^\bullet \cup p_2{}^\bullet = \{t_1, t_2\}$.
Another structural environment notion are conflict transitions. Intuitively, transitions may be in conflict if they share a place in their presets. In [38] Desel and Esparza extended this observation toward a decomposition of a P/T net into its *conflict clusters*.

Definition 2.3.3 (Conflict cluster).
Let $x \in P \cup T$ *be a node of a P/T net* $N = (P, T, F, W, m_0)$. *The* conflict cluster *of* x, *denoted* $[x]$, *is the minimal set of nodes such that:*

- $x \in [x]$;

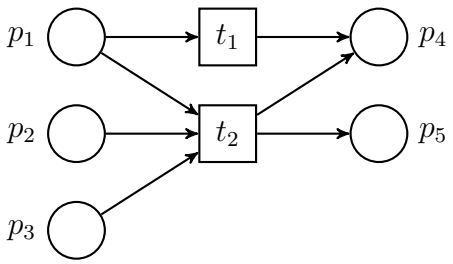


Figure 2.2: Presets and post-sets.

- *If* $p \in P$ *and* $p \in [x]$, *then* $p^\bullet \subseteq [x]$;
- *If* $t \in T$ *and* $t \in [x]$, *then* ${}^\bullet t \subseteq [x]$.

So far, a P/T net N represents only a static structure. The dynamic behavior is introduced by a notion of states and state changes. States of P/T nets are *markings* $m : P \to \mathbb{N}$. A marking of N can be viewed as a distribution of resources over (a subset of) the places of N. The resources are called *tokens*.

Definition 2.3.4 (Marking).
A marking m *of a P/T net* $N = (P, T, F, W, m_0)$ *is a mapping* $m : P \to \mathbb{N}$.

In a given state of a P/T net N, that is, for a marking m, each place $p \in P$ of N, is assigned its number $m(p)$ of tokens. Graphically, m is depicted by putting $m(p)$ black dots on every place p. Tokens are used in P/T nets to simulate the dynamic and concurrent activities of systems.

Definition 2.3.5 (Vector notation for markings).
Let $N = (P, T, F, W, m_0)$ *be a P/T net and* m *be a marking. We call* $m = (m(p_1), m(p_2), \ldots, m(p_n))$ *with* $p \in P, n = |P|$ *the* vector notation *for* m.

In this thesis, we use the traditional vector notation or the earlier introduced multiset notation to describe markings. For example consider a P/T net with 3 places and a marking m where the first place has 1 token, the second place has 2 tokens, and the last place is empty. The traditional vector notation would then be $m = (1, 2, 0)$ and the multiset notation $m = (p_1, p_2, p_2)$.
An example P/T net $N = (P, T, F, W, m_0)$ with a non-empty initial marking, can be seen in Figure 2.3. N consists of:

- $P = \{p_1, p_2, p_3, p_4\}$;
- $T = \{t_1, t_2, t_3, t_4, t_5\}$;
- $F = \{(p_1, t_1), (p_2, t_2), (p_2, t_3), (p_3, t_4), (p_4, t_5),$
  $(t_1, p_2), (t_2, p_1), (t_3, p_3), (t_4, p_4), (t_5, p_3)\}$;

- $W = \{((p_1, t_1), 1), ((p_2, t_2), 1), ((p_2, t_3), 1), ((p_3, t_4), 1), ((p_4, t_5), 1);$
  $((t_1, p_2), 1), ((t_2, p_1), 1), ((t_3, p_3), 1), ((t_4, p_4), 1), ((t_5, p_3), 1)\}$;
- $m_0 = (1, 0, 0, 0)$.

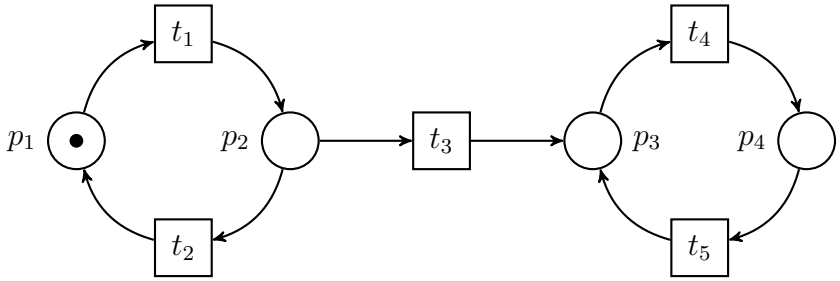


Figure 2.3: A P/T net with initial marking $m_0 = (1, 0, 0, 0)$.

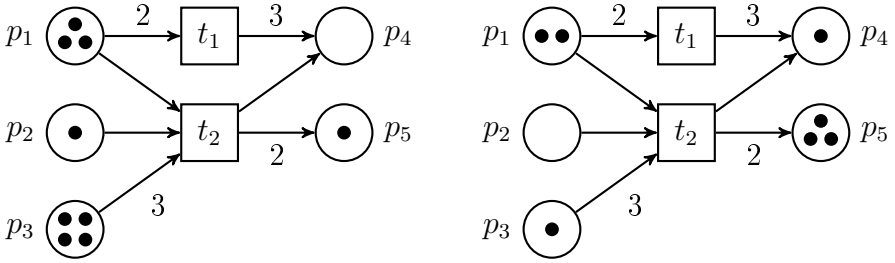
With the marking notion defined, we are now able to carry on with the introduction of state changes. To this end, we need to know which transitions are enabled.

Definition 2.3.6 (Enabledness).
Let $N = (P, T, F, W, m_0)$ be a P/T net. A transition $t \in T$ is enabled *in marking m if, for all $p \in {}^\bullet t$ it holds that $W(p, t) \leq m(p)$.*

In P/T nets, enabled transitions can trigger the events that cause a state change. In fact, once a transition t is enabled in marking m, it can fire and by doing so, consume $W(p, t)$ tokens from each place $p \in {}^\bullet t$ and produce $W(t, p')$ tokens on each place $p' \in t^\bullet$. This results in a new marking which is the successor of m. We formalize this behavior of a P/T net in the following *transition rule*.

Definition 2.3.7 (Transition rule of a P/T net).
Let $N = (P, T, F, W, m_0)$ be a P/T net and $t \in T$ a transition of N. If t is enabled in marking m, t can fire*, producing a new marking m' where, for all $p \in P$, $m'(p) = m(p) - W(p, t) + W(t, p)$. Then, m' is the successor marking of m with respect to the firing of t. The transition rule defines a relation $\rightarrow$ on the markings where $m \xrightarrow{t} m'$ denotes that firing t changes marking m into m'. We also say that m' is* reachable *via transition t from m.*

A transition would be disabled in markings where not all tokens that will be consumed are available in the respective places. The absence of tokens is the only way to disable transitions. In the example P/T net in Figure 2.4a, it is easy to see that both transitions t_1 and t_2 are enabled in



(a) Marking $m_0 = (3, 1, 4, 0, 1)$. (b) Marking $m_1 = (2, 0, 1, 1, 3)$.

Figure 2.4: The firing of t_2 in m_0 yields the successor marking m_1.

marking $m_0 = (3, 1, 4, 0, 1)$. The firing of t_2 in m_0 yields the successor marking $m_1 = (2, 0, 1, 1, 3)$, as depicted in Figure 2.4b. After firing t_2 only t_1 remains enabled in m_1. The transition t_2 is disabled because one token is missing on p_2 and three tokens are missing on p_3.
The transition rule can be extended to transition sequences.

Definition 2.3.8 (Transition sequence (reachability)).
Let $N = (P, T, F, W, m_0)$ be a P/T net and m, m' two markings. A transition sequence *is defined by the following inductive scheme: $m \xrightarrow{\varepsilon} m$, for the empty sequence ε, and $m \xrightarrow{wt} m'$ for a sequence $w \in T^*$ and a transition $t \in T$ if and only if there is a marking m_1 such that $m \xrightarrow{w} m_1$ and $m_1 \xrightarrow{t} m'$. Marking m' is* reachable *from marking m if and only if there is a transition sequence $w \in T^*$ such that $m \xrightarrow{w} m'$.*

Using reachability, a P/T net induces the *reachability graph*, also called the *state space* of a P/T net.

Definition 2.3.9 (Reachability graph).
Let $N = (P, T, F, W, m_0)$ be a P/T net. The reachability graph *$R_N = (M, E)$ of N has a set of vertices M that comprises all markings that are reachable by any sequence from the initial marking of N. Every element $m \xrightarrow{t} m'$ of the firing relation ($t \in T$) defines an edge E from m to m' annotated with t.*

The introduced notation for reachability is compatible with the corresponding notion for labeled transition systems. The state space of a P/T net can also be expressed with an LTS.

Definition 2.3.10 (P/T net state space).
Let $N = (P, T, F, W, m_0)$ be a P/T net and $TS = (S, A, \rightarrow, s_0)$ an LTS. Further, let $w \in T^*$ be a sequence and $t \in T$ a transition. TS is called the state space of N if and only if

- $S = \{m | m_0 \xrightarrow{w} m\}$;
- $A = \{T\}$;
- $(m, t, m') \in \rightarrow$ *if and only if* $m \xrightarrow{t} m'$;
- $s_0 = m_0$.

Figure 2.5 shows the reachability graph (state space) of Figure 2.3 using the multiset notion. The edges of the graph correspond to the transitions, and the paths correspond to the transition sequences. For example, the marking (p_2) has two outgoing edges, since in (p_2) the two transitions t_2 and t_3 are enabled. In addition to this, the figure shows the SCCs of the reachability graph. There are two SCCs. The first one is $C_1 = \{(p_1), (p_2)\}$ and the second one is $C_2 = \{(p_3), (p_4)\}$, which is also a terminal SCC.

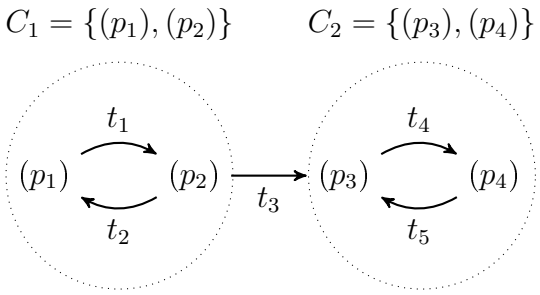


Figure 2.5: The reachability graph of Figure 2.3 with its (T)SCCs.

Since one of our research goals is *state space reduction*, we need to define the size of a state space in order to compare it.

Definition 2.3.11 (State space size).
The size $|R_N|$ *of the reachability graph* $R_N = (M, E)$ *of a P/T net* N *with initial marking* m_0 *is defined as the number of markings reachable from* m_0 *and the number of edges,* $|M| + |E|$. *Given another reachability graph* $R_{N'} = (M', E')$ *of P/T net* N', *we say that* $|R_N| \leq |R_{N'}|$ *(*R_N *is smaller than* $R_{N'}$*), if* $|M| + |E| \leq |M'| + |E'|$.

In the remainder of this thesis, we shall use P/T nets as basic formalism. The decision to focus on P/T nets is due to two reasons. First, as seen in this section, P/T nets have a strong mathematical foundation enabling a rigorous analyses and verification process. Second, based on our research goal *to increase the model checking efficiency of LoLA* we focus on P/T nets, because LoLA works on P/T nets. Further, we use the induced reachability graph of a P/T net as labeled transition system to analyze the behavior respectively to work with specifications, in the sequel.

The size of the reachability graph is usually several times larger than the underlying P/T net. For example, consider a process with n states and a system that is composed of m such processes. If the system is executed asynchronously, then the system has n^m states. "As the number of state variables in the system increases, the size of the system state space grows exponentially. This is called the state explosion problem" [26]. Thus, reachability is the fundamental verification problem in model checking [26, 130].

Definition 2.3.12 (Reachability problem).
Given is a tuple (N, m, m') consisting of a P/T net N and two markings m and m'. The question whether $m' \in R_N(m)$ is called the reachability problem.

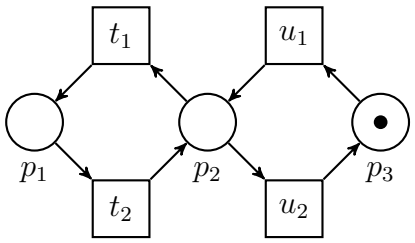
Lipton has shown that the problem is EXPSPACE-hard [83], which means it is intractable to solve. It is well known that a necessary condition for a positive answer to a reachability problem is the feasibility of the state equation [91].

Definition 2.3.13 (State equation).
The incidence matrix *of a P/T net N is a matrix $C_N : P \times T \longrightarrow \mathbb{Z}$ where, for all $p \in P, t \in T$, $C_N(p,t) = W(t,p) - W(p,t)$. Let $w \in T^*$ be a firing sequence of N, that is, the sequence of labels on a path from some marking m to a marking m' in the labeled transition system corresponding to N. Then the system of linear equations*

$$m + C_N \cdot \wp(w) = m'$$

is the Petri net state equation. *The vector $\wp(w)$ fulfilling the equation is called a solution or* Parikh vector *of w. We denote with $|\wp(w)(t)|$ the number of occurrences of t in the sequence w of the Parikh vector.*

Instead of Petri net state equation, we sometimes only say state equation or P/T state equation. If it is clear form the context, we refer to the incidence matrix of a P/T net N by C.
The state equation can be used in several situations. The first one is to compute a final marking. As an example consider Figure 2.6, it shows on the left side a P/T net and on the right side the corresponding incidence matrix C_N. Further, let $\wp(w) = (1, 0, 2, 1)$ be a Parikh vector and $m = (0, 0, 1)$ be the initial marking. Then we can apply the state equation to compute the final marking $m' = (1, 0, 0)$.
For the next situation, we first have to introduce the notion of *executable*.



$$\begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & u_1 & u_2 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \end{array}$$

Figure 2.6: State equation example. The left figure shows a P/T net and the right side shows the corresponding incidence matrix.

Definition 2.3.14 (Executable).
Let $N = (P, T, F, W, m_0)$ be a P/T net, m' a target (final) marking, and $\wp(w)$ a solution vector. If there exist a transition sequence w from m_0 to m' that contains exactly those transitions in $\wp(w)$, then w is executable in N.

A solution $\wp(w)$ of the state equation can be spurious. That is, $\wp(w)$ is not necessarily executable in the P/T net N, i.e., there is no transition sequence that can be executed in N. The reason for this is that the state equation is a linear algebraic over-approximation of the set of reachable states. However, the state equation can be used to compute a solution to a final marking. If the state equation provides no solution, then the final marking is not reachable. On the other hand, if it provides a solution that is executable, then the final marking is reachable. We will formalize and use this behavior in Chapter 6 to solve some specifications. With the state equation, we are also able to define two types of invariants: *transition invariants* (*T-invariant*), and *place invariants* (*P-invariant*) [74].

Definition 2.3.15 (T-invariant).
A Parikh vector $\wp(w)$ is a transition invariant, *also called* T-invariant, *if $C \cdot \wp(w) = 0$ holds. If the firing sequence w is executable, we call $\wp(w)$* realizable.

A realizable T-invariant is a cycle in the state space and will not change the marking.

Definition 2.3.16 (P-invariant).
A vector i is a place invariant, *also called* P-invariant, *if $i \cdot C = 0$ holds.*

Instead of $i \cdot C = 0$, we sometimes use $C^T i = 0$. A place invariant has the property that the *weighted sum of tokens* stays the same in each marking m.

Lemma 2.3.1 (Weighted sum of tokens [74]). *Let $N = (P, T, F, W, m)$ be a P/T net. If i is a place invariant, then the equation $i(p_1)m(p_1) + \cdots +$*

$i(p_n)m(p_n) = im_0$ *holds for all reachable markings in* N *with the set of places* $\{p_1, \ldots, p_n\}$.

Another property of P/T nets is *boundedness*.

Definition 2.3.17 (Bounded P/T net).
A P/T net $N = (P, T, F, W, m_0)$ *is* bounded *if it has only a finite number of reachable markings, i.e., if the set* $R_N(m_0)$ *is finite. Otherwise, it is* unbounded.

In this thesis, we restrict our analysis to bounded P/T nets. The reason for this is that bounded P/T nets have a finite state space. Verification algorithms for unbounded P/T nets need other algorithms such as the coverability graph [45] to deal with the infinite state space. Finally, we will introduce the notion of *deadlocks*, which is compatible with the notion of terminal state for LTS.

Definition 2.3.18 (Deadlock).
Let $N = (P, T, F, W, m_0)$ *be a P/T net.* N *has a* deadlock *if there exist a reachable marking from* m_0 *in which no transition is enabled.*

Chapter 3

Specification

In the previous chapter, we introduced and formalized systems. A system is one of two inputs for the model checker. The other input is the specification, which we are going to describe and formalize in this chapter. In order to analyze and verify a given model, we need a precise and unambiguous framework to state meaningful properties that a model must satisfy. With the knowledge of the meaningful properties, the formal model can be built capturing those properties. Details that are not meaningful should be abstracted away to keep the model and therefore the state space as small as possible. The actual verification then uses the formal model and the specified properties to reason about whether the model meets the specification.
Specifications are usually given in some logical formalism such as *temporal logic*. Temporal logic describes the ordering of events in time without introducing time explicitly. Basically, it extends traditional propositional logic with operators that refer to the behavior of systems over time. For example, a specification can ensure that some concurrent system has no deadlocks. Other examples are safety (something that never happens) and liveness (something good will eventually happen) properties. Further information about specifications for model checking can be found in [15, 24].
The rest of this chapter is organized as follows. In the next section, we introduce some basic definitions. We continue in Section 3.2 with the presentation of CTL*. Section 3.3 and 3.4 are concerned with the introduction of LTL and CTL, respectively. Section 3.5 presents the logic classes ACTL* and ECTL*.

3.1 Basics

The core elements of every specification are atomic propositions.

Definition 3.1.1 (Atomic proposition (in general)).
Let $TS = (S, A, \rightarrow, s_0)$ be an LTS. An atomic proposition $\alpha : S \rightarrow \mathbb{B}$ assigns truth values to states of TS. The fact that a state s satisfies atomic proposition α, that is $\alpha(s) =$ `TRUE`, is denoted by $s \models \alpha$.

Basically, we have $\alpha_1, \ldots, \alpha_n$, $n \in \mathbb{N}$ atomic propositions and a labeled transition system whose states are annotated with the values of $\alpha_1, \ldots, \alpha_n$. Since we mainly use P/T nets in this thesis, we introduce a definition of P/T net specific atomic propositions. To this end, we introduce four atomic proposition constants. In the sequel, let $N = (P, T, W, F, m_0)$ be a P/T net.

- `TRUE`: evaluates always to true;
- `FALSE`: evaluates always to false;
- `FIREABLE(`t`)` (for $t \in T$): evaluates to true, if t is enabled in a marking $m \in R_N$;
- `DEADLOCK`: evaluates to true, if N has a deadlock.

Definition 3.1.2 (Atomic proposition for P/T nets).
Let $N = (P, T, F, W, m_0)$ be a P/T net. An atomic proposition is one of the constants `TRUE`, `FALSE`, `FIREABLE(`t`)` *(for $t \in T$),* `DEADLOCK`*, or an expression of the shape $k_1 p_1 + \cdots + k_n p_n \leq k$, for some $n \in \mathbb{N}$, $k_1, \ldots, k_n, k \in \mathbb{Z}$, and $p_1, \ldots, p_n \in P$. For a marking m of N, m satisfies proposition $k_1 p_1 + \cdots + k_n p_n \leq k$ if and only if the term $\sum_{i=1}^{n} k_i m(p_i)$ actually evaluates to a number less or equal to k. The fact that a marking m satisfies atomic proposition α is again denoted by $m \models \alpha$.*

Intuitively, atomic propositions represent simple known facts about states. Therefore we assume that atomic propositions can be evaluated with negligible resources. Besides the atomic propositions, the logic consists of Boolean connectors such as negation, disjunction and conjunction ($\neg, \vee, \wedge$). These elements are used to build more complicated expressions to describe properties of the system.
Specifications are used to describe transition sequences and state changes. One formalism for describing transition sequences in a system is *temporal logic*. Temporal logic extends traditional propositional logic with operators that refer to the behavior of systems over time. It allows to specify properties that describe a before-after-relation between states along a path. Before we introduce a model for expressing properties that involve more than one path, we recall the traditional definition of a *directed labeled rooted tree*.

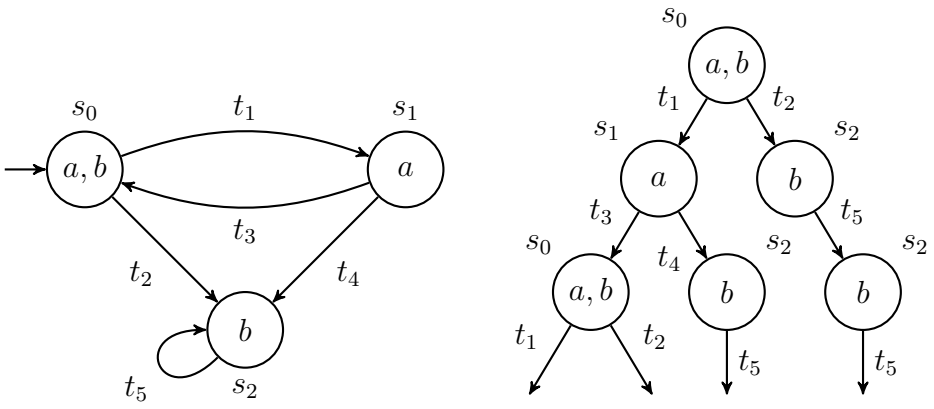
Definition 3.1.3 (Directed labeled rooted tree).
A directed labeled rooted tree is a connected and acyclic directed graph $G = (V, E, v_0)$ in which v_0 is the root and each vertex has a label.

We use the directed labeled rooted tree to define the *computation tree.*

Definition 3.1.4 (Computation tree).
Let $TS = (S, A, \rightarrow, s_0)$ be an LTS. A computation tree *$G = (V, E, v_0)$ for TS is a directed labeled rooted tree if and only if:*

- *every Vertex $v \in V$ is labeled with a state $s \in S$;*
- *the root vertex v_0 is labeled with s_0;*
- *every edge $e \in E$ is labeled with an action $a \in A$;*
- *an edge e connects two vertices v, v' only if the corresponding labels form a transition relation in $\rightarrow$;*
- *for every vertex v labeled with s, and every transition relation (s, a, s'), there is an edge labeled with a connecting v with a vertex labeled s'.*

The computation tree shows all possible paths starting from the initial state of an LTS. For example, a property might specify that a faulty state is never reached, or that something good will eventually happen. If the LTS contains no cycles, then the size of the computation tree is finite, otherwise it is infinite, because a cycle is unwound over and over again. The six top states of the computation tree of the LTS from Figure 3.1a are shown in Figure 3.1b.



(a) *LTS.* (b) Computation tree.

Figure 3.1: LTS and the Computation tree of the LTS.

3.2 Computational tree logic*

We start with the introduction of the syntax and semantics of the computational tree logic* (CTL*) defined by Emerson and Halpern [40]. CTL* allows

us to easily define the temporal logics we want to use, because CTL* is a superset of all of them. The syntax of CTL* is described by *state formulas* and *path formulas*. The idea is that state formulas being true in a specific state, and path formulas being true along a specific path.

Definition 3.2.1 (Syntax of CTL*).
For a given set of atomic proposition AP, the temporal logic CTL* is inductively defined as follows:

- *Every $\varphi \in AP$ is a state formula;*
- *If φ and ψ are state formulas, then $\neg\varphi$, $(\varphi \wedge \psi)$, and $(\varphi \vee \psi)$ are state formulas;*
- *Every state formula is a path formula;*
- *If φ and ψ are path formulas, then $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\mathbf{X}\,\varphi$, $\mathbf{F}\,\varphi$, $\mathbf{G}\,\varphi$, $(\varphi\,\mathbf{U}\,\psi)$, and $(\varphi\,\mathbf{R}\,\psi)$ are path formulas;*
- *If φ is a path formula, then $\mathbf{A}\,\varphi$ and $\mathbf{E}\,\varphi$ are state formulas.*

The set of all state formulas generated by these rules form the class of CTL* formulas. For a state s of a labeled transition system, $L(s)$ registers the set of atomic propositions that are valid in s. The semantics of CTL* is defined with respect to some labeled transition system.

Definition 3.2.2 (Semantics of CTL*).
Let TS be a labeled transition system and α an atomic proposition. Let φ_1 and φ_2 be state formulas and ψ_1 and ψ_2 path formulas. State s satisfies CTL* state formula φ_1 denoted as $s \models \varphi_1$ and path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ satisfies CTL* path formula π denoted as $\pi \models \psi_1$ if and only if

State formulas

1. $s \models \alpha \iff \alpha \in L(s)$;
2. $s \models \neg\varphi_1 \iff s \not\models \varphi_1$;
3. $s \models \varphi_1 \wedge \varphi_2 \iff s \models \varphi_1$ *and* $s \models \varphi_2$;
4. $s \models \varphi_1 \vee \varphi_2 \iff s \models \varphi_1$ *or* $s \models \varphi_2$;
5. $s \models \mathbf{A}\,\psi_1 \iff \pi \models \varphi_1$ *for all* $\pi \in$ Paths(s) *starting in s;*
6. $s \models \mathbf{E}\,\psi_1 \iff \pi \models \varphi_1$ *for some path* $\pi \in$ Paths(s) *starting in s.*

Path formulas
Let π^i be the suffix of π starting at s_i.

7. $\pi \models \varphi_1 \iff s_0 \models \varphi_1$;
8. $\pi \models \neg\psi_1 \iff \pi \not\models \psi_1$;
9. $\pi \models \psi_1 \wedge \psi_2 \iff \pi \models \psi_1$ *and* $\pi \models \psi_2$;
10. $\pi \models \psi_1 \vee \psi_2 \iff \pi \models \psi_1$ *or* $\pi \models \psi_2$;
11. $\pi \models \mathbf{X}\,\psi_1 \iff \pi^1 \models \psi_1$;
12. $\pi \models \mathbf{F}\,\psi_1 \iff$ *there exists a* $k \geq 0$ *such that* $\pi^k \models \psi_1$;
13. $\pi \models \mathbf{G}\,\psi_1 \iff$ *for all* $i \geq 0$, $\pi^i \models \psi_1$;
14. $\pi \models \psi_1\,\mathbf{U}\,\psi_2 \iff$ *there exists a* $k \geq 0$ *such that* $\pi^k \models \psi_2$ *and for all* $0 \leq j < k$, $\pi^j \models \psi_1$;
15. $\pi \models \psi_1\,\mathbf{R}\,\psi_2 \iff$ *for all* $j \geq 0$, *if for all* $i < j$, $\pi^i \not\models \psi_1$, *then* $\pi^j \models \psi_2$.

An LTS TS satisfies a state formula if its initial state does and it satisfies a path formula if all paths starting in the initial state do.

We denote the fact that if a state s of TS satisfies a state formula φ with $(TS, s) \models \varphi$. Similarly, if a path π of TS satisfies a path formula φ we write $(TS, \pi) \models \varphi$. If it is clear which LTS we are using, we only write $s \models \varphi$ respectively $\pi \models \varphi$.
For a better understanding, we also give an informal description for the quantifiers and operators. Path quantifiers are used to describe the branching structure.

- **A** requires that *all* paths have the specified property.
- **E** requires that there *exists* at least one path with the specified property.

We call **A** the *all* or *universal* path quantifier and **E** the *existential* path quantifier. A state s satisfies $\mathbf{A}\,\varphi$, if all paths starting in s satisfy φ. By analogy, a state s satisfies $\mathbf{E}\,\varphi$, if there exists at least one path starting in s that satisfy φ. Path quantifiers are concerned with states, whereas temporal operators are concerned with properties along a path in the computation tree. The three temporal operators $\mathbf{X}, \mathbf{F}, \mathbf{G}$ are concerned with a single property:

- **X** requires that a property holds in the *next* state of the path.
- **F** requires that a property holds *eventually* or *in the future* in some state of the path.
- **G** requires that a property holds in each state (*globally* or *always*) of the path.

The temporal operators **R** and **U** combine two properties:

– **U** requires that there is a state s on the path in which the second property holds, and, in addition, up *until* s the first property must hold in every previous state on the path.
– **R** requires that the second property holds at every state along the path up until and including the first state s in which the first property holds. If such an s does not exist, then the second property holds forever. The first property *releases* the second property.

There is another aspect to consider in the way we have defined the semantics of CTL*. We defined all operators explicitly for a better readability. However, the Boolean connector $\vee$, the path quantifier **A**, and the temporal operators **F**, **G**, and **R** can also be defined using the following tautologies.

Lemma 3.2.1 (Minimal set of operators [24]). *The operators* $\vee, \neg, \mathbf{X}, \mathbf{E}, \mathbf{U}$ *are sufficient to express any other CTL* formula.*

– $(\varphi \vee \psi) \iff \neg(\neg\varphi \wedge \neg\psi)$;
– $\mathbf{A}\,\varphi \iff \neg\mathbf{E}\,(\neg\varphi)$;
– $\mathbf{F}\,\varphi \iff (\texttt{TRUE}\,\mathbf{U}\,\varphi$;
– $\mathbf{G}\,\varphi \iff \neg F\neg\varphi$;
– $\varphi\,\mathbf{R}\,\psi \iff \neg(\neg\varphi\,\mathbf{U}\,\neg\psi)$.

Since we mainly focus on P/T nets in this thesis, we also specify how the semantics of CTL* changes when a P/T net is used. This is possible due to the fact that the reachability graph of a P/T net is an LTS and because CTL* is defined with respect to an LTS.

Definition 3.2.3 (Semantic extension of CTL* for P/T nets).
States are replaced by markings and the general-purpose atomic propositions are replaced by the atomic propositions specific to a P/T net.

1. $m \models \texttt{TRUE}$, $m \not\models \texttt{FALSE}$;
2. $m \models \texttt{FIREABLE}(t) \iff$ *t is enabled in m;*
3. $m \models \texttt{DEADLOCK} \iff$ *there is no enabled transition in m;*
4. $m \models k_1p_1 + \cdots + k_np_n \leq k \iff k_1m(p_1) + \cdots + k_nm(p_n) \leq k$;

In some cases, we want to avoid negations in front of path quantifiers or temporal operators. For such cases, we need a CTL* formula to be in *negation normal form.*

Definition 3.2.4 (Negation normal form).
A CTL formula is in* negation normal form *if every negation occurs directly in front of an atomic propositions.*

With CTL*, system relevant properties can be build. For the following examples, we use several intuitive propositions such as liveness, requirement, acknowledgment, read and some others that we do not describe further.

1. It is possible to quit a program at any time: $\mathbf{GEF}\, quit$.
2. The variable x is live: $\mathbf{AGEF}\, x$.
3. Every request is followed by an acknowledgment: $\mathbf{AG}\,(Reg \implies \mathbf{AF}\, Ack)$.
4. An error e leads to termination t: $\mathbf{G}\,(e \implies \mathbf{F}\, t)$.
5. A variable can not be read while it is written: $read\,\mathbf{U}\,write$
6. On all path, the system will eventually stabilize: $\mathbf{AFG}\, stable$.

We introduce several subclasses of CTL*. We will give a formal definition of the subclasses in the next sections. The subclasses are computational tree logic (CTL), linear time temporal logic (LTL), universal computational tree logic* (ACTL*) and existential computational tree logic* (ECTL*). The relations between these subclasses are depicted in Figure 3.2. The intersection of all classes contains atomic propositions.

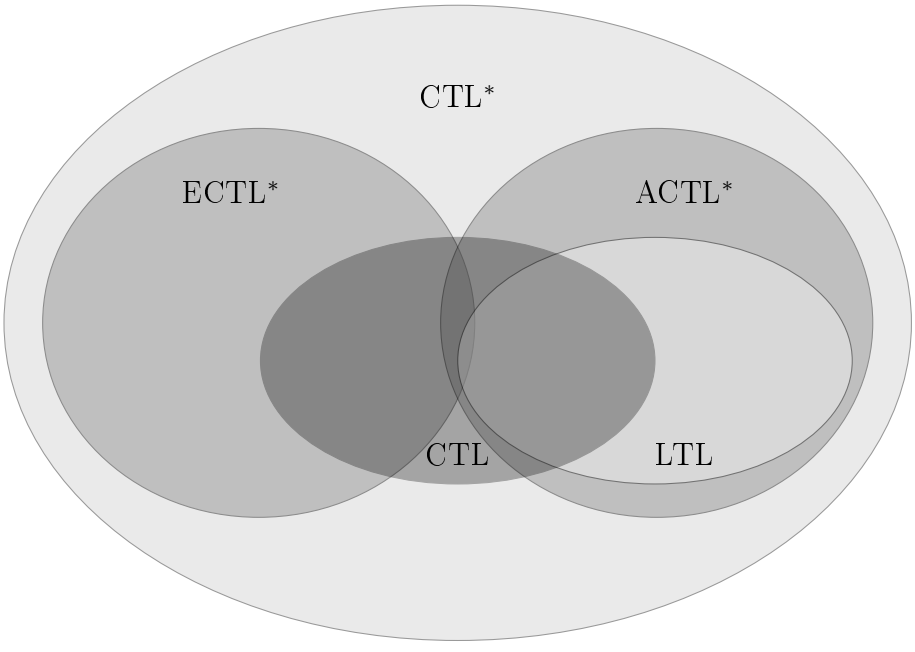


Figure 3.2: Temporal logic classes as Venn diagram.

3.3 Linear time temporal logic

The first class we consider is linear time temporal logic (LTL) [103]. In LTL, temporal operators are used to describe events along a single computation path. LTL is CTL* without path quantifiers, except one initial universal path quantifier. LTL formulas have the form $\mathbf{A}\,\varphi$, where φ is a path formula, in which atomic propositions are the only allowed state formulas.

Definition 3.3.1 (LTL).
An LTL path formula is either:

- *If α is an atomic proposition, then α is an LTL formula;*
- *If φ_1 and φ_2 are LTL formulas, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\mathbf{X}\,\varphi_1$, $\mathbf{F}\,\varphi_1$, $\mathbf{G}\,\varphi_1$, $\varphi_1\,\mathbf{U}\,\varphi_2$, and $\varphi_1\,\mathbf{R}\,\varphi_2$ are LTL formulas.*

A labeled transition system satisfies an LTL formula if all paths starting in the initial state do.

If it is clear from the context hat $\mathbf{A}\,\varphi$ is meant to be an LTL formula, then we omit the universal path quantifier $\mathbf{A}$ and simply write φ.
LTL is used to specify properties such as mutual exclusion algorithms, leader election protocols, communication channels and many more [15]. There are two main types of properties that can be expressed with LTL. The first type are *safety* properties. Safety properties state that *something bad will never happen.* This can be expressed as "globally not φ holds" ($\mathbf{G}\,\neg\varphi$). And the second type are *liveness* properties. Liveness properties state that *something good keeps happening.* This can be expressed as "globally it holds that eventually φ will happen" ($\mathbf{GF}\,\varphi$) or as "globally it holds that if φ_1 happens then eventually φ_2 will happen" ($\mathbf{G}(\varphi_\mathbf{1} \implies \mathbf{F}\,\varphi_\mathbf{2})$).

Definition 3.3.2 (Stutter-invariant).
An LTL formula φ is stutter-invariant *if for all finite paths w_1, all markings m and all infinite paths w_2 it holds: $w_1 m w_2 \models \varphi \iff w_1 m m w_2 \models \varphi$.*

Since LTL formulas describe events along a computation path, LTL formulas can be stutter-invariant or stutter-sensitive. If duplicating a state in the computation tree or removing a duplicated state does not change the outcome of the formula, then the LTL formula is stutter-invariant. Otherwise, the LTL formula is stutter-sensitive.

Lemma 3.3.1 (Stutter-invariant formulas [99])**.** *If an LTL formula φ does not contain the $\mathbf{X}$-operator then φ is stutter-invariant.*

Note, the absence of the **X**-operator is a sufficient but not a necessary condition for stutter-invariance. In the following we call the set of LTL formulas without the **X**-operator $\mathrm{LTL}_{-\mathbf{X}}$. There are properties that cannot be specified with LTL such as: from any state it is always possible to get to the reset state. This property is expressible in CTL*: **AGEF** *reset* but not in LTL.

3.4 Computational tree logic

Another sublogic of CTL* is computational tree logic (CTL) [20] also called the branching time logic. In CTL the temporal operators quantify over all possible paths from a given state. CTL formulas consist of atomic propositions, Boolean connectors, and pairs of a path quantifier combined with a temporal operator. That is, each path quantifier must be immediately succeeded by a temporal operator.

Definition 3.4.1 (CTL).
CTL *is the subset of CTL* with the restriction that the syntax of path formulas have the following form:*

- *If* φ_1 *and* φ_2 *are state formulas, then* $\mathbf{X}\,\varphi_1$, $\mathbf{F}\,\varphi_1$, $\mathbf{G}\,\varphi_1$, $\varphi_1\,\mathbf{U}\,\varphi_2$, *and* $\varphi_1\,\mathbf{R}\,\varphi_2$ *are path formulas.*

A labeled transition system satisfies a CTL formula if its initial state does.

With this, there are ten basic CTL operators:

- $\mathbf{AX}\,\varphi$ and $\mathbf{EX}\,\varphi$,
- $\mathbf{AF}\,\varphi$ and $\mathbf{EF}\,\varphi$,
- $\mathbf{AG}\,\varphi$ and $\mathbf{EG}\,\varphi$,
- $\mathbf{A}\,(\varphi_1\,\mathbf{U}\,\varphi_2)$ and $\mathbf{E}\,(\varphi_1\,\mathbf{U}\,\varphi_2)$,
- $\mathbf{A}\,(\varphi_1\,\mathbf{R}\,\varphi_2)$ and $\mathbf{E}\,(\varphi_1\,\mathbf{R}\,\varphi_2)$.

However, not all ten basic operators are needed. Some of these operators can be expressed by others. One possible minimal set of Boolean connectors and basic CTL operators is $\{\mathtt{TRUE}, \neg, \wedge, \mathbf{EX}, \mathbf{EU}, \mathbf{AU}\}$ [24]. With this, we can recreate all other operators via the application of tautologies:

$$
\begin{aligned}
\mathbf{EF}\,\varphi &= \mathbf{E}\,(\mathtt{TRUE}\,\mathbf{U}\,\varphi);\\
\mathbf{AX}\,\varphi &= \neg\mathbf{EX}\,(\neg\varphi);\\
\mathbf{AF}\,\varphi &= \neg\mathbf{EG}\,(\neg\varphi);\\
\mathbf{AG}\,\varphi &= \neg\mathbf{EF}\,(\neg\varphi);\\
\mathbf{A}\,(\varphi_1\,\mathbf{U}\,\varphi_2) &= \neg\mathbf{E}\,(\neg\varphi_2\,\mathbf{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)) \wedge \neg\mathbf{EG}\,(\varphi_2);\\
\mathbf{A}\,(\varphi_1\,\mathbf{R}\,\varphi_2) &= \neg\mathbf{E}\,(\neg\varphi_1\,\mathbf{U}\,\neg\varphi_2);\\
\mathbf{E}\,(\varphi_1\,\mathbf{R}\,\varphi_2) &= \neg\mathbf{A}\,(\neg\varphi_1\,\mathbf{U}\,\neg\varphi_2).
\end{aligned}
$$

CTL and LTL are both subclasses of CTL*. There are many formulas that are only expressible with one or the other. The aforementioned property, *from any state it is always possible to get to the reset state*, is not expressible with LTL, but it can be expressed with CTL: **AG EF** *reset*. On the other hand, the property that *a system will eventually stabilize* is not expressible with CTL but with LTL: **AFG** *stable*. Both temporal logics also have a common intersection, where some formulas can be expressed both in LTL and CTL, respectively. The property that *a request can always occur* is expressible with the CTL and LTL formula **AG** *request*.

3.5 ACTL* and ECTL*

Two other subclasses of CTL* are the universal (ACTL*) and the extensional (ECTL*) fragments of CTL*. These two temporal logics are restricted to either the all path quantifier (**A**) or the existential path quantifier (**E**), respectively.

Definition 3.5.1 (ACTL*).
ACTL is the subset of CTL* formulas that are in negation normal form and free of existential path quantifiers.*

Definition 3.5.2 (ECTL*).
ECTL is the subset of CTL* formulas that are in negation normal form and free of universal path quantifiers.*

ACTL* is a superset of LTL.

Chapter 4

Model checking

Model checking [19, 21, 39, 104] answers the question if a given system meets a given specification. In chapter 1, we have motivated model checking as field of research and have restricted our focus on Petri net model checking. In this chapter, we introduce formally the model checking problem and the algorithms we use. The goal of this thesis is to improve model checking efficiency. For our newly developed techniques, the algorithms from this chapter serve as a reference for experimental validation. For model checking a whole range of different methods have been introduced in the literature over the years. The methods can be classified into *symbolic* and *explicit* techniques.

Symbolic model checking [14], works with an implicit representation of the set of reachable states, respectively the set of paths through the transition system. Symbolic methods usually have the disadvantage that if they are not able to build a suitable representation of the state space, then they are not able to answer even the simplest queries. On the other hand, if they are able to build a suitable representation, then they are able to answer several queries at once.

Explicit model checking generates and evaluates states and paths one by one. The "on-the-fly" advantage of this approach is that exploration of state space can be aborted as soon as the query goal is reached. Of course this can also be a disadvantage if the query goal is unreachable. Then, we have to explore the whole state space.

At present, leading Petri net model checkers such as TAPAAL [34] and LoLA [142] use explicit model checking algorithms. However, ITS-Tools [122] is making up ground with its symbolic solution engine in recent years.

Several symbolic techniques are based on different types of decision diagrams [12] such as binary decision diagrams [96], hierarchical set decision diagrams [30], interval decision diagrams [124], or multi-core decision dia-

grams [133]. Other symbolic techniques are based on automata [9, 42], or another one uses a Boolean satisfiability problem (SAT) approach [17].
There are also several explicit model checking approaches that differ from the method that we are going to introduce in this chapter. They are based for example on dependency graphs [31,84], coverability graphs [45,63], sweep-line methods [16], or random walk methods [88, 108].
In this work, we do not use any of these explicit approaches due to the following reasons. Even though dependency graphs have been around for quite some time, their breakthrough for model checking is rather new [31]. TAPAAL ranked first in the CTL category of the MCC 2018 using dependency graphs [66]. Coverability graphs have their strengths in the verification of unbounded P/T nets. The sweep-line method and the random walk methods preserve not all properties.
Next to state space model checking algorithms there are also structural methods that are based on the structure of the P/T net. For example, such methods are the state equation [140], or the siphon-trap property [94]. There are many more algorithms and procedures for the verification.
This chapter describes algorithms used for explicit model checking and the chapter is organized as follows. Section 4.1 formalizes the model checking problem and states complexity results. Section 4.2 gives a brief overview of explicit model checking algorithms. Section 4.3 and Section 4.4 sketch, how exactly the explicit CTL and LTL model checking algorithms work.

4.1 Complexity

Model checking is the automated verification that systematically checks if a given model of a system holds for a given formal property. The general model checking procedure is illustrated in Chapter 1, Figure 1.1.

Definition 4.1.1 (Model checking problem [113]).
Let S range over all labeled transition systems and $\varphi \in L$ a formula where L is one of the temporal logics CTL*, LTL, or CTL. The model checking problem MC for L is the decision problem associated with the language (set)

$$MC(L) := \{(S, \varphi) \mid S \models \varphi\}.$$

This means, for any labeled transition system $TS = (S, A, \to, s_0)$ and any $\varphi \in L$, $MC(L)$ is the problem of deciding whether $TS \models \varphi$ or not. To measure the cost of algorithms deciding whether $TS \models \varphi$ or not, the size of the inputs are used. The size of $|TS|$ is given by the size of its underlying graph, that is, the sum $|S| + | \to |$ of the number of nodes and the number of

edges. The size of $|\varphi|$ is the number of symbols in φ. The following theorem states the complexity of model checking. For basic notions of complexity theory the reader is referred to [95].

Theorem 4.1.1 (Complexity of model checking)**.** *Let φ be a formula of the respective temporal logic and $|\varphi|$ the size of φ. Further, let TS be an LTS and $|TS|$ the size of TS. The model checking problem for*

- CTL* *is PSPACE-complete and can be solved in time $\mathcal{O}(2^{|\varphi|}|TS|)$ [22, 113];*
- LTL *is PSPACE-complete and can be solved in time $\mathcal{O}(2^{|\varphi|}|TS|)$ [113, 115];*
- CTL *is P-complete and can be solved in time $\mathcal{O}(|\varphi||TS|)$ [22, 113].*

The CTL* and LTL model-checking problem remains computationally hard and is "probably" not efficiently solvable. Even in simple cases, with only one atomic proposition LTL model checking remains PSPACE-complete [37]. Although there is a big difference in complexity between LTL/CTL* and CTL, in reality this situation is not so clear-cut as shown in [113] and, thus, this should not be interpreted as: "CTL model checking is more efficient than LTL model checking" [15]. The reason for this is that in real world situations, $|\varphi|$ is a rather small and $|TS|$ is usually quite large [113]. This is highlighted in the MCC, too. In the CTL category of recent model checking contests, even less problems have been solved than in the LTL category.
In this thesis, we focus on LTL and CTL model checking, because there are more efficient techniques available for completely path or state based formulas.

4.2 Explicit model checking

Explicit model checking explores the reachable states of a system one by one, starting with the initial state. Together with the corresponding transition occurrences, these states form a labeled transition system. Exploring a state means to recursively visit all its immediate successors. As search strategy depth-first search (DFS), breadth first search (BFS) or some other search heuristic such as the one proposed by Jensen et al. in [58] are used. BFS has the disadvantage that there is no fast BFS-algorithm detecting SCCs. Therefore, DFS dominates, in case SCCs of the transition system need to be recognized. Already visited states are stored to avoid exploring a state multiple times and to ensure termination. To this end, prefix trees [35] or similar data structures [60] can be used.
As a reference algorithm in this thesis, we use Tarjan's algorithm [119] that combines a DFS with the detection of SCCs to build the reachability graph $R_N = (M, E)$ of a P/T net N in time $\mathcal{O}(|M| + |E|)$.

4.3 CTL model checking

This section addresses CTL model checking, which means a decision algorithm that checks whether $TS \models \varphi$ for a given labeled transition system TS and a given CTL formula φ. In the sequel, we describe how explicit CTL model checking works. Let $N = (P, T, F, W, m)$ be a P/T net with the corresponding reachability graph TS and φ a CTL formula.
We consider *local* model checking [1], that is, we want to evaluate φ just for the initial marking m_0. Other markings are only considered as far as necessary for determining the value at m_0. In global model checking [1], one would be interested in the value of φ in all reachable markings. In other words, global model checking procedures first compute all states of a transition system that satisfy φ and then check whether these states are included in the set of initial states. The advantage of local model checking is that it directly answers the question whether the initial state satisfies φ.
Vergauwen and Lewi proposed in [139] an explicit local CTL model checking algorithm. This algorithm has the advantage that it is goal-oriented, which means that only a necessary part of the state space is investigated. In contrast, the global labeling algorithm introduced in [21] needs to check a priori all subformulas for all states. In this thesis, we use the algorithm from Vergauwen and Lewi, called ALMC, as a reference, because we are interested in explicit, local, on-the-fly model checking. Furthermore, it is the CTL model checking algorithm implemented in LoLA, and serves as a base for our extensions.

Theorem 4.3.1 (ALMC algorithm [139])**.** *Let $N = (P, T, F, W, m)$ be a P/T net with reachability graph TS and φ a CTL formula. ALMC(TS, φ) is correct and runs in time $\mathcal{O}(|\varphi||TS|)$.*

We briefly sketch this algorithm. The following description is based on our conference paper [79]. The main procedure can be seen in Algorithm 1. Remember that $\{\texttt{TRUE}, \neg, \wedge, \mathbf{EX}, \mathbf{EU}, \mathbf{AU}\}$ is a minimal set of operators to verify CTL formulas. The remaining CTL operators can be traced back to the two until operators using tautologies (see Section 3.4).
In CTL all (sub-)formulas concern states. Assume that, attached to every marking, there is a vector that has an entry for every subformula ϕ of the given CTL query φ. The value of a single entry can be true, false, or unknown which are represented by $T, F, ?$. This vector is used to keep track of the gathered information to avoid unnecessary computations. The states m together with the values of each subformula form $L(m, \phi)$ which represents the truth value of ϕ in m. To access the value of a subformula in a marking m, one has to inspect the corresponding value. If it is unknown, a recursive

Algorithm 1: CTL model checking: main procedure

Input : P/T net $N = (P, T, F, W, m)$ with reachability graph TS, CTL formula φ.

Output: `TRUE` iff $(TS, m) \models \varphi$, `FALSE` iff $(TS, m) \not\models \varphi$.

```
1  CTL(m, φ)
2      if L(m, φ) ≠ ? then return;
3      switch φ do
4          case φ = atomic proposition do compute L(m, φ);
5          case φ = ¬ψ do
6              CTL(m, ψ);
7              if L(m, ψ) then L(m, φ) = F;
8              else L(m, φ) = T;
9          case φ = ψ ∧ χ do
10             CTL(m, ψ);
11             if L(m, ψ) then CTL(m, χ); L(m, φ) = L(m, χ);
12             else L(m, φ) = F;
13         case φ = EX ψ do
14             forall m' : m -t-> m' do
15                 CTL(m', ψ);
16                 if L(m', ψ) = T then L(m, φ) = T; return;
17             L(m, φ) = F;
18         case φ = AX ψ do
19             forall m' : m -t-> m' do
20                 CTL(m', ψ);
21                 if L(m', ψ) = F then L(m, φ) = F; return;
22             L(m, φ) = T;
23         case φ = A (ψ U χ) do CheckAU(m, ψ, χ);
24         case φ = E (ψ U χ) do CheckEU(m, ψ, χ);
```

procedure is launched to evaluate ϕ in m. If ϕ is an atomic proposition or a Boolean combination of subformulas, evaluation is trivial. For evaluating a formula of shape $\mathbf{EX}\,\varphi$ or $\mathbf{AX}\,\varphi$, one needs to proceed to the immediate successor states and evaluate φ in those states. If the successor marking has not been visited yet, it is added to the set of visited markings and its vector of values is initialized.

We continue with the procedures for $CheckAU(m, \psi, \chi)$, which is shown in

Algorithm 2.

Theorem 4.3.2 (Correctness of $CheckAU(m, \psi, \chi)$ [139]). *The procedure $CheckAU(m, \psi, \chi)$ is correct if and only if $m \models \mathbf{A}(\psi \mathbf{U} \chi)$.*

Algorithm 2: CheckAU(m, ψ, χ) procedure

```
   variables: black: set of seen nodes
              gray: set of active nodes
              white: set of not seen nodes
 1 black := gray := ∅; white := V;
 2 CheckAU(m, ψ, χ)
 3   white := white\{m}; gray := gray ∪{m};
 4   if L(m, A(ψ U χ)) = F then exit CheckAU;
 5   if L(m, A(ψ U χ)) = T then return;
 6   CTL(m, χ);
 7   if L(m, χ) = T then L(m, A(ψ U χ)) := T; return;
 8   L(m, A(ψ U χ)) := F;
 9   CTL(m, ψ);
10   if L(m, ψ) = F then exit CheckAU;
11   forall m' : m --t--> m' do
12     if m' ∈ white then CheckAU(m', ψ, χ);
13     else if m' ∈ gray then exit CheckAU;
14   gray := gray\{m}; black := black∪{m};
15   L(m, A(ψ U χ)) = T;
```

The procedure is called if φ has the shape $\mathbf{A}(\psi \mathbf{U} \chi)$. A depth-first search is launched from m, aiming at the detection of a counterexample. The search proceeds through markings that satisfy ψ, violate χ, and for which $\mathbf{A}(\psi \mathbf{U} \chi)$ is recorded as unknown. Whenever the space of states satisfying these assumptions is left, there is a reaction that does not require continuation of the search beyond that marking, as follows.
If χ is satisfied, or $\mathbf{A}(\psi \mathbf{U} \chi)$ is recorded as true in any marking m', the algorithm backtracks since there cannot be a counterexample path containing m'. If χ and ψ are violated, or $\mathbf{A}(\psi \mathbf{U} \chi)$ is recorded as false, the search is exited since the search stack forms a counterexample for $\mathbf{A}(\psi \mathbf{U} \chi)$ in m. If a marking m' is hit on the search stack, a counterexample is found, too (a path where ψ and not χ hold forever). The depth-first search assigns a value different from unknown to all states visited during the search: For markings on the search stack (i.e., participating in the counterexample), $\mathbf{A}(\psi \mathbf{U} \chi)$ is

false, while for states that have been visited but already removed from the search stack, $\mathbf{A}\,(\psi\,\mathbf{U}\,\chi)$ is actually true.
We continue with the remaining procedure for $CheckEU(m, \psi, \chi)$ which is depicted in Algorithm 3.

Theorem 4.3.3 (Correctness of $CheckEU(m, \psi, \chi)$ [139]). *The procedure $CheckEU(m, \psi, \chi)$ is correct if and only if $m \models \mathbf{E}\,(\psi\,\mathbf{U}\,\chi)$.*

Algorithm 3: CheckEU(m, ψ, χ) procedure

```
   variables: Tarj: stack of nodes
              white: set of not seen nodes
              maxdfs ∈ ℕ: maximal DFS number
 1 Tarj := empty stack; white:=V; maxdfs = 0;
 2 CheckEU(m, ψ, χ)
 3   if L(m, E(ψ U χ)) = T then exit CheckEU;
 4   if L(m, E(ψ U χ)) = F then L(m, E(ψ U χ)) := F; return;
 5   CTL(m, χ);
 6   if L(m, χ) = T then L(m, E(ψ U χ)) := T; exit CheckEU;
 7   CTL(m, ψ);
 8   if L(m, ψ) = F then L(m, E(ψ U χ)) := F; return;
 9   L(m, E(ψ U χ)) := T; m.dfs = maxdfs; maxdfs += 1;
10   white := white\{m}; push(Tarj,m); m.lowlink := m.dfs;
11   forall m' : m --t--> m' do
12     if m' ∈ white then
13       CheckEU(m', ψ, χ);
14       m.lowlink := MIN(m.lowlink, m'.lowlink);
15     else if m' ∈ Tarj then m.lowlink := MIN(m.lowlink, m'.dfs);
16   if m.lowlink = m.dfs then
17     repeat m' := pop(Tarj); L(m, E(ψ U χ)) = F; until m = m'
```

$CheckEU(m, \psi, \chi)$ is called if φ has the shape $\mathbf{E}\,(\psi\,\mathbf{U}\,\chi)$. A similar depth-first search is launched, aiming at the detection of a witness path. Tarjan's algorithm [119] is integrated to detect SCCs during the search. It proceeds through markings that satisfy ψ, violate χ, and for which $\mathbf{E}\,(\psi\,\mathbf{U}\,\chi)$ is recorded as unknown. If a marking m' is hit where χ is satisfied, or $\mathbf{E}\,(\psi\,\mathbf{U}\,\chi)$ is true, a witness is found. In states where ψ and χ are violated, or $\mathbf{E}\,(\psi\,\mathbf{U}\,\chi)$ is known to be false, the algorithm backtracks since there cannot be a witness path containing such a marking. Again, a value different from unknown is

assigned to every marking visited during the search. Markings that are on the search stack as well as markings that are not on the search stack but appear in SCCs that have not yet been completely explored, get value true. An SCC is not yet fully explored if it contains elements that are still on the search stack. Then, however, a path to the search stack extended by the remaining portion of the search stack forms a witness. For markings in SCCs that have been completely explored, $\mathbf{E}(\psi \mathbf{U} \chi)$ is false.
We can see that existential and universal until operators are not fully symmetric. This is due to the fact that a cycle of markings that satisfy ψ and violate χ, form a counterexample for universal until but no witness for existential until. Any SCC with more than one member would contain such a cycle. Consequently, universal until will never remove markings from the search stack without closing a whole (singleton) SCC.

Theorem 4.3.4 (Runtime of ALMC [139]). *Let $N = (P, T, F, W, m)$ be a P/T net with reachability graph TS and φ a CTL formula. The overall runtime of the ALMC algorithm is $O(|\varphi||TS|)$ where $|\varphi|$ is the length of φ (the number of subformulas), and $|TS|$ is the number of markings reachable from m_0.*

Since every search assigns values to all visited markings, the overall runtime of the algorithm is $O(|\varphi||R_N|)$. The dominating factor for complexity is $|R_N|$ due to the state explosion problem.
In the implementation of the algorithm, we use the concept of dynamic programming to speed up the computations. This means, we explicitly store the intermediate results and use the stored values in subsequent queries instead of calling the subroutine again.

4.4 LTL model checking

This section is concerned with the LTL model checking problem. For a given labeled transition system TS and LTL formula φ, the problem is to check whether $TS \models \varphi$.
We consider on-the-fly automata-theoretic LTL model checking as originally suggested by Vardi and Wolper [135]. Since this approach uses Büchi automata [13], we also briefly introduce the standard definition of Büchi automata later in this section and outline how they are used in LTL model checking.
The basic idea of automata-theoretic explicit LTL model checking is to explore the states of the product automaton of two ω-automata [107], which accept or reject infinite inputs. Since ω-automata handle infinite paths, we

denote the accepted language of an ω-automata by L^ω. A Büchi automaton is a special form of an ω-automaton. The first automaton of the product automaton represents the system as a reachability graph TS of a given P/T net. The second automaton represents the negation of the property under investigation φ. More precisely, L^ω_{TS} is the set of all paths that can be executed in TS and L^ω_φ is the set of all paths that satisfy φ. TS satisfies φ if and only if every path in TS satisfies φ, i.e., $L^\omega_{TS} \subseteq L^\omega_\varphi$. We can equivalently decide whether the intersection of TS and negation of φ is empty:

$$L^\omega_{TS} \cap L^\omega_{\neg\varphi} = \emptyset. \tag{4.1}$$

This means, the property φ is verified by TS if the product automaton rejects every input. Algorithm 4 shows the procedure of on-the-fly automata-theoretic LTL model checking.

Algorithm 4: LTL model checking algorithm(TS, φ)

Input : P/T net $N = (P, T, F, W, m)$ with reachability graph TS, LTL formula φ.
Output: `TRUE` iff $(TS, m) \models \varphi$, `FALSE` iff $(TS, m) \not\models \varphi$.
variables: $B_{\neg\varphi}$: Büchi automaton of the negated formula φ
B^*: product (Büchi) automaton of TS and $B_{\neg\varphi}$

```
1 LTL(TS, φ)
2   Build B_¬φ;
3   Build B* from TS and B_¬φ;
4   if there exist SCC in B* that contains elements of every
      acceptance set then
5     build counterexample from found SCC;
6     return FALSE;
7   else
8     return TRUE;
```

Checking whether the product automaton language is empty requires to check if there is no cycle with elements of every acceptance set. This is done by computing the SCCs using for example Tarjan's algorithm. Instead of using Tarjan's algorithm for detecting SCCs, it is also possible to use the double search algorithm attributed to Kosaraju and first published in [114]. A different approach would be to use nested depth-first search [56]. We use SCC detecting algorithms, since they have on average the best performance [46].

Theorem 4.4.1 (On-the-fly automata-theoretic LTL MC [135])**.** *Let $N = (P, T, F, W, m)$ be a P/T net with reachability graph TS and φ an LTL formula. The LTL model checking algorithm(TS, φ) is correct and the overall*

runtime is $\mathcal{O}(2^{|\varphi|}|TS|)$ where $|\varphi|$ is the length of φ (the number of subformulas), and $|TS|$ is the number of markings reachable from m_0.

The overall running time of the algorithm is $\mathcal{O}(2^{|\varphi|}|TS|)$. Just like CTL model checking, LTL model checking is dominated in its complexity by the factor $|TS|$ and therefore suffers from the state explosion problem.
We continue with the introduction of Büchi automaton [13]. As reference for the following definitions see [15, 104].

Definition 4.4.1 (Büchi automaton).
A Büchi automaton $B = (\Sigma, Q, \delta, Q_0, Q_F)$ *consists of*

- Σ *an alphabet;*
- Q *a finite set of states;*
- $\delta : Q \times \Sigma \to 2^Q$ *a transition function;*
- $Q_0 \subset Q$ *a set of initial states;*
- $Q_F = \{F_1, \ldots, F_n\}$ *with* $F_i \subseteq Q$ *consisting of finitely many acceptance sets.*

If the Büchi automaton has only a single initial state q_0, we sometimes write q_0 instead of Q_0 for better readability. A *run* through a Büchi automaton is defined as follows.

Definition 4.4.2 (Run).
A run *for $\sigma = Q_1 Q_2 \cdots \in \Sigma^\omega$ denotes an infinite sequence $q_0 q_1 q_2 \ldots$ of states in B such that $q_0 \in Q_0$ and $q_i \xrightarrow{A_i} q_{i+1}$ for all $i \geq 0$.*

We continue with the acceptance criterion for Büchi automata.

Definition 4.4.3 (Acceptance criterion for Büchi automata).
A run $q_0 q_1 q_2 \ldots$ is accepting if for every $F \in Q_F$ there exist infinitely many $j \in \mathbb{N}$ with $q_j \in F$.

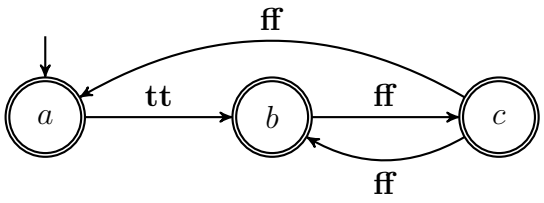
As seen in Equation 4.1 all states of the automaton for TS are accepting. The following construction transforms a finite LTS to a Büchi automaton.

Definition 4.4.4 (LTS to Büchi automaton).
Let $TS = (S, A, \to, s_0)$ be a labeled transition system, AP a set of atomic propositions and $L : S \times AP \to \{\mathbf{tt}, \mathbf{ff}\}$ a labeling function. The corresponding Büchi automaton $B = (\Sigma, Q, \delta, Q_0, Q_F)$ consists of

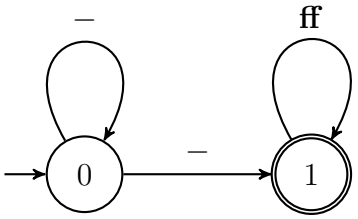
- $\Sigma = 2^{AP}$ *an alphabet symbol corresponds to AP;*

- $Q = S$;
- $(s, a, s') \in \delta = \begin{cases} (s, s') \in \rightarrow \text{ and } p \in a & \text{iff } L(s', p) = \mathbf{tt} \\ \emptyset, & \text{otherwise;} \end{cases}$
- $Q_0 = \{s_0\}$;
- $Q_F = S$.

Figure 4.1 shows two example Büchi automata with their initial states and acceptance sets. The arc labels $\mathbf{tt}, \mathbf{ff}, -$ represent true, false, everything.



(a) The Büchi automaton represents the reachability graph TS of a P/T net N with: $Q_0 = \{a\}$ and $Q_F = \{\{a, b, c\}\}$.



(b) The Büchi automaton represents the formula $\varphi = \mathbf{FG}(\neg a)$ with respect to TS with: $Q_0 = \{0\}$ and $Q_F = \{\{1\}\}$.

Figure 4.1: Büchi automata

Proposition 4.4.2 (Transform LTL to Büchi automaton [134])**.** *For every LTL formula φ, there exists a Büchi automaton that accepts exactly those paths which violate φ.*

Vardi and Wolper showed in [134] that every LTL formula φ, with size $|\varphi|$ which is the number of symbols in φ, can be translated to a Büchi automaton B_φ, with size $|B_\varphi| \leq 2^{|\varphi|}$, which accepts exactly those infinite runs that satisfy φ.

Definition 4.4.5 (Product system).
Let $B^1 = (\Sigma, Q^1, \delta^1, Q_0^1, Q_F^1)$ and $B^2 = (\Sigma, Q^2, \delta^2, Q_0^2, Q_F^2)$ be two Büchi automata. Assume that the acceptance set of B^2 is F^2 and that B^1 represents the TS which means the acceptance set of B^1 is Q_F^1. The product system of B^1 and B^2 is a Büchi automaton $B^* = (\Sigma, Q^*, \delta^*, Q_0^*, Q_F^*)$ and consists of

- $\Sigma = \Sigma$;
- $Q^* = Q^1 \times Q^2$;
- $((q_1, q_2), a, (q_1', q_2')) \in \delta^*$ *if and only if* $(q_1, a, q_1') \in \delta^1$ *and* $(q_2, a, q_2') \in \delta^1$;

- $Q_0 = Q_0^1 \times Q_0^2$;
- $Q_F = Q_F^1 \times F^2$.

Lemma 4.4.3 (Product system accepts $\neg\varphi$)**.** *The product automaton represents a combination of a marking of the reachability graph TS and the state of the Büchi automaton. It accepts exactly those infinite paths of TS which violate property φ.*

This means, to verify φ, we check whether there exists a path in the product automaton that can be executed in the reachability graph. If we have found such a path, then φ does not hold in the system. Otherwise, the property φ is satisfied in the system.

Lemma 4.4.4 (Language of the product system [134])**.** *The language of the product automaton B^* is not empty if and only if there is a nontrivial SCC C in B^* reachable from the initial state and if C contains elements of every acceptance set.*

Proposition 4.4.5 (Emptiness check [134])**.** *There exists an infinite path, realizable in the reachability graph (M, E) of a given P/T net N and accepted by Büchi automaton B if and only if the product automaton $(M, E) \cap B$ has an accepting run.*

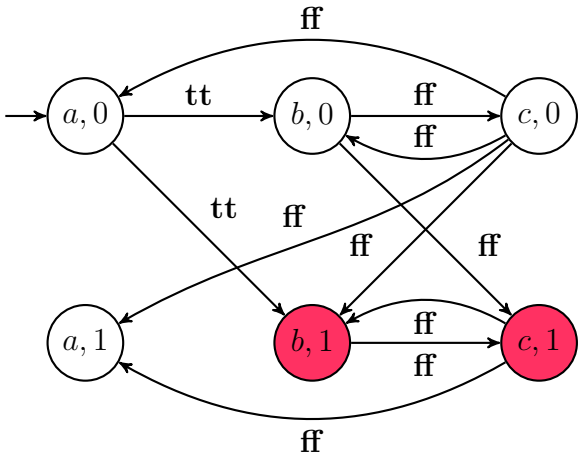


Figure 4.2: Product automaton of Figure 4.1a and 4.1b with $Q_0 = \{(a, 0)\}$ and $Q_F = \{Q, \{(a, 1), (b, 1), (c, 1)\}\}$.

Figure 4.2 shows the product automaton of TS and φ from Figure 4.1a and 4.1b, respectively. We start the exploration of the product automaton in the initial state $(a, 0)$. We continue with the exploration using DFS. We stop

the exploration in the moment, we encounter the nontrivial SCC containing the nodes $(b, 1)$ and $(c, 1)$. At this point, we have found the counterexample $a(bc)^*$ and know that the formula does not hold in the system.
Since we use on-the-fly verification, it is much easier to reject a property than to prove one. If an LTL formula does not hold, the search can be interrupted as soon as an accepting SCC is found. If an LTL formula holds, then the whole product automaton needs to be explored.
This procedure works with any transition system TS and, thus, reduction techniques such as partial order reduction which we are going to introduce in Chapter 10 can be used to reduce TS. This means, model checking with Büchi automata can be combined with other reduction techniques.
Several different other automata types can be used instead of Büchi automata. For example, Streett [102] and Rabin [2,69] automata are also suitable candidates to represent an LTL formula. We use Büchi automata, since its acceptance criterion is a special case of the parity criterion [90], which in turn is a special case of both the Rabin and the Streett criterion [6] and we do not need the more general acceptance behavior.

Part II

Reduction techniques

Portfolio approach means that several methods run in parallel and the fastest successful method determines the runtime. To benefit from the wide range of different verification methods introduced in Chapter 4, we believe it is meaningful to use a portfolio approach. Using portfolio approaches opens new opportunities because we gain room for incomplete methods, which, in some cases, return a definite result more efficiently but sometimes return an indefinite result or do not terminate at all. The chapters in this part introduce two such methods.
This part is organized as follows. In the next Chapter 5, we propose a verification technique that is based on the reduction of tokens in the initial marking of a P/T net. We continue in Chapter 6 with the introduction of a linear algebraic approach to solve certain formulas.

Chapter 5

Verification with under-approximation

This chapter introduces an incomplete method that complements our portfolio. This method handles two of our research goals, namely it is a *model simplification* and it represents an *alternative reduction technique*. We have published this method with its results in [80] (and submitted [81]). The following chapter is based on these publications.

Model checking is subject to the state explosion problem. If the system is a P/T net, one possible cause for this problem is the size of the model. Another reason is the cardinality of the initial marking, i.e., the number of tokens on the places of the P/T net in the initial state. This chapter focuses on P/T nets that have a large number of initial tokens. In many models, a scaling parameter describes how many tokens are in the initial marking. We call such P/T nets *token-scaling models*. Token-scaling models are widespread in several fields. For example, in biochemistry the tokens represent chemical or biological entities, such as molecules, and their initial number is subject to a model parameter. In scheduling problems, tokens stand for available resources which are variable, to. Another example is an election protocol where tokens represent voters.

To tackle the state explosion problem on token-scaling models, we introduce a method for under-approximate model checking. The idea is to simply reduce the number of tokens in the initial marking, on places which have more than one token on them. This will essentially reduced the state space. The question is how this affects the result of model checking. If we find a witness path (counterexample) for the property under investigation in the reduced state space, then this property also holds in the original state space. Otherwise, the result is indefinite.

Section 5.1 starts with a motivational example. In Section 5.2, we introduce

the theory behind this method. Section 5.3 introduces heuristics to reduce the initial marking, followed by some implementation remarks in Section 5.4. Section 5.5 carries out experimental validation which is discussed and concluded in Section 5.6. We also give directions for future research.

5.1 Motivational example

An example token-scaling model of a biochemical process is *Angiogenesis* [93]. The model describes the formation of new vessels from existing ones. It consists of 39 places, 64 transitions, and 185 arcs. If the scaling parameter is set to 25, meaning that every initially marked place has 25 tokens on it, the state space contains more than $4.3 \cdot 10^{19}$ reachable markings.
The *RobotManipulation* model [65] as seen in Figure 5.1 is an example for a scheduling problem. The model has several instances due to a scaling parameter $n = 1..10000$, which defines the number of tokens on several initially marked places. In fact, the three marked places p_i1, *access* and $r\_stopped$ get the following number of tokens depending on n: $p\_i1 = 2 \cdot n + 1$ and $access = r\_stopped = 2 \cdot n$.

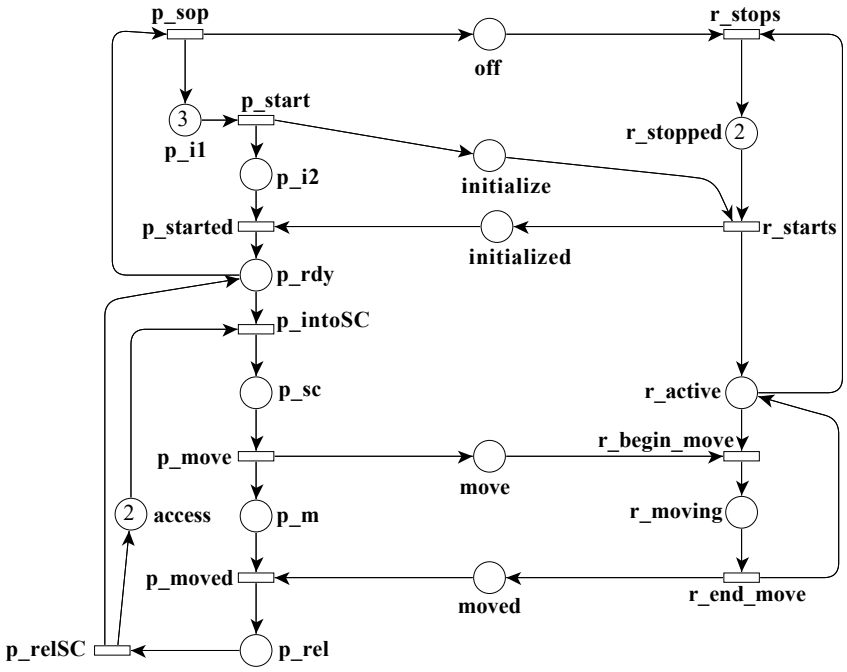


Figure 5.1: RobotManipulation model [65] with $n = 1$.

The model consists of 15 places, 11 transitions, and 34 arcs. Despite its simple structure, the size of the state space for $n = 10000$ is rather large with

$2.8 \cdot 10^{33}$ reachable markings. This happens by the large number of tokens in the model which end up on places in a huge number of combinations. This demonstrates that explicit verification of token-scaling models can become extremely complex.
However, some properties are not really dependent on the actual number of tokens. An example for this is checking whether there is a reachable marking where the number of tokens on a certain place is more or less than the number of tokens on some another place or a given constant.
So, instead of using the actual number of initial tokens, the idea is to reduce their number on places with more than one token. Afterwards, the usual model checking techniques to verify the property can be applied. If a witness path or a counterexample for the property under investigation is found in the reduced state space, then the property also holds in the original setting. If not, we simply do not know.
For the RobotManipulation model with $n = 5000$, consider specification φ which compares the token cardinality of some places:

$$\varphi = \mathbf{EF}\,(p\_rel > p\_m \wedge p\_m > p\_rdy).$$

In other words, φ asks if there is an execution where finally p_rel has more tokens then p_m and where p_m has more tokens than p_rdy. With $n = 5000$ the places *access* and $r\_stopped$ would have 10000 tokens each, and p_i1 another 10001 tokens. With other optimizations switched off, our verification tool LoLA, could not verify this property, even after evaluating over 1 billion states. Using the proposed idea from this chapter, we reduced the number of tokens to 5 on *access*, $r\_stopped$ and p_i1. With this, LoLA found a witness path of only 112 transitions, evaluating only 159 markings in total. Hence, the under-approximation approach resulted in a significant speed up. However, due to its incomplete nature the new method can only be a valuable addition to a portfolio.

5.2 The theory of under-approximation

The under-approximation we are proposing is based on a so-called *simulation relation*. We recall the classical notion of simulation [89] in order to compare the behavior of two labeled transition systems, which, in our case, are given by two P/T nets. Intuitively, an LTS P is simulated by another LTS Q if P has less or equal behavior then Q. Technically, this is expressed by a simulation relation Σ between (the states of) P and (the states of) Q satisfying a certain semantic requirement. More precisely, every transition of P has to be somehow mimicked by an equally labeled transition of Q.

Definition 5.2.1 (Simulation [89]).
Let $P = (S, A, \rightarrow, s_0)$ and $Q = (S', A', \rightarrow', s'_0)$ be two labeled transition systems. A relation $\Sigma \subseteq S \times S'$ is called simulation relation *where P is simulated by Q if and only if*

- $(s_0, s'_0) \in \Sigma$;
- *for all $(s_1, s'_1) \in \Sigma$, $a \in A$, and $s_2 \in S$ with $s_1 \xrightarrow{a} s_2$ in P, exists a state s'_2, such that $s'_1 \xrightarrow{a} s'_2$ in Q and $(s_2, s'_2) \in \Sigma$.*

If there is a simulation relation Σ between P and Q, then we also say that Q *simulates* P, Q is an *over-approximation* of P, or P is an *under-approximation* of Q.

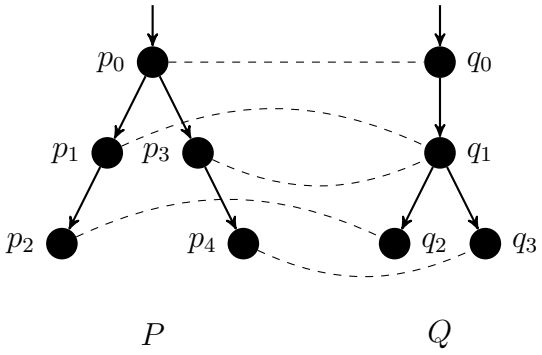


Figure 5.2: LTS Q simulates LTS P

As an example for simulation consider Figure 5.2. The figure shows two labeled transition systems P and Q. Q simulates P using the simulation relation $\Sigma = \{(p_0, q_0), (p_1, q_1), (p_3, q_1), (p_2, q_2), (p_4, q_3)\}$.

Lemma 5.2.1. *If P is simulated by Q, then the computation tree of P can be viewed as a subtree of the computation tree of Q.*

From this, we conclude the following.

Lemma 5.2.2. *Properties that hold for all branches of the computation tree of Q, hold for all branches of the computation tree of P as well.*

It follows that the simulation relation preserves certain temporal logic fragments. For the following statements, we assume that formulas are in negation normal form (see Definition 3.2.4).

Proposition 5.2.3 (Simulation preserves ACTL* [51])**.** *Let P and Q be two LTS with initial states s_0 and s'_0, respectively. If P simulates Q, then*

– $(P, s_0) \models \varphi$ *implies* $(Q, s_0') \models \varphi$, *for any ACTL* formula* φ;
– $(Q, s_0') \models \varphi$ *implies* $(P, s_0) \models \varphi$, *for any ECTL* formula* φ.

Since negations are absent, we cannot turn a universal path quantifier into an existential path quantifier or vice versa [25]. But, the negation of every ACTL* formula is an ECTL* formula and vice versa. The reason for this is that for all operators we have opportunities to drive negations from the top of the formula to the bottoms.
Simulation is used to show that some transformation of a labeled transition system preserves ACTL*. Using this, we can find a counterexample for ACTL* using an equivalent ECTL* formula.

Proposition 5.2.4 (Counterexample for ACTL*)**.** *For every ACTL* formula* φ *there exists an ECTL* formula* ψ *such that* $\neg\varphi$ *and* ψ *are equivalent CTL* formulas.*

We use this to propose our new idea for the verification of token scaling models. Places that contain more than a given $\lambda \in \mathbb{N}$ of tokens in the initial marking are simply cut to λ tokens.

Definition 5.2.2 (Reduced net N_r).
Let $N = (P, T, F, W, m_0)$ *be a P/T net,* $n \leq |P|$ *the number of initially marked places and* $I = \{p_1, \ldots, p_n\} \subseteq P$ *the set of initially marked places. For a given set of thresholds* $\{\lambda_1, \ldots, \lambda_n\}$*, a P/T net* $N^r = (P, T, F, W, m_0^r)$ *is called the* reduced net*, where the initial marking* m_0 *is substituted by the* reduced initial marking m_0^r *with*

$$m_0^r(p_i) = max\{1, min\{m_0(p_i), \lambda_i\}\}$$

for all $p_i \in I$ *and* $i \in \{1, \ldots, n\}$.

Now we have all the ingredients together to propose an under-approximation for token-scaling models.

Theorem 5.2.5 (N simulates N^r)**.** *If* N *is a P/T net, with LTS* $P = (S, A, \rightarrow, s_0)$ *and* N^r *is the corresponding reduced net, with LTS* $Q = (S^r, A^r, \rightarrow^r, s_0^r)$*, then* P *simulates* Q *and* $(Q, s_0^r) \models \varphi$ *implies* $(P, s_0) \models \varphi$*, for all ECTL* formulas* φ.

Proof. The existence of the simulation together with Proposition 5.2.3 preserves ECTL*. Since the reduced system Q is an under-approximation, it holds that the original LTS P is relative to the reduced system an over-approximation. For over-approximations, it is well known (Proposition 5.2.3) that the simulation preserves ACTL*. And with the inversion, which is the under-approximation, the simulation preserves ECTL*. □

Corollary 5.2.6. *The approach is able to verify temporal properties in labeled transition systems that have a witness path or a counterexample.*

The reason for this is that for every marking reachable in the reduced net there exists a reachable marking in the original net.

Proposition 5.2.7. *If an ECTL* formula φ is true in a reduced net N^r, then φ is also true in the original net N. If an ACTL* formula φ is false in the reduced net N^r, then φ is also false in N.*

Proof. Follows directly from Corollary 5.2.6. □

Corollary 5.2.8. *Since LTL is a subset of ACTL*, the proposition for ACTL* is also true for very LTL formula.*

For model checking, we can use this approach with standard model checking algorithms (see Chapter 4) on the reachability graph and can combine it with other reduction techniques such as partial order reduction (see Chapter 9). The method is sufficient for ECTL* and reachability, and it is necessary for ACTL* and LTL.

5.3 Heuristics

Finding the optimal threshold for the reduced initial marking of a P/T net is hard, as two objectives oppose each other. On the one hand, a small state space is desired and therefore initially marked places should have as few tokens as possible. On the other hand, less tokens also mean a lesser probability to still verify the property under investigation. It, thus, is an optimization problem to exactly find the fewest number of tokens, which are needed to produce the witness path or counterexample, respectively. Since the solution to this problem is as hard as the original issue, we propose three heuristics to get a good threshold.
In the sequel, let $N = (P, T, F, W, m_0)$ be a P/T net, $n \leq |P|$ the number of initially marked places and $I = \{p_1, \ldots, p_n\} \subseteq P$ the set of initially marked places. Further, let $N^r = (P, T, F, W, m_0^r)$ be the reduced net and $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ the set of thresholds.
The idea behind the first heuristic is to assigns each initially marked place a constant.

Definition 5.3.1 (Simple threshold heuristic).
Let $x \in \mathbb{N}$ be a constant. If all $\lambda_i = x$ with $\lambda_i \in \Lambda$ and $i \in \{1, \ldots, n\}$, then the initial marking is computed by the simple threshold heuristic.

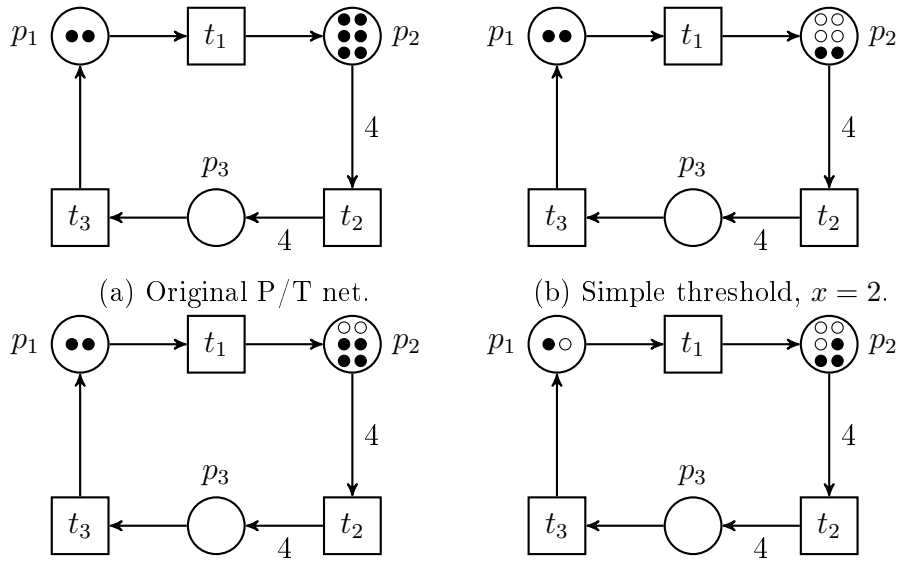
If a constant is used, then it is also obvious to use a percentage.

Definition 5.3.2 (Percentage heuristic).
Let $x \in \mathbb{N}$ be a multiplier. If all $\lambda_i = m_0(p_i)/100 \cdot x$ with $\lambda_i \in \Lambda$ and $i \in \{1, \ldots, n\}$, then the initial marking is computed by the percent heuristic.

The hypothesis for the third heuristic is that at least as many tokens are needed to satisfy the greatest "constant" of the P/T net, meaning the largest arc weight, and/or the formula, meaning the largest value of $k_1 p_1 + \cdots + k_n p_n \leq k$.

Definition 5.3.3 (Largest constant heuristic).
Let φ be a formula and $x \in \mathbb{N}$ a multiplier. Further, let W^* be the maximal arc weight *with $W^* = max\{W(f) | f \in F\}$ of N. In addition, if φ has the form $k_1 p_1 + \cdots + k_n p_n \leq k$, then K^* is the* maximal formula constant *with $K^* = max\{k_i \in \varphi | i \in \{1, \ldots, n\}\}$. Otherwise, $K^* = 0$. If all $\lambda_i = max\{W^*, K^*\} \cdot x$ with $\lambda_i \in \Lambda$ and $i \in \{1, \ldots, n\}$, then the initial marking is computed by the* largest constant heuristic.

The multiplier x is used as an additional buffer. The size of x is arbitrary and a suitable value must be determined by experiments.



(a) Original P/T net.

(b) Simple threshold, $x = 2$.

(c) Largest constant, $W^* = 4, x = 1$.

(d) Proportional, $x = 50$ (%).

Figure 5.3: Heuristics for reduced initial markings.

Figure 5.3 shows the different approaches to compute the reduced initial marking. The original P/T net can be seen in Figure 5.3a. Places p_1 and

p_2 have two respectively six tokens on them. Figure 5.3b shows the *simple threshold* approach with $x = 2$. The result is that the number of tokens on p_2 is reduced to two tokens. Place p_1 remains unchanged. The white tokens are the tokens that are removed from the original initial marking. In Figure 5.3c the *largest constant heuristic* is used. Since we do not use a formula in this example, the largest constant is based on the maximum arc weight, which is in this case four. Four is also the threshold, because the multiplier is one ($x = 1$). Only the number of tokens on p_2 is reduced to four. The *percentage heuristic*, with $x = 50$ (%), is shown in Figure 5.3d. The token count is halved on all places.

5.4 Implementation

In the implementation of the under-approximation, we need to reflect the reduced tokens also in the atomic propositions of the property under investigation. This requires to adjust the atomic propositions, too. We differentiate between two approaches based on the type of atomic proposition we want to verify.

- **Monotonous method.** If a certain atomic proposition is monotonic, meaning it compares the number of tokens to a marking in a way that we always state that we require the number of tokens to be greater or equal to a certain threshold, then we could use the original atomic proposition. Thus, the monotonous method is only applicable for positive comparisons, meaning that the places appear only with positive factors in the atomic proposition ($\forall i : k_i \cdot m(p_i)$ it holds that $k_i \geq 1$). This approach is basically valuable for `FIREABLE(`t`)` predicates which talk about the fireability of a transition t. Fireability is inherently the comparison of token counts to thresholds which are the weight of the arcs.

- **Shift method.** For all token count comparisons, we can use the shift approach. Here, we reflect the effect of the removed tokens by a modification of the atomic proposition. Take for example the atomic proposition $2 \cdot p_1 + p_2 \geq 7$ which involves place p_1 with multiplier 2. Then, three removed tokens mean that the threshold of the remaining tokens need to exceed only one, $2 \cdot p_1 + p_2 \geq 1$, in order for the original number of tokens to exceed seven.

5.5 Experimental validation

As a proof-of-concept, we implemented the under-approximation method in our explicit model checker LoLA [142]. For evaluating the method and its different heuristics, we used the benchmark provided by the MCC 2019 [68]. The benchmark consists of 94 models. Due to the scaling parameter this results in 1018 model instances. We only consider P/T nets that are not safe and where the initial marking has removable tokens. We also avoided one instance with more than 2^{32} tokens. Even though, this instance is well suited for the under-approximation method due to the enormous token count, we had to ignore it to avoid an overflow in LoLA.
As specifications, we used the reachability formulas provided in the MCC 2019, too. Although the introduced method works for ACTL*, ECTL*, and LTL formulas, we only present the experimental results for reachability queries, which have the form $\mathbf{AG}\,\varphi$ or $\mathbf{EF}\,\varphi$ and are part of ACTL* or ECTL*, respectively. The results for LTL and CTL are similar and so we simply skip them. For each instance, we have 32 reachability formulas: 16 of them are concerned with the cardinality of tokens on places, and the other 16 are concerned with the fireability of transitions. All in all, we consider 21 models resulting in 214 instances with 32 formulas for each instance. This results in a total of 6848 verification runs.
We executed the experiments on our machine Ebro, which has 32 physical cores running at 2.7 GHz and 1 TB of RAM. The operation system running on Ebro is CentOS Linux 7 (Core).
We run the new method with partial order reduction (see Chapter 9) in parallel against two other methods from the portfolio. Without partial order reduction the state spaces are simply too large. The two other methods are the regular state space search and a structural method, named counterexample guided abstraction refinement (CEGAR) [140]. Both methods are available for many years, highly optimized, and already pushed to the limits. Furthermore, for all verification runs formula simplification (see Section 8.1) was used.
From the 6848 formulas the whole portfolio failed on 122 formulas. 5152 formulas were reachable, meaning they had a definite result with a witness path or counterexample. Remember that these are formulas of the form $\mathbf{EF}\,\varphi = \texttt{TRUE}$ or $\mathbf{AG}\,\varphi = \texttt{FALSE}$. The remaining 1574 formulas were either unreachable or trivial. If a formula is unreachable, then our new approach cannot yield a result. Trivial means that we find out that the formula is reachable or unreachable in the preprocessing using formula simplification.
As seen in Table 5.1, our new method, with the simple threshold set to 5 tokens, and run in parallel to the other two methods, was able to solve 436

Heuristic	All (5152)		Failed (122)	
	#	%	#	%
Simple threshold, $x = 5$	436	8.5	0	0
Largest constant, $x = 2$	267	5.2	0	0
Percentage, $x = 33$ (%)	659	12.8	8	6.5

Table 5.1: Performance of heuristics.

(8.5 %) of the 5152 reachable queries, but it was not able to solve any of the 122 unsolved ones. For a complementary method in the portfolio 8.5 % is a solid result. Especially under the consideration that both competing methods are available for many years, highly optimized, and already pushed to the limits.
The picture stays more or less the same if we change the threshold to anything between 1 and 10. The largest constant heuristic does not perform very well with multiplier 1 or 2, although increasing the multiplier is improving the performance a little bit. For higher token counts the percentage heuristic is better suited. The percentage heuristic performs really well with x set to anything between 10 and 33. With $x = 33$, the new method could solve 659 (12.8 %) queries. Furthermore, it was even able to solve 8 (6.5 %) additional queries from the 122 unsolved ones. All in all, the experiments show that the under-approximation method is a valuable addition to a portfolio.

5.6 Discussion

Token-scaling P/T nets tend to have large state spaces, since they have a large number of tokens on the initially marked places. Their applications are widespread, as for instance in biochemical reaction chains [55, 93].
We introduced an under-approximation for token-scaling P/T nets, as a lightweight addition to existing portfolios. It can be used as a sufficient, respectively necessary quick check, depending on the verified property. More precisely, the method can be applied to temporal logic formulas that are reachable, that is, whenever a witness path or a counterexample can be found, which happens to be reachability, ACTL*, ECTL*, and LTL formulas.
The method is compatible with other reduction techniques used in explicit state space verification, such as partial order reduction [49, 97, 127], or symmetry [110]. Furthermore, the under-approximation method complements overapproximation approaches like the ones based on the state equation [140]. This is convenient since under-approximation gives a definite result in exactly the opposite cases then over-approximation.

The experiments show that running the under-approximation method in a portfolio with other model checking algorithms speeds up the verification process up to 12.8 %, which, in reverse, gives more time to other aspects of verification. But not only the runtime is reduced, but also the memory consumption. In addition to the smaller state space, fewer tokens are used in the markings, which means that storing a marking requires less memory.
All in all, we dealt with three of our research goals: we *simplified the model*, presented an *alternative reduction technique*, and *improved model checking efficiency*.
The biggest open issue is to find the number of required tokens on the initially marked places, which results in the smallest possible state space that is still able to verify the property under investigation. To tackle this problem, we introduced some heuristics that were able to increase the performance. To find better heuristics using structural information of the P/T net and also information regarding the formula is left for future work. Another question is, whether it is possible to also prove unreachable queries with a reduced number of tokens in the initial marking. A solution to this problem would make the method complete.

Chapter 6

Linear algebra for finite-single-path formulas

In the previous chapter, we under-approximated the verification of token scaling models. This chapter introduces an *over-approximation* as another *alternative reduction technique* to extend the portfolio. We have published this method in [78] and extended it in [82]. The following chapter is based on those publications.

For our over-approximation approach, we combine the concept of counterexample guided abstraction refinement (CEGAR) [23] with the Petri net state equation (see Definition 2.3.13). The latter an over-approximation of its own, uses linear algebra to find a superset of reachable states. As this contributes to state space explosion, Wimmel and Wolf proposed to additionally apply the CEGAR approach in 2012 [140]. Hajdu et al. examined the correctness and completeness of this method in 2014 [54] and extended the method in 2015 [53] to a broader class of P/T nets. The concept behind CEGAR combined with the state equation is to reduce the state exploration in the reachability graph to integer linear programming (ILP) [32]. Because ILP is NP-complete it only requires polynomial space. The procedure of the CEGAR idea is that spurious solutions of the state equation are iteratively analyzed and then constraints are added to the ILP problem to exclude spurious solutions but not real ones.

Because ILP-problems can become infeasible, the CEGAR approach is especially good for verifying that no state is reachable where a particular property holds. This makes it a valuable complement to explicit model checking algorithms which are in general good for verifying reachable states that satisfy a certain property due to the on-the-fly effect. Wimmel and Wolf proved the success of this approach with their proof-of-concept tool named *Sara* [140]. Up until now, this approach was only able to solve reachability queries of the

form $\mathbf{EF}\,\varphi$ and $\mathbf{AG}\,\varphi$. In this chapter, we propose techniques to extend this approach to several other temporal logic formulas. To the best of our knowledge, this is the first time that this approach is used to verify temporal logic formulas other than reachability. To this end, we propose two techniques to solve queries of the form $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$ and $(\mathbf{EX})^k\varphi$ with the CEGAR approach for P/T nets [78]. Using well known tautologies, $\mathbf{A}\,(\varphi\,\mathbf{R}\,\psi)$ and $(\mathbf{AX})^k\varphi$ are solvable with these techniques as well.
Using our explicit model checker LoLA, we found out in [79] that only 62.3 % of the $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)/\mathbf{A}\,(\varphi\,\mathbf{R}\,\psi)$ formulas from the MCC 2018 [66] were solved. The reason for this is that the on-the-fly effect has no or only a very limited effect for some properties, e.g., if the property φ and $\neg\psi$ hold in the entire state space, then the search for $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$ needs to explore the entire state space. For this particular property, the CEGAR approach will terminate very quickly with a negative result since the ILP-problem is infeasible. That is the reason that the CEGAR approach is generally good for verifying unreachable properties.
We use specialized routines for $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$ and $(\mathbf{EX})^k\varphi$ as building bricks to solve a larger class of CTL formulas, namely the *finite-single-path* CTL formulas [82]. This class is characterized by two conditions: First, these formulas end in a final marking, meaning that no cycles are involved, hence, they are finite, and secondly, they have a linear witness. Using tautologies, we can again check both the existential and the universal finite-single-path formulas. We are not able to solve formulas of the form $\mathbf{EG}\,\varphi$ with our approach, since the verification of $\mathbf{EG}\,\varphi$ involves cycles. However, we propose several sufficient and necessary quick checks to verify such formulas under certain conditions.
In addition, to support different verification tasks, we present in this chapter a set of necessary or sufficient "quick checks" for a whole range of CTL formulas. The quick checks are based on the introduced CEGAR approaches. Thus, the quick checks can be run in parallel to the actual verification algorithm.
The biggest drawdown is that we can not provide a proof-of-concept. The reason for this is that there is not enough time available for the considerable effort that an implementation would cost. To implement our extensions, we first have to fully integrate Sara into LoLA which means that we have to convert Sara's data structures to LoLA's and then we can integrate our extensions. We want to avoid extending Sara directly since we want to combine other reduction techniques from LoLA with this approach, e.g., partial order reduction (see Chapter 9).
The rest of this chapter is organized as follows. For self-containedness, we give a brief overview of the CEGAR approach for the Petri net state equation in Section 6.1. Section 6.2 introduces additional basic concepts that we

use in the rest of this chapter. We continue in Section 6.3 and Section 6.4 with the introduction of the specialized routines for $\mathbf{E}(\varphi \mathbf{U} \psi)$ and $(\mathbf{EX})^k \varphi$, respectively. Section 6.5 proposes a method to solve finite-single-path CTL formulas using CEGAR. Section 6.6 is concerned with quick checks for $\mathbf{EG}\,\varphi$. Section 6.7 introduces necessary and sufficient quick checks for a whole range of CTL formulas. We conclude this chapter in Section 6.8 with a discussion.

6.1 CEGAR for reachability analysis

This section briefly summarizes the CEGAR approach for the Petri net state equation given by Wimmel and Wolf [140], as well as the refined one by Hajdu et al. [53, 54].
The idea of abstraction is to omit irrelevant details of systems behavior to simplify analysis and verification. But if it is too coarse, then verification might fail, and if it is too fine, then state space explosion may occur. One way to a middle ground is to start with an over-approximation and then to iteratively refine the abstraction on spurious counterexamples. This is just the way of CEGAR as illustrated in Figure 6.1 and explained in the following paragraphs.

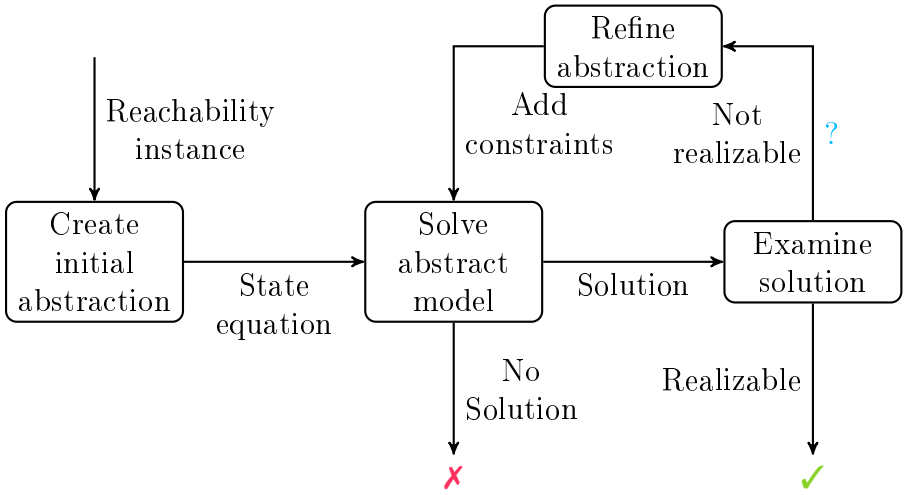


Figure 6.1: CEGAR for reachability analysis.

Create initial abstraction

In the remainder, let $N = (P, T, F, W, m)$ be a P/T net and $m' \in R_N(m)$ be an instance of the reachability problem transformed into a state equation

of the form $m + C \cdot \wp(w) = m'$. Without additional constraints, this is an over-approximation because every solution to the instance m' satisfies the equation, but not vice versa. Here, it serves as the initial abstraction for the reachability problem.

Solve abstract model

Solving the state equation is an ILP-problem. The objective function f for the ILP-problem minimizes the length of the firing sequence that leads from the initial marking m to the final marking m'. This is described by the Parikh vector $f(w) = \sum_{t \in T} |\wp(w)(t)|$. A solution $\wp(w)$ of the equation is called *realizable*, if there is a corresponding firing sequence that is executable in N. The feasibility of the ILP-problem is a necessary condition for reachability, but not a sufficient one. We can distinguish between three different situations:

- If the ILP-problem is *infeasible*, then the necessary condition is violated and the final marking is not reachable.
- If the ILP-problem has a *realizable solution*, then the sufficient condition is satisfied and the final marking is reachable.
- If the ILP-problem has an *unrealizable solution*, which can be considered a counterexample, then the abstraction has to be refined.

Infeasibility is the reason for the particularity good performance for negative results, because often the initial abstraction is enough to prove that a final marking is not reachable.

Examine solution

A solution of the state equation is a vector $\wp(w) \in \mathbb{N}^{|T|}$. Remember that every transition $t \in T$ fires $|\wp(w)(t)|$ times in any sequence from m to m' that is described by $\wp(w)$. This leaves us with the problem to restore w. We use the notation of the solution space of the state equation as follows.

Corollary 6.1.1 (Solution space [140]). *For a given state equation $m + C \cdot \wp(w) = m'$ of a P/T net $N = (P, T, F, W, m)$, a set of vectors $B = \{b_i \in \mathbb{N}^{|T|} \mid 1 \leq i \leq j\}, j \in \mathbb{N}$ is called* base vectors *if all $b_i \in B$ are pairwise incomparable (by standard componentwise comparison for vectors) and a set of vectors $P = \{p_i \in \mathbb{N}^{|T|} \mid 1 \leq i \leq k\}, k \in \mathbb{N}$ is called* period vectors *if P forms a basis for the non-negative* solution space $P^* = \{\sum_{i=1}^{k} n_i p_i \mid n_i \in \mathbb{N}, p_i \in P\}$ *of $C \cdot \wp(w) = 0$ such that:*

– *for all solutions* $\wp(w)$ *there are* $n_i \in \mathbb{N}$ *for* $1 \leq i \leq k$ *and* $n \in \{1, \ldots, j\}$ *such that* $\wp(w) = b_n + \sum_{i=1}^{k} n_i p_i$*;*

– *for every solution* $\wp(w)$*, all vectors of the set* $\wp(w) + P^*$ *are solutions as well.*

Because the state equation is an over-approximation, we have to examine whether a solution is a realizable solution or a spurious one. The problem is that a solution does not provide any information regarding the order in which the transitions have to fire. Furthermore, no information about the enabledness of transitions is provided. We explore, therefore, a reduced state space for a sequence w where every transition t can fire at most $|\wp(w)(t)|$ times. If the target marking m' is reachable in this constraint state space, then the solution is realizable, which is a sufficient proof for reachability. Otherwise, the solution is unrealizable. That is, there exists at least one $t \in T$ which has fired less than $|\wp(w)(t)|$ times. Thus, the examined solution is a counterexample and we have to refine the abstraction.

For example, assume that for a reachability query in the P/T net depicted in Figure 6.2 the solution vector is $\wp(w) = (0, 1, 1, 1)$ to reach the final marking $m_f = (p_6)$. This means, to reach m_f, transitions t_2, t_3, and t_4 have to fire once in a sequence w not specified by $\wp(w)$. The only transition that can fire in the initial marking is t_2. Beginning at the initial marking $m = (p_1, p_3)$, we search the reduced the state space where exactly t_2, t_3, t_4 can fire at most once. However, t_3 and t_4 are not enabled, neither in m nor after firing t_2. Thus, the solution $\wp(w) = (0, 1, 1, 1)$ is not realizable. A possible solution to reach m_f would be to fire t_2 and afterwards, transfer or borrow tokens to fill the scapegoat place p_2. This would enable t_3, which in turn would enable t_4.

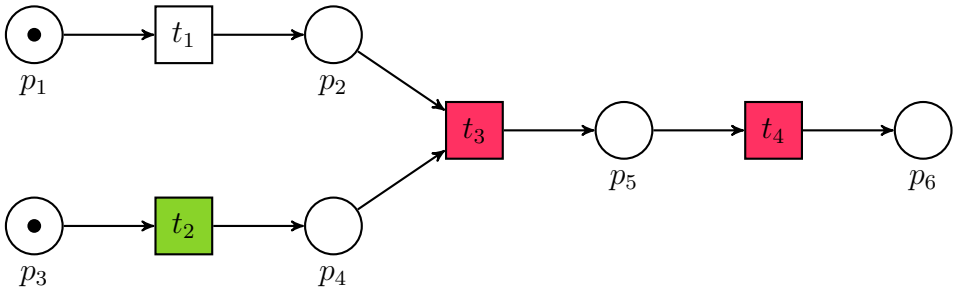


Figure 6.2: Unrealizable solution $\wp(w) = (0, 1, 1, 1)$.

Refine abstraction

To avoid spurious solutions the ILP-problem needs to be solved by a different solution. To enforce the computation of a different solution, a refined abstraction is built. We define two types of constraints, both being linear inequalities over transitions.

Definition 6.1.1 (Jump constraints [53, 140]).
Jump constraints have the form $|t| < n$ *with* $n \in \mathbb{N}$ *and* $t \in T$.

The firing count of transition t is represented by $|t|$. Using the fact that base solutions are pairwise incomparable, jump constraints intend to generate a new base solution.

Definition 6.1.2 (Increment constraints [53, 140]).
Increment constraints have the form $\sum_{i=1}^{k} n_i|t_i| \geq n$, *where* $n_i \in \mathbb{Z}$, $n \in \mathbb{N}$, *and* $t_i \in T$.

Increment constraints are used to get a new non-base (non-minimal) solution, i.e., T-invariants (see Definition 2.3.15) are added, since their interleaving with another sequence w' may turn w from unrealizable to realizable.

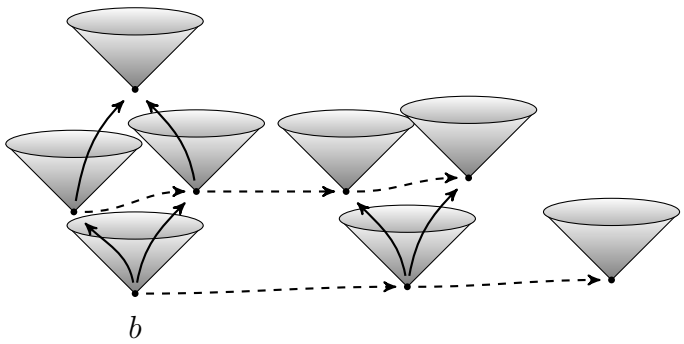


Figure 6.3: Solution space (the graphic is from [140]).

Running the CEGAR approach

By adding the two types of constrains to existing solutions, we can control traversal through the solution space. Every new solution is then checked whether it is realizable, or if the ILP-problem becomes infeasible. Figure 6.3, shows the solution space. From the initial solution b there are paths to any solution which are shown as black dots. The cones represent the linear solution spaces from a given solution. Solution spaces can intersect each other

or even include each other. Increment constraints, which add T-invariants to a solution, are shown by a normal arrow. Jump constraints, represented by a dashed arrow line, ensure jumps to an incomparable greater solution.
In the example from Figure 6.2, the solution vector $\wp(w) = (0, 1, 1, 1)$ needs tokens on place p_2. The only way to do this, is to fire t_1. Therefore, the increment constrain $|t_1| > 0$ is added. With this, a new minimal solution $\wp(w) = (1, 1, 1, 1)$ is found, which is realizable. The firing of t_1 in the initial marking and then t_2 enables t_3 which in turn enables t_4.
The idea is to transfer or borrow tokens from other places. In the example above, this was done with t_1 which could provide extra tokens. Another approach would be to add T-invariants to the solution. T-invariants have the property that, after fired completely, they are back in the same marking. Thus, the marking before and after firing it is the same one.

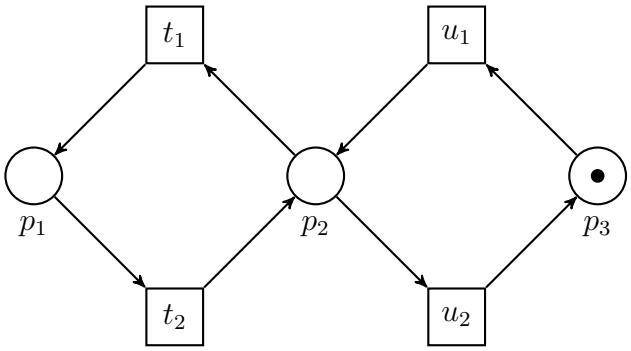


Figure 6.4: Borrowing tokens via T-invariants.

As an example consider Figure 6.4. The T-invariant t_1t_2 is not able to fire on its own. But it can borrow a token from an intermediate marking of the T-invariant u_1u_2. The firing sequence $u_1t_1t_2u_2$ leads to the same marking as firing t_1t_2. This shows that we can add T-invariants to an unrealizable solution to transform the solution to a realizable one.
The concept of *partial solutions* takes care of the organization of the different solutions provided by the ILP solver.

Definition 6.1.3 (Partial solutions [140]).
For a P/T net $N = (P, T, F, W, m)$ *and a reachability instance* $m' \in R_N(m)$ *a tuple* $ps = (\Gamma, \wp(w), \sigma, r)$ *is a* partial solution *to an ILP instance if*

- Γ *is a* set of jump and increment constraints*;*
- $\wp(w)$ *is a* minimal solution *fulfilling the ILP-problem;*
- $\sigma \in T^*$ *is a* maximal firing sequence*, with* $m \xrightarrow{\sigma}$ *and* $\wp(\sigma) \leq \wp(w)$*;*

– r *is the* remainder $r = \wp(w) - \wp(\sigma)$ *such that* $\forall t \in T : r(t) > 0 \Longrightarrow \neg m \xrightarrow{\sigma t}$.

The set of jump and increment constraints, together with the state equation defines the ILP-problem. Partial solutions are produced during the examination of a solution $\wp(w)$ by exploring the reduced state space of N. For the exploration, a standard model checking algorithm can be used to build a tree of reachable markings, such that for all transitions $t \in T$ it holds that they only occur $|\wp(w)(t)|$ times. Each path to a leaf represents a maximal firing sequence of a new partial solution. A maximal firing sequence means that no $t \in r(t)$ can fire anymore. Partial solutions with an empty remainder $r = 0$, are full solutions and they satisfy the reachability problem. If no full solution exists, then $\wp(w)$ might be realizable by another firing sequence σ'. This means, either the addition of a jump constraint to get to a new base solution, or the addition of an increment constraint to get additional tokens for transitions with $r(t) > 0$. If all possible partial solutions are explored, and no full solution is found, then the reachability problem can not be satisfied.

Theorem 6.1.2 (Reachability of solutions [53,140])**.** *If the reachability problem has a solution, a realizable solution of the state equation can be reached by repeatedly expanding the minimal solution with jump and increment constraints.*

One drawback is that the termination of the introduced approach is not guaranteed. Thus, the procedure is incomplete [140]. This drawback can be handled with a portfolio where traditional algorithms are combined with CEGAR. The method described in this section is subsequently called the *state equation approach.*
There are some more details involved in the CEGAR approach, such as *generating constraints*, *better intermediate markings*, *optimizations*, and *distant T-invariants*. More information regarding these concepts can be found in [53, 54, 140]. For the extensions we are introducing in the next sections, these details are not relevant.

6.2 Basics

This section formalizes the notion of *increasing* and *decreasing transitions* of a P/T net. A formal sum $s = k_1x_1 + \ldots + k_nx_n$ is an atomic proposition of P/T nets. Every marking m is associated with the integer $v_s(m) = k_1m(p_1) + \ldots + k_nm(p_n)$. The firing rule of P/T nets immediately leads to:

Definition 6.2.1 (Delta of a transition).
Let $N = (P, T, F, W, m)$ be a P/T net and C the corresponding incidence matrix. If s is a formal sum and $t \in T$ a transition, then $\Delta_{t,s}$ is defined as $\Delta_{t,s} = k_1 C(p_1, t) + \ldots + k_n C(p_n, t)$, with $p_1, \ldots, p_n \in P$ and $k_1, \ldots, k_n \in \mathbb{Z}$.

Lemma 6.2.1. *For all markings m where $m \xrightarrow{t} m'$ implies $v_s(m) + \Delta_{t,s} = v_s(m')$.*

Proof. Follows directly from the Petri net state equation (Definition 2.3.13). □

As we assume the transition system to be finite, there is only a finite range of values that $v_s(m)$ can take. We call an integer number k a *lower bound* for formal sum s if, for any reachable marking m, $v_s(m) \geq k$, and *upper bound* for s if, for any reachable m, $v_s(m) \leq k$. There exist several approaches in Petri net theory for computing bounds. As an example, we can solve the following optimization problem: s is the objective function (to be minimized or maximized) and the state equation serves as a side condition. If the problem yields a solution with non-diverging value for the objective function, then that value is a lower respectively an upper bound for s.
Based on Lemma 6.2.1, we can identify increasing and decreasing transitions.

Definition 6.2.2 (Increasing, decreasing).
Let $N = (P, T, F, W, m)$ be a P/T net and $s \leq k$ an atomic proposition, where s is a formal sum and k is an integer. Further, let L be a lower bound and U an upper bound for s. We call transition $t \in T$ with respect to the formal sum s:

1. weakly increasing *if and only if* $\Delta_{t,s} < 0$,
2. weakly decreasing *if and only if* $\Delta_{t,s} > 0$,
3. strongly increasing *if and only if there is an upper bound U for s where* $\Delta_{t,s} \leq k - U$, *or*
4. strongly decreasing *if and only if there is a lower bound L for s where* $\Delta_{t,s} > k - L$.

Increasing transitions have the tendency to turn a false proposition into a true one, when they fire and decreasing transitions often turn a true proposition into a false one. For example, let $p \leq 0$ be an atomic proposition where p is the number of tokens on place p in a P/T net. Then all transitions in the preset of p are strongly decreasing. As another example, consider the P/T net in Figure 6.5 and the formula $\varphi = \mathbf{EF}\,(2 \cdot p_2 + 4 \cdot p_3 \leq 5)$. It is easy to see that $m_0 \not\models \varphi$. The increasing transition t_3 with $\Delta_{t_3,\varphi} = -4$ turns m_0

into $m' = (1, 0, 1, 1)$ which satisfies φ with $(2 \cdot 0 + 4 \cdot 1 \leq 5)$. On the other hand, the decreasing transition $t_1$with $\Delta_{t_1,\varphi} = 2$ does not lead to a satisfying marking.

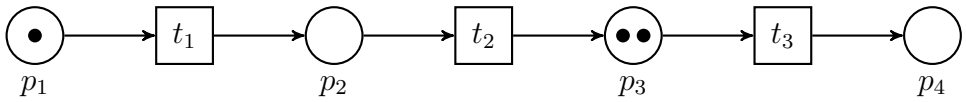


Figure 6.5: Example for $\Delta_{t,\varphi}$.

The tendency from above is formally supported by:

Lemma 6.2.2. *Let $N = (P, T, F, W, m_0)$ be a P/T net with reachability graph R_N. Further, let m, m' be markings, $t \in T$ a transition with $m \xrightarrow{t} m'$, and $s \leq k$ an atomic proposition with formal sum s and integer k.*

1. *If $s \leq k$ is false in m and true in m' then t is weakly increasing w.r.t. s.*
2. *If $s \leq k$ is true in m and false in m' then t is weakly decreasing w.r.t. s.*
3. *If t is strongly increasing w.r.t. $s \leq k$ then $s \leq k$ is true in m'.*
4. *If t is strongly decreasing w.r.t. $s \leq k$ then $s \leq k$ is false in m'.*

Proof. Regarding 1, we have $v_s(m) > k$ and $v_s(m') \leq k$. By Lemma 6.2.1, we conclude $\Delta_{t,s} < 0$. Regarding 3, we have $v_s(m) \geq L$ (since L is a lower bound). Hence, $v_s(m') = v_s(m) + \Delta_{t,s} \leq L + \Delta_{t,s}$ and, according to Definition 6.2.2, $v_s(m') \leq k$. Cases 2 and 4 are analog. □

6.3 Solving $(\mathbf{EX})^k\varphi$

The CEGAR approach has been proven able to solve formulas of the form $\mathbf{EF}\,\varphi$ and $\mathbf{AG}\,\varphi$ [140]. This section extends the abilities of CEGAR to solve $(\mathbf{EX})^k\varphi$ formulas. Let us start with a formal definition of $(\mathbf{EX})^k\varphi$.

Definition 6.3.1 ($(\mathbf{EX})^k\varphi$).
If $N = (P, T, F, W, m)$ is a P/T net, φ is an atomic proposition, and $k \in \mathbb{N}_1$ then $m \models (\mathbf{EX})^k\varphi \iff \exists w \in T^k : m \xrightarrow{w} m_k \wedge m_k \models \varphi$.

This means, there exists a path $m \xrightarrow{w} m_k$ with $|w| = k$ transitions in it and $m_k \models \varphi$. For example, if $k = 2$, then $(\mathbf{EX})^2\varphi = \mathbf{EX}\,\mathbf{EX}\,\varphi \iff \exists t_1 t_2 \in T^2 : m \xrightarrow{t_1 t_2} m_2 \wedge m_2 \models \varphi$. The idea for $(\mathbf{EX})^k\varphi$ is to solve $\mathbf{EF}\,\varphi$ with an additional *length constraint*. The length constraint ensures that the length of the solution $|\wp(w)|$ to the ILP-problem is equal to k.

Definition 6.3.2 (Length constraint).
Given a P/T net $N = (P, T, F, W, m)$ with state equation $m + C \cdot \wp(w) = m'$ and $k \in \mathbb{N}_1$, we call $\sum_{t \in T} |\wp(w)(t)| = k$ a length constraint.

The sum of the number of occurrences of all transitions in the solution vector $\wp(w)$ must be exactly k.

Theorem 6.3.1. *Let $N = (P, T, F, W, m)$ be a P/T net and $(\mathbf{EX})^k \varphi$ a formula, where φ is an atomic proposition, and $k \in \mathbb{N}_1$.*
If $(\mathbf{EX})^k \varphi$ has a realizable solution, it can be reached by solving $\mathbf{EF}\,\varphi$ with the CEGAR approach from [140] with an additional length constraint *in the initial abstraction.*

Proof. Based on Definition 6.3.1, $m \models (\mathbf{EX})^k \varphi \iff \exists w \in T^k \wedge m \xrightarrow{w} m' \wedge m' \models \varphi$. The length constraint $\sum_{t \in T} |\wp(w)(t)| = k$ from Definition 6.3.2 ensures that only solutions $\wp(w)$ of the ILP-problem are found, such that the length of the firing sequence w is exactly k, and results in the final marking $m_k \models \varphi$. □

6.4 Solving E (φ U ψ)

This section extends CEGAR to solve $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$.

Definition 6.4.1 ($\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$).
For a P/T net $N = (P, T, F, W, m)$ and two atomic propositions φ, ψ, $m \models \mathbf{E}\,(\varphi\,\mathbf{U}\,\psi) \iff \exists w \in T^* : m \xrightarrow{w} m'$, with $\exists i \in \mathbb{N}\ \forall j < i : (m_j \models \varphi) \wedge (m_i \models \psi)$.

This means, in every state along a path w, φ is true until a state is reached where ψ is true. It is well known that $\mathbf{EF}\,\psi$ can be rewritten as $\mathbf{E}\,(\mathtt{TRUE}\,\mathbf{U}\,\psi)$. To solve $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$, we solve $\mathbf{EF}\,\psi$ with CEGAR and, along the witness path for $\mathbf{EF}\,\psi$, we need to keep φ true. To fulfill these objectives, we introduce so-called *balance constraints*. In addition to this, we cut off exploration paths in states where both φ and ψ are false.

Definition 6.4.2 (Balance constraints).
Given is a P/T net $N = (P, T, F, W, m)$, two atomic propositions ψ and $\varphi = s_0 \leq k_0 \wedge s_1 \leq k_1 \wedge \ldots \wedge s_n \leq k_n$, with $n, k_1, \ldots, k_n \in N$, formal sums $s_1, \ldots, s_n$, and $T_i = \{t \in T | \Delta_{t,s_i} \neq 0\}$ a the set of transitions that can change the value of s_i. Furthermore, the set $T_{i,\psi}$ is defined as $T_{i,\psi} = \{t \in T_i | \Delta_{t,s_i} > 0 \wedge \Delta_{t,\psi} < 0\} \subseteq T_i$ and contains the set of decreasing transitions with respect

to s_i, which are at the same time also increasing transitions with respect to ψ. For every i. let $\delta_i = \begin{cases} 0, & \text{if } T_{i,\psi} = \emptyset \\ max\{\Delta_{t,s_i} | t \in T_{i,\psi}\}, & \text{otherwise.} \end{cases}$
Furthermore, let $\theta_i = k_i - v_{s_i}(m)$ be the number of tokens that can be consumed from the initial marking while leaving the truth value of $s_i \leq k_i$ unchanged. We call $\sum_{t \in T_i} \Delta_{t,s_i} \leq \theta_i + \delta_i$ balance constraint *with respect to s_i and m.*

T_i contains all weakly/strongly increasing/decreasing transitions with respect to s_i. In the remainder, we call θ_i the offset.

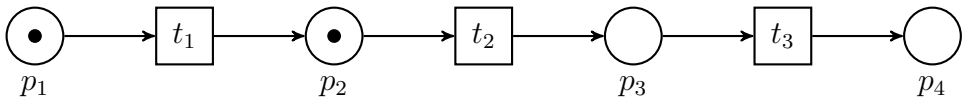


Figure 6.6: Example for balance constraints.

As an example for balance constraints, consider Figure 6.6 and the formula $\varphi = \mathbf{E}\,((p_2 > 0)\,\mathbf{U}\,(p_4 > 0))$. Note that φ and every other formula of this form can be rewritten into the required $s \leq k$ format: $\mathbf{E}\,((-p_2 \leq -1)\,\mathbf{U}\,(-p_4 \leq -1))$. To satisfy φ, we check $\mathbf{EF}\,(p_4 > 0)$, while keeping $p_2 > 0$ true along the path. The minimal solution to the ILP-problem would be (t_2, t_3). The sequence $m \xrightarrow{t_2 t_3} m'$ leads to m' which satisfies $p_4 > 0$. But after firing transition t_2 which is weakly decreasing with respect to $p_2 > 0$, a marking (p_1, p_3) is reached, that does neither satisfy $p_4 > 0$ nor $p_2 > 0$. Hence, to avoid this spurious solution, we add a balance constraint to the ILP-problem. The new solution firstly fires the weakly increasing transition t_1. The sequence $m \xrightarrow{t_1 t_2 t_3} m'$ now keeps $p_2 > 0$ true along the path and results in marking $m' = (p_2, p_4) \models (p_4 > 0)$.
Balance constraints in general ensure that the sum of all increasing and decreasing transitions with respect to a formal sum is smaller than the offset. This is based on the initial marking and the maximal arc weight of all transitions $t \in T_{i,\psi}$. The idea behind the offset is to make sure that decreasing transitions can fire without violating a necessary property, especially when they attack the support of increasing transitions. As soon as the offset θ_i becomes negative, φ will definitely be violated and thus, $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$ takes the value of ψ. We can detect this case already in the initial marking, before we compute the balance constraints and then, we can return with a definitive answer directly. The only transitions which are allowed to violate φ are those in $T_{i,\psi}$ because they also have the effect of turning ψ true. Due to this, such transitions tend to occur at the end of firing sequences. That adding balance

constraints to the initial abstraction and running CEGAR for $\mathbf{EF}\,\psi$ is correct for $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$ is stated subsequently.

Lemma 6.4.1. *Let $N = (P, T, F, W, m)$ be a P/T net, ψ and $\varphi = s_0 \leq k_0 \wedge s_1 \leq k_1 \wedge \ldots \wedge s_n \leq k_n$ two atomic propositions with $n, k_1, \ldots, k_n \in N$, $s_1, \ldots, s_n$ formal sums and it holds that $m \models \varphi$.*
If all balance constraints for φ are added to the ILP-problem for $\mathbf{EF}\,\psi$ and if $\theta_i \geq 0$ holds, then it is guaranteed that

1. *after executing the entire firing sequence, which is given as a solution to the ILP-problem, that ψ is true;*
2. *a firing sequence w exists;*
3. *if w is fired in the right order that φ is true along the path*
4. *the proposition φ can only be violated, if at all, in the final marking, but there ψ holds.*

Proof. Regarding the 3 claim, we know, based on Definition 6.2.2, that only increasing/decreasing transitions affect $s_i \leq k_i$. The offset θ_i ensures that the truth value of $s_i \leq k_i$ stays unchanged. The balance constraint makes sure that φ is not violated minus the δ_i-offset, which ensures the possibility of a firing sequence which does not violate φ along the path, until ψ holds.
If the set $T_{i,\psi}$ is not empty, the δ_i-offset based on the maximum of Δ_{t,s_i} ensures that transitions are not ignored in the balance constraint that violate φ but also turn ψ to true. The additional offset, which is the maximal arc weight of the transitions in the set, is enough to make sure that only one transition is allowed to fire with the effect of making φ false and ψ true. We use the maximum because an arc weight which is not the maximum will have a smaller effect and will not change the outcome. Transitions from the set $T_{i,\psi}$ can also fire, if they are fired in a different context, i.e., when they do not turn φ to false.
And finally, Theorem 6.1.2 ensures that if the complete sequence w of solution $\wp(w)$ is fired, we get to the final marking m' which satisfies ψ. □

The final marking m' is reached after firing the entire solution $\wp(w)$. Applying Lemma 6.4.1 only ensures that m' satisfies ψ, but it does not guarantee that intermediate markings satisfy φ. The reason for this is that also decreasing transitions with respect to φ are allowed to fire and can turn φ to false. This is not a problem if some increasing transitions are fired first to balance out this effect. Consequently, in the exploration of a realizable solution, paths must be cut off that are not balanced out.

Lemma 6.4.2. *In the exploration of the solution space, if all sequence w are* cut off *in markings m with $m \models \neg\varphi \wedge \neg\psi$, then only partial solutions remain that can become full solutions.*

Proof. Based on Definition 6.4.1, marking $m \models \neg\varphi \wedge \neg\psi$ violates the formula $\mathbf{E}(\varphi \,\mathbf{U}\, \psi)$. All paths extending m also violating $\mathbf{E}(\varphi \,\mathbf{U}\, \psi)$ and no extension of the path can turn the property to true. □

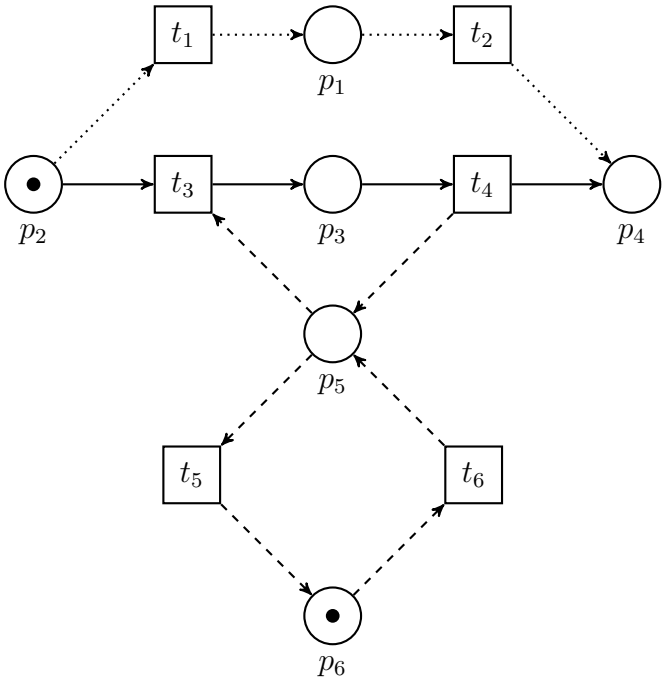


Figure 6.7: Example for cut off.

As an example for such a cut off, consider the P/T net in Figure 6.7 and the formula $\mathbf{E}((p_2 + p_3 > 0)\,\mathbf{U}\,(p_4 > 0))$. The minimal solution of the ILP-problem is (t_1, t_2) (dotted lines). After firing t_1, marking $m' = (p_1, p_6)$ is reached which violates $(p_2 + p_3 > 0)$ as well as $p_4 > 0$. Applying Lemma 6.4.2, the solution is cut off at m', because both propositions are violated. There are also no increasing transitions that can be added to this solution. Hence, we have to jump to a new base solution, which is (t_3, t_4) (solid lines). But this solution is only a partial solution, since neither t_3 nor t_4 can fire and both are part of the remainder. At this point, CEGAR would add the T-invariant (t_5, t_6) (dashed lines) to the solution. The T-invariant is able to "borrow" tokens for the transitions in the remainder. With the additional T-invariant, we have a full solution and we get the path $m \xrightarrow{t_6 t_3 t_4 t_5} m'$ which

keeps ($p_2 + p_3 > 0$) true along the path until $p_4 > 0$ holds. Now we have everything together to introduce the main theorem.

Theorem 6.4.3. *Let $N = (P, T, F, W, m)$ be a P/T net, ψ and $\varphi = s_0 \leq k_0 \wedge s_1 \leq k_1 \wedge \ldots \wedge s_n \leq k_n$ two atomic propositions with $n, k_1, \ldots, k_n \in N$, $s_1, \ldots, s_n$ formal sums and it holds that $m \models \varphi$.*
If $\mathbf{E}(\varphi \,\mathbf{U}\, \psi)$ has a realizable solution, it can be reached by solving $\mathbf{EF}\,\psi$ with the CEGAR approach, where balance constraints *are added to the initial abstraction for all s_i, and all paths are* cut off *whenever $m^* \models \neg\varphi \wedge \neg\psi$ is reached.*

Proof. Theorem 6.1.2 proofs $\mathbf{EF}\,\psi$. We repeatedly add jump and increment constraints to get to a full solution, such that the final marking m' of this solution satisfies ψ. Applying Lemma 6.4.1, we only get solutions, such that after firing the complete solution $\wp(w)$, ψ holds. Applying Lemma 6.4.1, we know that a firing sequence w exists and if w is fired in the right order, it is ensured together with the use of Lemma 6.4.2 that φ is true along the path and is only violated, if at all, in the final marking, where ψ holds. □

6.5 Solving finite-single-path CTL formulas

Theorems 6.1.2, 6.3.1, and 6.4.3 show approaches to solve the simple and frequently occurring formulas types $\mathbf{EF}\,\varphi$, $\mathbf{EX}\,\varphi$, and $\mathbf{E}(\varphi \,\mathbf{U}\, \psi)$. With the following tautologies $\mathbf{AX}\,\varphi$, $\mathbf{AG}\,\varphi$, and $\mathbf{A}(\varphi \,\mathbf{R}\, \psi)$ become solvable, too, because instead of a witness path, we search for a counterexample.

$$\begin{aligned}
\mathbf{AX}\,\varphi &= \neg\mathbf{EX}(\neg\varphi);\\
\mathbf{AG}\,\varphi &= \neg\mathbf{EF}(\neg\varphi);\\
\mathbf{A}(\varphi_1 \,\mathbf{R}\, \varphi_2) &= \neg\mathbf{E}(\neg\varphi_1 \,\mathbf{U}\, \neg\varphi_2).
\end{aligned}$$

Thus, six out of ten basic CTL operators (see Section 3.4) are solvable with these theorems. Having the formulas with the universal path quantifier, we also gain more possibilities for LTL formulas, which can now be solved with the same approaches. All these formula types are solved by a single, finite witness path (counterexample).
In the following section, we use these formulas to solve a larger subclass of CTL with CEGAR. This class is therefore called *finite-single-path formulas*. We have to differentiate between existential and universal finite-single-path formulas. Both are defined inductively.

Definition 6.5.1 (Existential finite-single-path formulas).
If φ and ψ are existential finite-single-path formulas *and α is an atomic*

proposition, then the following formulas are existential finite-single-path formulas*:*

- α *(the base of the inductive definition);*
- $\mathbf{EF}\,\varphi$*;*
- $\mathbf{EX}\,\varphi$*;*
- $\mathbf{E}\,(\alpha\,\mathbf{U}\,\varphi)$*;*
- $\varphi \vee \psi$*;*
- $\varphi \wedge \alpha$*.*

The existentially quantified formulas are paired with universally quantified formulas (see Section 3.4). These two types of formulas can be reduced to each other by negation. Hence, they permit the application of the same verification methods. Instead of a witness, the class of universal finite-single-path formulas is characterized by a counterexample and is defined accordingly:

Definition 6.5.2 (Universal finite-single-path formulas).
If φ *and* ψ *are* universal finite-single-path formulas *and* α *is an atomic proposition, then the following formulas are* universal finite-single-path formulas*:*

- α *(the base of the inductive definition);*
- $\mathbf{AG}\,\varphi$*;*
- $\mathbf{AX}\,\varphi$*;*
- $\mathbf{A}\,(\alpha\,\mathbf{R}\,\varphi)$*;*
- $\varphi \wedge \psi$*;*
- $\varphi \vee \alpha$*.*

Lemma 6.5.1. *The negation of an existential finite-single-path formula is a universal finite-single-path formula and vice versa.*

Therefore, we may restrict subsequent considerations to existential finite-single-path formulas. Universal finite-single-path formulas can be verified by checking the corresponding negation.

Definition 6.5.3 (Initial ILP-problem for existential finite-single-path formulas).
Let $N = (P, T, F, W, m)$ *be a P/T net with incidence matrix* C *and* φ *an existential finite-single-path formula that contains* $i \in \mathbb{N}$ *CTL operators.*
We call the following an ILP-problem for an existential finite-single-path formula (ILP_φ for short). *For all CTL operators in* φ *add a new set of*

variables for the Parikh vector $\wp(w)_i$ and the product of $C \cdot \wp(w)_i$ to the state equation:

$$m + C \cdot \wp(w)_1 + \ldots + C \cdot \wp(w)_i = m'.$$

Furthermore, add for all EU-operators balance constraints and for all EX-operators length constraints based on their corresponding variables.

Lemma 6.5.2. *If for a P/T net $N = (P, T, F, W, m)$ and existential finite-single-path formula φ the ILP_φ is build based on Definition 6.5.3, then it is guaranteed that a solution to ILP_φ, if fired in the right order, fulfills φ. If no such solution exists then $m \not\models \varphi$.*

Once we build the initial ILP-problem, we use CEGAR and add jump and increment constraints until we either find a realizable solution or the ILP-problem becomes infeasible. While trying to realize the solution, it is important to first use all transitions from the first Parikh vector $\wp(w)_1$ to keep the structure of the formula in place. If all transitions from $\wp(w)_1$ are used in the realization, then we can start trying to realize the rest of the solution with the transitions from $\wp(w)_2$. For our example this means, $\wp(w)_1$ keeps α_1 true until α_2 is reached.

Definition 6.5.4 (Realization ordering).
Let $N = (P, T, F, W, m)$ be a P/T net, φ an existential finite-single-path formula that contains $i \in \mathbb{N}$ CTL operators, and ILP_φ the corresponding ILP-problem.
If $\wp(w)_j$ is used before $\wp(w)_k$ with $\forall j, k \in \mathbb{N} : 0 \leq j < k \leq i$, then the resulting sequence is a valid realization ordering.

This keeps the structure of φ in place while realizing a solution of ILP_φ. We continue with the introduction of the main theorem of this section.

Theorem 6.5.3. *Let $N = (P, T, F, W, m)$ be a P/T net with incidence matrix C and φ an existential finite-single-path formula, and ILP_φ the corresponding ILP-problem.*
If φ has a realizable solution in the solution space, it can be reached by applying Theorems 6.1.2, 6.3.1, and 6.4.3 with ILP_φ as the initial ILP-problem and by applying the realization ordering (Definition 6.5.4) for finding a realizable solution.

Proof. We proceed by induction, according to Definition 6.5.1.

Case α (atomic proposition): In CTL an atomic proposition is satisfied, if it holds in the initial marking. Based on Definition 6.5.3 and the fact that no CTL operator is present, no product of $C \cdot \wp(w)$ is added to the equation. It follows that $m = m'$, which means that the atomic proposition must hold in the initial marking.

Case $\mathbf{EF}\,\varphi$: This case can be traced back to Case $\mathbf{E}\,(\alpha\,\mathbf{U}\,\varphi)$ using the tautology $\mathbf{EF}\,\varphi \iff \mathbf{E}\,(\mathtt{TRUE}\,\mathbf{U}\,\varphi)$.

Case $\mathbf{EX}\,\varphi$: Definition 6.5.3 ensures that $C \cdot \wp(w)$ is added to the state equation and that the length constraint for $\mathbf{EX}\,\varphi$ is added to ILP_φ. A witness path for $\mathbf{EX}\,\varphi$ is an existential finite-single-path to the next marking which satisfies φ. The path extended by a witness path for φ at the final marking (which exists by induction hypothesis) yields a witness path for $\mathbf{EX}\,\varphi$. Theorem 6.3.1 makes sure that if a realizable solution exists, the witness path for $\mathbf{EX}\,\varphi$ is found, and Definition 6.5.4 ensures that the witness path is added at the correct position to keep the structure of the formula in place.

Case $\mathbf{E}\,(\alpha\,\mathbf{U}\,\varphi)$: This case is similar to the previous one. Definition 6.5.3 ensures that $C \cdot \wp(w)$ is added to the state equation and that the balance constraints are added to ILP_φ. A witness path for $\mathbf{E}\,(\alpha\,\mathbf{U}\,\varphi)$ is an existential finite-single-path where α is true in every marking until a marking is reached where φ holds. Theorem 6.4.3 makes sure that if a realizable solution exists, the witness path for $\mathbf{E}\,(\alpha\,\mathbf{U}\,\varphi)$ is found and Definition 6.5.4 ensures that the witness path is added at the correct marking (which exists by induction hypothesis) to keep the structure of the formula in place.

Case $\varphi \vee \psi$: If φ is satisfied, then there exists a witness path for φ for which the induction hypothesis may be applied. Otherwise, there is a witness path for ψ for which again the induction hypothesis applies. A formula like $\mathbf{EX}\,(\varphi \vee \psi)$ can be rewritten to $\mathbf{EX}\,\varphi \vee \mathbf{EX}\,\psi$ and both sides are verified separately.

Case $\varphi \wedge \alpha$: In this case, φ and α must be satisfied. Since α is an atomic proposition, only the initial marking of the path is concerned. Hence, the induction hypothesis applied to φ yields the desired result.

□

As an example, consider $\mathbf{E}\,(\alpha_1\,\mathbf{U}\,(\mathbf{E}\,(\alpha_2\,\mathbf{U}\,\varphi)))$ as a formula with nested CTL operators. The idea is to use for each CTL operator one state equation with its own set of variables and constraints, and then solve the ILP-problem consisting of all state equations and constraints.
In our example, the first objective would be to solve the left (outer) $\mathbf{EU}$-formula. That is, we have to reach a marking $m' \models \alpha_2$, while keeping α_1 true along the path. For this, we have to solve the ILP-problem consisting of the state equation $m + C \cdot \wp(w)_1 = m'$ and the balance constraints for α_1.
The second objective is to solve the right (inner) $\mathbf{EU}$-formula. Here we are using the same approach as before, that is, we have to reach a marking $m'' \models \varphi$, while keeping α_2 true along the path. For this, we add a slightly

different state equation $m' + C \cdot \wp(w)_2 = m''$ to the ILP-problem. In this state equation, we start with the final marking of the first state equation, namely in marking m'. Furthermore, we introduce a new set of variables $\wp(w)_2$ for our second Parikh vector to reach the final marking m''. The balance constraints to keep α_2 true are added as well. With the marking m' appearing on both sides, both state equations can be merged into one equation $m + C \cdot \wp(w)_1 + C \cdot \wp(w)_2 = m''$.

6.6 Solving EG φ partially

Unfortunately, we are not able to extend this approach to the formula types $\mathbf{EG}\,\varphi$ and $\mathbf{E}\,(\varphi\,\mathbf{R}\,\psi)$. Both can have an infinite witness path. However, we are still able to verify these formulas in certain cases. In this section, we introduce some quick checks to solve $\mathbf{EG}\,\varphi$ with CEGAR in specific situations.

Definition 6.6.1 ($\mathbf{EG}\,\varphi$).
If $N = (P, T, F, W, m)$ is a bounded P/T net and φ an atomic proposition, then $m \models \mathbf{EG}\,\varphi \iff \exists w \in \mathrm{Paths(m)} : m \xrightarrow{w} m'$, with $\forall i \in \mathbb{N} : (m_i \models \varphi)$.

The definition says that the property φ is true along a path w, if at least one of two conditions is fulfilled. Either there exists an infinite path containing a cycle or the path ends in a deadlock. More precise:

1. Either the witness path is infinite by going through a cycle. Then there are two finite sequences $w_1 w_2$ with $m \xrightarrow{w_1} m' \xrightarrow{w_2} m'$, where w_1 is the finite path leading to a marking m' where w_2 is repeated infinitely. Each state in both w_1 and w_2 satisfies φ.
2. Or the witness path w ends in a deadlock, then every state in w including the last (deadlock) state must satisfy φ.

If there are deadlocks in the system, then we can create necessary or sufficient conditions to solve $\mathbf{EG}\,\varphi$ in both cases. If there are no deadlocks, the only possibility to satisfy $\mathbf{EG}\,\varphi$ is, to prove the existence of a path w_1 to a cycle w_2 where φ stays true along both w_1 and w_2. The cycle is basically a T-invariant and we can reformulate the problem of solving $\mathbf{EG}\,\varphi$ into solving the state equation once to reach a marking m' and finding a T-invariant that starts in m' and keeps φ true. This situation can be formalized as

$$m \xrightarrow{m + C \cdot \wp(w_1) = m'} m' \xrightarrow{C \cdot \wp(w_2) = 0} m'.$$

However, solving this equation is difficult because m' (and $\wp(w_1)$) is not known in advance. The reason for this is that there can be exponentially

many T-invariants which keep φ true in every state and they can potentially start in several different markings. In addition to this, we would have to solve the problem of finding a minimal marking to fire a T-invariant, where minimal is in regard to the entire token number of the marking. Although, it would also make no difference if minimal is meant in regard to the componentwise comparison of markings where no more tokens can be removed from places. To the best of our knowledge there is no polynomial algorithm known for this problem.
We could use a brute force method to calculate for every possible sequence of a T-invariant the minimal required markings to fire completely. This means, all permutations have to be computed. Then, all markings can be compared and we can search for the minimal markings. The runtime for this approach would be exponential. All in all, this is not a suitable approach to solve $\mathbf{EG}\,\varphi$, especially in connection with the possibility that there can be exponentially many T-variants.
But on the bright side, the second part of the equation, the T-invariant, can be used to build a necessary condition for the verification of $\mathbf{EG}\,\varphi$. In fact, if there are no T-invariants that keep φ true and if there is also no deadlock, then we know that $\mathbf{EG}\,\varphi$ cannot be true. To verify this, we can add an adjusted version of the balance constraint to an according ILP-problem to find suitable T-invariants.

Definition 6.6.2 (Minimum constraints).
Let $N = (P, T, F, W, m)$ be a P/T net, $\varphi = s_0 \leq k_0 \wedge s_1 \leq k_1 \wedge \ldots \wedge s_n \leq k_n$ an atomic propositions with $k_1, \ldots, k_n, n \in N$, and $s_1, \ldots, s_n$ formal sums. Further, let $T_i = \{t \in T | \Delta_{t,s_i} \neq 0\}$ be the set of transitions that can change the value of s_i. The set T_i contains all weakly/strongly increasing/decreasing transitions with respect to s_i. We call $\sum_{t \in T_i} \Delta_{t,s_i} \leq 0$ the minimum constraint *with respect to s_i.*

The minimum constraints ensure that the sum of all increasing and decreasing transitions is smaller than or equal to zero. Otherwise, the truth value of the proposition will be changed.

Theorem 6.6.1. *Let $N = (P, T, F, W, m)$ be a P/T net, $\varphi = s_0 \leq k_0 \wedge s_1 \leq k_1 \wedge \ldots \wedge s_n \leq k_n$ an atomic propositions with $k_1, \ldots, k_n, n \in N$, $s_1, \ldots, s_n$ formal sums and it holds that $m \models \varphi$.*
If N has no deadlocks and if the ILP-problem for finding a T-invariant, $C \cdot \wp(w) = \mathbf{0}$, in addition with the minimum constraints for all s_i, has no solution, then m does not satisfy $\mathbf{EG}\,\varphi$.

Proof. Based on Definition 6.6.1, if N has no deadlocks, then the only way to satisfy $\mathbf{EG}\,\varphi$ is to find a cycle w which keeps φ true in every state. Such a

cycle w would represent a T-invariant and, hence, the equation $C \cdot \wp(w) = 0$ must be fulfilled by w. Applying Definition 6.6.2, the minimum constraints ensure that φ remains true in the cycle. If the ILP-problem $C \cdot \wp(w) = 0$ in addition with the minimum constraints is infeasible, then no T-invariant exists. It follows that there is also no cycle that keeps φ true. Hence, $\mathbf{EG}\,\varphi$ can not be true. □

To check the P/T net for the presence of deadlocks seems to be an additional hurdle, but in many cases this hurdle is easily overcome. There exist, next to the standard search for deadlocks, supplementary methods that can find deadlocks or their absence quite fast. For example, such methods are based on the siphon-trap property [94] or on random walks/find-path algorithms [108]. In case the P/T net has deadlocks, we can build a sufficient quick-check to prove $\mathbf{EG}\,\varphi$. We use the fact that $\mathbf{EG}\,\varphi$ is true if the path ends in a deadlock and every state along the path satisfies φ. In CTL this condition can be rewritten to $\mathbf{E}\,(\varphi\,\mathbf{U}\,(\varphi \wedge \texttt{DEADLOCK}))$. We already know how to solve $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$ formulas. Therefore, the only thing left to do is to encode the DEADLOCK-predicate into the ILP-problem. The DEADLOCK-predicate can be easily expressed as a conjunction of disjunctions over atomic propositions.

Definition 6.6.3 (Deadlock constraint).
If $N = (P, T, F, W, m)$ is a P/T net, then

$$\bigwedge_{t \in T} \bigvee_{p \in {}^\bullet t} m'(p) < W(p,t)$$

is a deadlock constraint *for m'.*

A deadlock constrain, which describes a deadlock marking m'.

Lemma 6.6.2. *Let $N = (P, T, F, W, m)$ be a P/T net with deadlocks. $\mathbf{EG}\,\varphi$ holds in m if the ILP-problem for $\mathbf{E}\,(\varphi\,\mathbf{U}\,(\varphi \wedge \texttt{DEADLOCK}))$ has a realizable solution.*

Proof. If N has deadlocks, then according to Definition 6.6.1 $\mathbf{EG}\,\varphi$ is true if a path that satisfies φ in every state ends in a deadlock. Furthermore, Definition 6.4.1 states that φ is true until ψ holds and ψ is in this case $(\varphi \wedge \texttt{DEADLOCK})$. □

6.7 Quick checks

This section presents a set of necessary or sufficient "quick checks" for a whole range of CTL formulas. The quick checks are based Petri net structure theory. Such structure theory is for example the Commoner's theorem [27, 52]. If

it applies, **EF** DEADLOCK evaluates to false. The conditions of the theorem can be checked as a satisfiability problem in propositional logic (SAT) [94]. For the verification of other reachability queries, the state equation approach described in this chapter provides a powerful method based on an ILP-problem. Since structural methods can be traced back to NP-complete problems such as SAT and ILP, respectively and therefore use only polynomial space, they can be applied in parallel to state space exploration. In the following, we introduce several quick checks for some specialized routines. We leave out the approaches, we presented earlier in this chapter, but we believe these approaches should always be run in parallel to the actual state space exploration.
Every existentially quantified formula discussed below corresponds to another universally formula. In fact, they can be reduced to each other by negation and thus, permit the application of the same verification method. We therefore may restrict subsequent considerations to existentially quantified formulas.

- $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$, $\mathbf{A}\,(\varphi\,\mathbf{R}\,\psi)$.
  We can employ linear programming for checking a necessary and a sufficient condition for $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$. A necessary criterion is obviously $\mathbf{EF}\,\psi$, and the state equation approach can be used for checking this condition. A sufficient condition is the reachability of ψ using only transitions that are invisible to φ, in addition to checking φ in the initial marking. This can be checked by removing all transitions visible for φ from N and applying the state equation approach to the resulting net.

- $\mathbf{EG}\,\mathbf{EF}\,\varphi$, $\mathbf{AF}\,\mathbf{AG}\,\varphi$.
  We can add a check for $\mathbf{EF}\,\varphi$ as a necessary condition and $\mathbf{AG}\,\varphi$ as a sufficient criterion to a portfolio for $\mathbf{EG}\,\mathbf{EF}\,\varphi$. Again, the state equation approach can be used in order not to take too much memory away from the main search procedure.

- $\mathbf{EF}\,\mathbf{EG}\,\varphi$, $\mathbf{AG}\,\mathbf{AF}\,\varphi$.
  As in previous cases, $\mathbf{AG}\,\varphi$ is a sufficient condition for $\mathbf{EF}\,\mathbf{EG}\,\varphi$ while $\mathbf{EF}\,\varphi$ is necessary. The state equation approaches to these properties may be added to the portfolio for $\mathbf{EF}\,\mathbf{EG}\,\varphi$.

- $\mathbf{EF}\,\mathbf{AG}\,\varphi$, $\mathbf{AG}\,\mathbf{EF}\,\varphi$, $\mathbf{EF}\,\mathbf{AG}\,\mathbf{EF}\,\varphi$, $\mathbf{AG}\,\mathbf{EF}\,\mathbf{AG}\,\varphi$.
  For all these properties, $\mathbf{AG}\,\varphi$ is a sufficient condition and $\mathbf{EF}\,\varphi$ is necessary. Using the state equation approach, we can add these checks to our portfolio. This way, we have an additional opportunity to answer the query early while using only a moderate amount of additional memory.

6.8 Discussion

In this chapter, we presented an *alternative reduction technique*. We introduced two methods to solve $\mathbf{E}(\varphi \mathbf{U} \psi)$ respectively $(\mathbf{EX})^k \varphi$ with the CEGAR approach for Petri nets. The main concept behind them is to add constraints to the Parikh vector to refine solutions until they become realizable or infeasible. The specialized routines can be used to solve the whole class of finite-single-path formulas. Furthermore, we introduced some quick-checks for solving $\mathbf{EG}\,\varphi$ under certain circumstances as well. In addition, we proposed a collection of quick checks based on structural methods that can be run in parallel to the actual verification. All introduced techniques reduce the state explosion problem by a guiding search with the intermediate solution vectors of the refined ILP instances.
We point out the search for realizing sequences can also be supported with partial order reduction (see Chapter 9 and 11).
To the best of our knowledge, this is the first extension of CEGAR to temporal logic other than reachability. The question whether CEGAR can be extended to other classes of temporal logic is open. To solve branching formulas with this approach is rather difficult because CEGAR is based on searching a single counterexample.
Another future field of research is to look further into the remaining four basic CTL operators, namely $\mathbf{EG}\,\varphi$, $\mathbf{E}(\varphi \mathbf{R} \psi)$, and their universal counterparts. But this requires to deal with infinite paths.
Furthermore, we also want to find in a systematically way additional quick checks based on structural methods for other formula types.
As a future work, we want to implement the proposed techniques into our model checker LoLA [142]. Once implemented, we expect a substantial increase in the verification performance of these formulas. Especially in case of negative results the procedure will terminate quickly, because the ILP-problem will become rather fast infeasible. We expect a similar performance increase as it was the case for the CEGAR approach for reachability analysis. The performance of LoLA in the reachability category increased from 75 % to 90 % in the MCC with the introduction of the CEGAR approach.

Part III

Supplementary strength reduction

The previous part was concerned with reduction techniques. To increase the efficiency of model checking, all aspects involved in model checking should be tuned. This part is dedicated to supplementary strength reduction techniques. These are not reduction techniques per se, but additional methods that can accelerate the model checking process.
A lot of challenges in model checking reduce to searching the reachability graph. Therefore, building, browsing and accessing the reachability graph are time-critical operations deserving elaborate acceleration efforts in every detail. One of the critical points while building the state space, is to determine the neighbors of a state, which reduces to testing whether a transition is enabled or not [87]. This means, in each state the list of enabled transitions, which can lead to new states, must be computed. For efficiency, the aim is to avoid checking in every state every transition for enabledness. To prevent this, we only want to update the enabledness information of transitions that are possibly disabled or enabled whenever a transition t fires. To this end, before building the state space, we preprocess a data structure $DI(N)$ that for all given transitions t lists the transitions whose enabling status could have changed by firing t. Although saving up much time during the actual state space exploration, the preprocessing of $DI(N)$ has been an unpleasantly costly investment. In Chapter 7, we introduce a new, generally much faster method to compute $DI(N)$.
In addition, Chapter 8 is going to introduce a range of formula simplification techniques to speed-up model checking. It is easier and thus more performant to verify a simplified specification [7].

Chapter 7

Acceleration of enabledness-updates

In this chapter, we are concerned with our research goal of accelerating the *state space exploration*. In model checking, the main task is to build the, possibly reduced, state space of a P/T net $N = (P, T, F, W, m_0)$. One of the critical points while building the state space, is to determine whether a transition is enabled or not to compute successor states. That is, in each state the list of enabled transitions, which can lead to new states, must be computed. Computing the enabledness information for a single transition is cheap, however, computing it for all transitions in every marking amounts to a lot of computing resources. And since the state space in real world P/T nets is usually vast, browsing follow up states has to be fast. Thus, the aim is to avoid checking in every state every transition for enabledness.
In real world P/T nets, it is noticeable that the impact of firing a transition $t \in T$ is usually local and does not affect the enabledness of all other transitions. This means, only for some transitions the enabledness actually changes. Using this insight, we check the enabledness information of all transitions only once, namely in the initial state. After this, whenever a fired transition t leads to a new state, we just want to update the enabledness information of transitions that may have been disabled or enabled by the firing of t. To this end, in the preprocessing, before building the state space, we compute a look-up data structure, called $DI(N)$, which, basically, maps to every transition a list of possibly disabled/enabled transitions. The advantage is that most lists of $DI(N)$ are generally shorter than the list of all transitions. Thus, on average, they are faster to process.
The problem of speeding-up the computation of the state space using enabling tests has been studied since 25 years [87, 121]. The problem is connected to the so-called token game, i.e., the firing of transitions, while simu-

lating a Petri net [11]. In [11] a method is described that is based on a notion called linear enabling functions and the according classification of transitions into five categories. One issue with this approach is that the input Petri net has to be transformed, and so-called silent and preemptive transitions have to be added to it. In [87] a method is presented mainly to work in the context of unfolding algebraic nets [105, 138]. The authors compared their implementation with the tool LoLA 1.0 [111] and came to the conclusion that LoLA 1.0 is faster but needs more memory. Since 2010, LoLA 2.0 [142], the successor of LoLA 1.0 and in the following only called LoLA, uses a more advanced method, which we describe in Section 7.3. This method performs well in practice, which is highlighted by the benchmark of LoLA at the yearly MCC [66–68]. However, the overhead is unpleasantly costly and sometimes the approach needs several minutes or more for certain models at the benchmark to compute $DI(N)$.
In this chapter, we introduce a new and faster method for preprocessing $DI(N)$. The key to the performance gain is an indexed graph data structure and a rigorous reduction of costly copying operations. We have published this method with its results in [77]. The following chapter is based on this publication. Our approach is also used to speed up the computation of conflicting transitions in partial order reduction techniques [49,97,127] which we are going to introduce in Chapter 9. We will describe the exact procedure for this speed up computation in Section 9.4.
The remainder of this chapter is organized as follows. Section 7.1 starts with a motivational example. Section 7.2 introduces concepts to categorize transitions based on their firing effects on other transitions. We continue in Section 7.3 with the description of the former computation of $DI(N)$. Section 7.4 presents our new approach to compute $DI(N)$. Section 7.5 is dedicated to an experimental validation of the proof-of-concept implementation, where we compare the former approach with the new one. Finally, Section 7.6 concludes this chapter with some final remarks and a discussion.

7.1 Motivational example

Consider the P/T net $N = (P, T, F, W, m_0)$ shown in Figure 7.1. It consists of $n \in \mathbb{N}$ transitions and has sequential behavior, meaning that in each marking, except the last deadlock marking, only one transition is enabled. In m_0 transition t_1 is enabled. After firing t_1, we reach a new marking m_1. In a naive approach, all transitions need to be checked for their enabledness in m_1. However, as seen in the figure, only the enabledness information of t_1 and t_2 change in the transition from m_0 to m_1. Thus, instead of n enabledness

checks in each marking, we only need two.

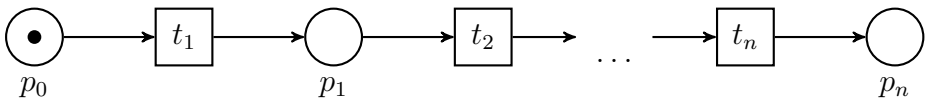


Figure 7.1: Example for unnecessary enabledness updates.

This is an example for the local impact of firing a transition t. Usually, it does not affect the enabledness of all transitions. This leads us to the question: How can the local effect of firing t be described? We firstly observe that the neighborhood of t is static. Hence, multiple firing of t always affects the same neighborhood. In the example illustrated in Figure 7.2, firing t_1 always has an effect on t_1 and t_2 but never t_3.

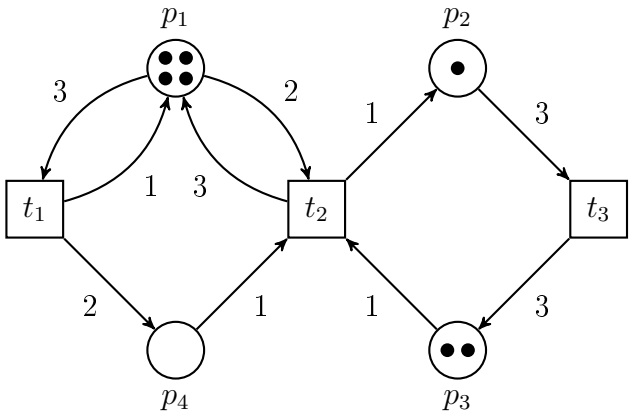


Figure 7.2: Running example P/T net.

Depending on a given transition t, we, thus, want the list of all transitions that are affected by firing t. After the enabledness information of all transitions have been computed for the initial marking, we never have to do the whole job again. Instead, in each marking, we just update this list for transition that could have been affected in their enabledness. For example, in the initial marking m_0 of the P/T net N illustrated in Figure 7.2, only t_1 is enabled. As seen in Figure 7.3 which shows the reachability graph of N, firing t_1 in m_0 leads to marking $m_1 = (2, 1, 2, 2)$. Here, we have to update the enabledness information from t_1 and t_2, but not from t_3.
We use the example P/T net from Figure 7.2 with its reachability graph shown in Figure 7.3 as a running example in this chapter.

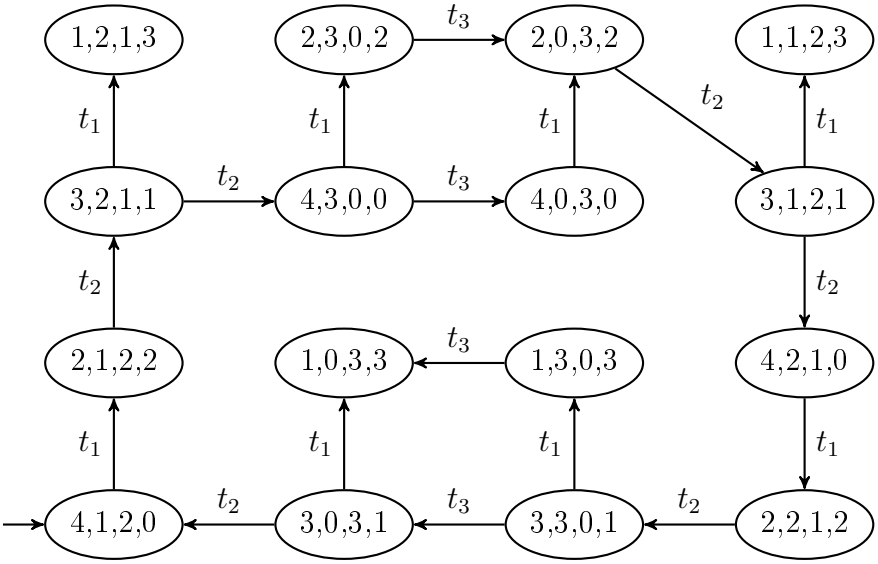


Figure 7.3: Reachability graph of the example P/T net from Figure 7.2.

7.2 Preprocessing decreasing and increasing transitions

If a transition t fires, and thereby takes N from marking m to m', it may, in the course of this, disable a previously enabled transition t', if t consumes from a place $p \in {}^\bullet t \cap {}^\bullet t'$. More precisely, we would like to capture the situation (i) where both, t and t', are enabled in m, hence, $m(p) \geq \max\{W(p,t), W(p,t')\}$, and (ii) where t' is not enabled in m', anymore, hence, $W(p,t') > m'(p) = m(p) - W(p,t) + W(t,p)$. As preprocessing is unaware of the specific markings m and m', we need to combine (i) and (ii) for the condition

$$W(p,t') > \max\{W(p,t), W(p,t')\} - W(p,t) + W(t,p).$$

In such a case, we say that t *decreases* t' or that t' is *decreased by* t and denote this as the binary relation $t \searrow t'$. In the case of $W(p,t) \leq W(p,t')$, the decreasing condition reduces to $W(t,p) < W(p,t)$ and, otherwise, it simply becomes $W(t,p) < W(p,t')$. Taking everything together, we get the following definition:

Definition 7.2.1 (Decreasing transitions).
We call a transition $t' \in T$ decreased by *a transition* t*, if there exists a place* $p \in {}^\bullet t \cap {}^\bullet t'$ *with* $W(t,p) < W(p,t)$ *and* $W(t,p) < W(p,t')$ *and denote this by* $t \searrow t'$.

As an example, consider the firing of t_1 in m_0 from our running example. In this case, t_1 is decreased by itself, because firing t_1 reduces the number of tokens on p_1 and t_1 leaves less tokens on p_1 than it needs to fire again. More precisely, t_1 is decreased by itself, because with p_1, according to Definition 7.2.1, there exists a place that is in the set ${}^\bullet t \cap {}^\bullet t'$ with

$$1 = W(t_1, p_1) < W(p_1, t_1) = 3 \text{ and } 1 = W(t_1, p_1) < W(p_1, t_1) = 3$$

On the other hand, if $m \xrightarrow{t} m'$, it may also happen that a previously disabled transition t' becomes enabled in m', namely if t produces tokens on a place $p \in t^\bullet \cap {}^\bullet t'$. In this simpler situation, we have

$$0 < W(p, t') \leq m'(p) = m(p) - W(p, t) + W(t, p),$$

which leads to the following definition, right away:

Definition 7.2.2 (Increasing transitions).
We call a transition $t' \in T$ increased by *a transition t, if there exists a place $p \in t^\bullet \cap {}^\bullet t'$ with $W(t, p) > W(p, t)$ and $0 < W(p, t')$ and denote this by $t \nearrow t'$.*

As an example, consider again the firing of t_1 in m_0 from our running example. In this case, t_2 is increased by t_1, because t_1 produces additional tokens on p_4. More precisely, t_2 is increased by t_1, because with p_4, according to Definition 7.2.2, there exists a place that is in the set $t^\bullet \cap {}^\bullet t'$ with

$$2 = W(t_1, p_4) > W(p_4, t_1) = 0 \text{ and } 0 < W(p_4, t_2) = 1$$

By breaking down the concepts of decreasing and increasing to binary relations, we are able to describe all information as directed graphs.

Definition 7.2.3 (Decrease and increase graph).
If N is a P/T net with transition set T then we call the directed graph $D(N) = (T, \searrow)$ with node set T and arc set $\searrow$ the decrease graph *of N and the directed graph $I(N) = (T, \nearrow)$ with node set T and arc set $\nearrow$ the* increase graph *of N. The pair of decrease- and increase graph is subsequently denoted as $DI(N)$.*

Let us consider our running example, again, to build the increase graph $I(N)$ which is depicted in Figure 7.4. As we have seen before, our example has an arc from t_1 to t_2 caused by place p_4. Furthermore, we see that t_2 also increases t_1. This happens according to place p_1 being in $t_2^\bullet$ and in ${}^\bullet t_1$, which means that t_2 produces more tokens on p_1 than it consumes. In addition, t_2 increases itself, too, also based on p_1. The place p_2 is responsible for an arc from t_2 to t_3. Finally, p_3 adds an arc from t_3 back to t_2.

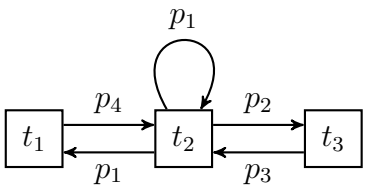


Figure 7.4: Increase graph of the running example.

Using $I(N)$, we can always get a list of candidate transitions that can become enabled after a firing. For example, if t_3 fires then we have to test t_2 for possible enabledness. However, we also need to know which transitions become disabled after firing to remove them from the list of enabled transitions. For this, we build the decrease graph $D(N)$ analogously. The decrease graph is shown in Figure 7.5. For example, if t_1 fires then we have to evaluate t_1 and t_2. Combining the increase and the decrease graph, we get $DI(N)$, the data structure that solves the problem of updating the list of enabled transitions.

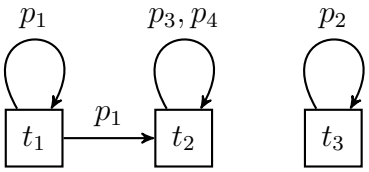


Figure 7.5: Decrease graph of the running example.

Having preprocessed $DI(N)$, building the reachability graph of a net N can be sped up. In fact, whenever we traverse an edge $m \xrightarrow{t} m'$ from a node m to a new node m' by firing a transition t, we have to determine the set of reachability arcs that are incident to m'. In other words, this means to compute the set $T(m')$ of transitions in N that are enabled in m'. However, as we come from m, we are already in possession of $T(m)$ and, probably, the difference between $T(m)$ and $T(m')$ is not too big. This is where $DI(N)$ helps us to make just a few updates of $T(m)$ in order to get $T(m')$.
We begin with the decrease graph $D(N)$ and obtain the neighborhood $d(t)$ of t, which consists of all transitions that may become disabled by firing t. Secondly, we also compute the neighborhood $i(t)$ of t in $I(N)$ for the transitions that may become enabled after firing t. Afterwards, the job of computing $T(m')$ reduces to the following update:

$$\begin{aligned} T(m') = T(m) &- \{t' \in d(t) \mid \exists p \in {}^\bullet t' : W(p,t') > m'(p)\} \\ &+ \{t' \in i(t) \mid \forall p \in {}^\bullet t' : W(p,t') \leq m'(p)\} \end{aligned}$$

As this update considers only the transitions of $d(t) \cup i(t)$, which is usually much smaller than the whole T, investing into the computation of $DI(N)$ pays out.
However, the MCC benchmark proofs that the preprocessing of $DI(N)$ can be a very time consuming preprocess. For some models, preprocessing $DI(N)$ took several minutes or even several hours. For more details see the experimental Section 7.5. Before we introduce our new approach to compute $DI(N)$, we describe the former method.

7.3 The former computation of the decrease-increase-graph

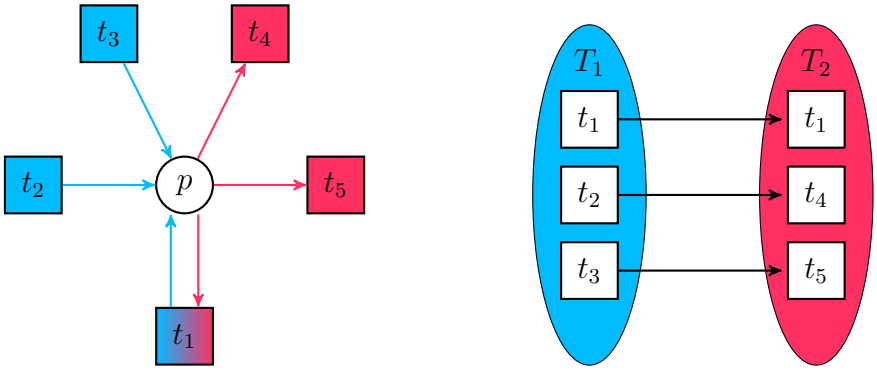
For any ordering $p_1, \ldots, p_n$ of the places of the P/T net $N = (P, T, F, W, m_0)$, we define for every $i \in \{0, \ldots, n\}$ the subnet N_i that, while containing all transitions T, only consists of the places $P_i = \{p_1, \ldots, p_i\}$ and the arcs that go between T and P_i. The former approach to the computation of $DI(N) = DI(N_n)$ is to start with $DI(N_0)$ and then consider the place sequence $p_1, \ldots, p_n$ in order to successively obtain $DI(N_i)$ for all $i \in \{1, \ldots, n\}$. It is easy to see that $DI(N_{i+1})$ is just $DI(N_i)$ plus the edges induced by p_{i+1}. This happens, as the decreasing and the increasing relations between transitions are defined only existentially over the place set. In other words, if $DI(N_i)$ has a $\searrow$-edge or $\nearrow$-edge, respectively, between two transitions t, t' then considering an additional place p_{i+1} cannot revoke the existence of the place $p \in \{p_1, \ldots, p_i\}$ that justified the aforesaid edge between t, t'. For that reason, it makes sense to define $\searrow(p_{i+1})$ and $\nearrow(p_{i+1})$, the edges of $DI(N_{i+1})$ that are additionally introduced by the consideration of p.

Definition 7.3.1 (Incremental edges).
For all places $p \in P$, the set $\searrow(p) = \{(t, t') \mid W(t,p) < W(p,t), W(t,p) < W(p,t')\}$ is called decreasing edges *of p and the set $\nearrow(p) = \{(t, t') \mid W(t,p) > W(p,t), 0 < W(p,t')\}$ is the* increasing edges *of p.*

The set $\nearrow(p)$ can be represented in the following way. There are always two transition sets $T^0, T^1 \subseteq T$ such that $\nearrow(p) = \{(t, t') \mid t \in T^0, t' \in T^1\}$. In fact, $T^0 = \{t \mid W(t,p) > W(p,t)\}$ and $T^1 = p^\bullet$. We capture this property in the following definition.

Definition 7.3.2 (Homogeneous pair).
A pair (T^0, T^1) of transition subsets of T is called $\nearrow$-homogeneous *if $t \nearrow t'$ for all $t \in T^0$ and all $t' \in T^1$. This is also denoted as $T^0 \nearrow T^1$. Analogously,*

it is called $\searrow$-homogeneous *if* $t \searrow t'$ *for all* $t \in T^0$ *and all* $t' \in T^1$*, which is denoted as* $T^0 \searrow T^1$*.*



(a) Snippet of a P/T net. (b) Homogeneous pair of the snippet.

Figure 7.6: Example homogeneous pair.

As an example for a homogeneous pair for $I(N)$ consider Figure 7.6. The left Figure 7.6a illustrates a snippet of a P/T net and the right Figure 7.6b shows the homogeneous pair (T_1, T_2) for place p. All blue transitions form the set T_1 and the red transitions form T_2. All transitions in T_1 are increasing all transitions in T_2, meaning that there is an arc from every transition in T_1 to every transition in T_2 caused by p. Note that transitions can be in both sets at the same time like transition t_1.
If (T^0, T^1) is exactly the $\nearrow$-homogeneous pair of $\nearrow(p)$, this is made explicit by writing $T^0 \nearrow_p T^1$. The set $\searrow(p)$, in turn, is generally not describable as a single $\searrow$-homogeneous pair. That is why we fall back on the set

$$\dot{\searrow}(p) = \{(t, t') \mid W(t,p) < W(p,t), 0 < W(p,t')\}.$$

As, obviously, $\searrow(p) \subseteq \dot{\searrow}(p)$, this set is weaker but sufficient for the anticipated purpose. Moreover, $T^0 = \{t \mid W(t,p) < W(p,t)\}$ and $T^1 = {}^\bullet p$ provide a $\dot{\searrow}$-homogeneous pair for $T^0 \dot{\searrow}_p T^1$, which stands for $\dot{\searrow}(p) = \{(t,t') \mid t \in T^0, t' \in T^1\}$. In practice, we almost always have $\searrow(p) = \dot{\searrow}(p)$, which justifies this simplification.
The graphs of every $DI(N_i)$ are represented by a list of homogeneous pairs each. More precisely, $I(N_i)$ is given by a list of $\nearrow$-homogeneous pairs. This basically corresponds to a compressed adjacency list representation, where all transitions $t_1, t_2, \ldots$ with the same adjacency list T^1 are together in $T^0 = \{t_1, t_2, \ldots\}$ and we get $T^0 \nearrow T^1$.
Therefore, in iteration $i+1$, we have to go through all homogeneous pairs $T^0_j \nearrow T^1_j$ of $I(N_i)$ and make updates according to $T^0 \nearrow_{p_{i+1}} T^1$ in order to

obtain $I(N_{i+1})$. More precisely, in $I(N_{i+1})$ every pair $T_j^0 \nearrow T_j^1$ is replaced by the new pairs

$$(T_j^0 \setminus T^0) \nearrow T_j^1 \quad \text{and} \quad (T_j^0 \cap T^0) \nearrow (T_j^1 \cup T^1)$$

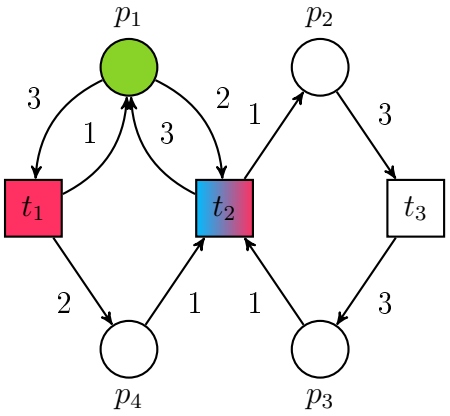
unless they are empty.
Equivalently, every graph $D(N_i)$ is implemented as a list of $\dot{\searrow}$-homogeneous pairs, which have to be modified according to $T^0 \dot{\searrow}_{p_{i+1}} T^1$ for the next step $D(N_{i+1})$.
In our implementation, we use numbers to represent transitions and keep the pair items T_j^0 and T_j^1 as ordered lists. This make the computation of $T_j^0 \setminus T^0$, $T_j^0 \cap T^0$, and $T_j^1 \cup T^1$ linear time operations, which is fairly efficient. Nevertheless, we are forced to touch every pair, even though most of them are probably not intersected by T^0 or T^1. Hence, in worst case, the computation of $DI(N)$ takes $\mathcal{O}(|P| \cdot |T|^2)$ time, as, for every place, we need to consider $\mathcal{O}(|T|)$ homogeneous pairs in $DI(N_i)$ and process each of them in linear time $\mathcal{O}(|T|)$.

7.4 Accelerated computation of the decrease-increase-graph

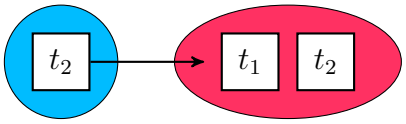
In our former approach, a lot of time is wasted in updating the adjacency lists. In every iteration, all lists have to be touched while most of them are not even relevant and, in case of an actual update, many copying operations occur. The speed-up idea is to create a more efficient way of determining the necessary updates. Moreover, in order to omit unnecessarily copying arrays around, the actual creation of adjacency lists is postponed until after the iteration of all places. If we want to build the graph $G(N)$, which is either $I(N)$ or $D(N)$, our new approach works in three steps:

1. Homogeneous pairs

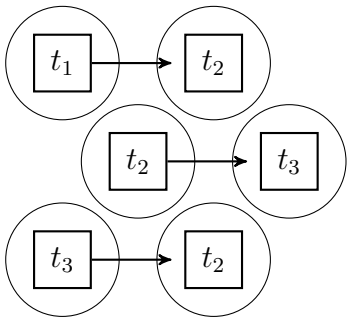
For a given ordering $p_1, \ldots, p_n$ of the places, we generate the corresponding list of homogeneous pairs $T_1^0 \to T_1^1, T_2^0 \to T_2^1, \ldots$, where $\to$ stands for $\nearrow$ in case of the computation of $I(N)$, or for $\dot{\searrow}$ if $D(N)$ is about to be built. For that matter, we like to point out that the new computation method of $D(N)$ does not fall back onto $\dot{\searrow}$ but, instead, processes every place into a set of possibly more than one homogeneous pair. As the definition of these homogeneous pairs is clear at this point, we do not go into the details of their computation in this section, however, we provide an example.



(a) Running example.



(b) Homogeneous pair for p_1.



(c) Remaining homogeneous pairs.

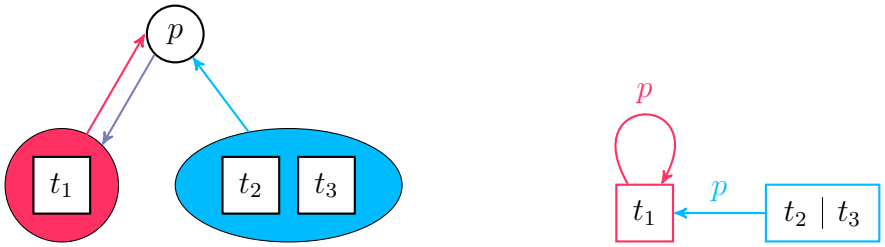
Figure 7.7: Homogeneous pairs of the running example.

In our running example, as seen in Figure 7.7a, we focus on $I(N)$. Accordingly, the green place p_1 induces a homogeneous pair depicted in Figure 7.7b with T_1 containing just the transition t_2 and the set T_2 consists of the transitions t_1 and t_2. This is done for all places $p \in P$ which leads to several homogeneous pairs. In our example, this results in four pairs as seen in Figure 7.7c. In the general setup, we end up with a list of homogeneous pairs that altogether already describe all arcs of $I(N)$ or $D(N)$.

2. Intermediate directed graph

Iterating through the list of homogeneous pairs, we progressively create an intermediate directed graph $H(N) = (X, Y, Z, \theta)$ on node sets X and Y, directed edges $Z \subseteq (X \times Y) \cup (Y \times X)$, and a partial function $\theta : T \to X$. This graph means to implicitly encode $G(N)$. Every node in $x \in X$ represents a set $T_x^0 = \{t \in T \mid \theta(t) = x\}$ of transitions with the same adjacency list. The adjacency between the different nodes of X is realized indirectly. More precisely, every $x \in X$ defines a compressed adjacency list by the homogeneous pair $T_x^0 \to T_x^1$ with $T_x^1 = \bigcup_{(x,y)\in Z} \bigcup_{(y,x')\in Z} \{t' \mid \theta(t') = x'\}$. Altogether, every adjacency list of the target graph $G(N)$ is represented by one node of X.

As an example, consider Figure 7.8. The left Figure 7.8a shows the intermediate graph $H(N)$ of homogeneous pair (T_1, T_2), $T_1 = \{t_1\}$, $T_2 = \{t_2, t_3\}$ based on place p. The right Figure 7.8b shows the computed increase graph $I(N)$ based on $H(N)$. Note, if two transitions have the same environment



(a) Intermediate adjacency graph $H(N)$. (b) Increase graph based on $H(N)$.

Figure 7.8: New computation of $I(N)$.

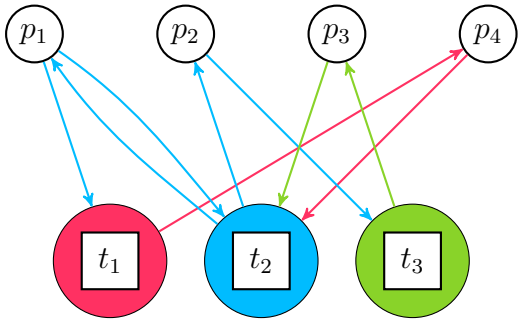
like t_2 and t_3 then we only need to compute one. For example, if we compute t_2 for $I(N)$ then t_3 just needs a pointer to t_2.

The efficient computation of $H(N)$ works iteratively. We assume that we have computed the list of homogeneous pairs $T_1^0 \to T_1^1, T_2^0 \to T_2^1, \ldots, T_k^0 \to T_k^1$ from the list of places. Then we start from an empty graph $H_0(N)$ and, step-by-step, integrate every pair $T_{j+1}^0 \to T_{j+1}^1$ into $H_j(N)$ to get $H_{j+1}(N)$ and, at the end, $H(N) = H_k(N)$. Accordingly, every partial solution $H_j(N)$ encodes a graph $G_j(N)$ with all the edges of $T_1^0 \to T_1^1, \ldots, T_j^0 \to T_j^1$.

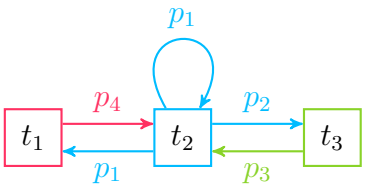
Then, when integrating the pair $T_{j+1}^0 \to T_{j+1}^1$, we firstly have to process all the nodes $x_1, \ldots, x_r, x'_1, \ldots, x'_s \in X$ where T_{j+1}^0 properly intersects $T_{x_i}^0, i \in \{1, \ldots, r\}$ or T_{j+1}^1 properly intersects $T_{x'_i}^0, i \in \{1, \ldots, s\}$. More precisely, we have to make copies $\hat{x}_1, \ldots, \hat{x}_r, \hat{x}'_1, \ldots, \hat{x}'_s$ having the same neighborhoods in Y as the original nodes. Moreover, we need to redefine $\theta(t) = \hat{x}_i$ for all $t \in T_{j+1}^0 \cap T_{x_i}^0$ and $\theta(t') = \hat{x}'_i$ for all $t' \in T_{j+1}^1 \cap T_{x'_i}^0$. This is necessary as, in contrast to the transitions $T_{x_i}^0 \setminus T_{j+1}^0$, all elements of $T_{\hat{x}_i}^0$ will obtain an enhanced neighborhood in $G_{j+1}(N)$, namely T_{j+1}^1. Similarly, the transitions of $T_{\hat{x}'_i}^0$ have to be divided from $T_{x'_i}^0 \setminus T_{j+1}^1$, as only they are neighbors of T_{j+1}^0.

Up to this point, however, $H_{j+1}(N)$ still represents $G_j(N)$ as no edge of $T_{j+1}^0 \to T_{j+1}^1$ has been included. To this end, we add a new node $x \in X$ and define θ for $T_x^0 = T_{j+1}^0 \setminus (T_{x_1}^0 \cup \cdots \cup T_{x_r}^0)$. We also add a new node $x' \in X$ and define θ for $T_{x'}^0 = T_{j+1}^1 \setminus (T_{x'_1}^0 \cup \cdots \cup T_{x'_s}^0)$. Moreover, we add a new node $y \in Y$ for the connection $T_{j+1}^0 \to T_{j+1}^1$. To implement this homogeneous pair in $G_{j+1}(N)$, we, thus, include the edges (x, y) and $(\hat{x}_i, y), i \in \{1, \ldots, r\}$ into $H_{j+1}(N)$ as well as all the edges (y, x') and $(y, \hat{x}'_i), i \in \{1, \ldots, s\}$.

Figure 7.9 shows the new computation of $I(N)$ based on $H(N)$ from the running example. The left Figure 7.9a shows the intermediate graph $H(N)$ of all homogeneous pairs shown in Figure 7.7c. The colors represent increasing relations between the transitions. The right Figure 7.8b shows the full increase graph $I(N)$ based on $H(N)$.



(a) Intermediate adjacency graph $H(N)$.



(b) Increase graph based on $H(N)$.

Figure 7.9: New computation of $I(N)$ for the running example.

3. Compressed adjacency list

In the final step, we take $H(N)$ and extract the compressed adjacency lists of $G(N)$ according to the definition above. After computing an inverse mapping of function θ, this is straightforward and does not need further explanation.

The reason that computing $H(N)$ is much faster than our old method, comes from the fact that finding the intersected nodes $x_1, \ldots, x_r, x'_1, \ldots, x'_s$ does not work by exhaustive search as before. Instead, we just once iterate through the elements t of T^0_{j+1} and T^1_{j+1}, respectively, and use $\theta(t)$ to get the intersected set. Moreover, having the intersected sets, we do not split entire transition arrays but copy only small amounts of edges between X and Y. But while the computation of $H(N)$ works in $\mathcal{O}(|P| \cdot |T|)$ time, the expansion of $H(N)$ into the compressed form of $G(N)$ can still take $\mathcal{O}(|P| \cdot |T|^2)$ time in the worst case. However, that the new approach is usually the better one, even with the overhead of the subsequent adjacency list extraction, is demonstrated experimentally in the next section.

7.5 Experimental validation

Both methods discussed in this chapter are implemented in our proof-of-concept model checker LoLA. For evaluating the methods, we used the benchmark provided by the model checking contest 2019 [68]. The benchmark consists of 94 Petri nets, which result in 1018 instances due to the scaling parameter of some models. We restrict the benchmark to P/T nets and for each net, we only consider the largest available instance. If the model scales over more than one parameter, we choose for every parameter the largest

instance. We ignored smaller instances since they have proportionally the same effect as their larger counterparts. For the "FamilyReunion" net we chose a smaller instance, since the largest instance runs out of memory on our test machine. Overall, our benchmark consists of exactly 100 models.

Model	Model size			Decreased		Increased		Difference		
	|P|	|T|	|F|	Old	New	Old	New	Decr.	Incr.	Both
AirplaneLD-PT-4000	28019	32008	122028	87.9	2.2	49.2	0.5	85.7	48.7	134.4
ASLink-PT-10b	4410	5405	16377	0.1	0.0	1.3	0.0	0.0	1.3	1.3
AutoFlight-PT-96b	7894	7868	18200	0.2	0.0	3.1	0.0	0.2	3.1	3.3
BART-PT-60	8130	12120	97200	7.8	0.1	3.9	0.1	7.7	3.8	11.5
BridgeAndVehicles-PT-V80P50N50	228	8588	67470	3.2	1.0	0.0	0.1	2.2	-0.1	2.1
CloudDeployment-PT-7b	2271	19752	389666	0.2	38.5	0.6	0.0	-38.3	0.6	-37.7
DatabaseWithMutex-PT-40	12920	12800	156800	0.3	0.1	9.7	0.1	0.2	9.6	9.8
Dekker-PT-200	1000	40400	320400	21.7	1.4	0.2	0.4	20.4	-0.2	20.2
DLCflexbar-PT-8a	3971	32571	129321	126.5	0.0	0.7	0.1	126.5	0.6	127.1
DLCflexbar-PT-8b	47560	76160	216499	9.8	0.4	335.2	0.2	9.4	335.1	344.5
DLCround-PT-13b	5343	8727	24849	0.1	0.0	2.2	0.0	0.1	2.2	2.3
DLCshifumi-PT-6a	3568	25936	101182	61.9	0.0	0.6	0.1	61.9	0.5	62.4
DLCshifumi-PT-6b	44243	66611	182532	8.3	0.3	260.2	0.1	8.0	260.1	268.1
DoubleExponent-PT-200	10604	9998	28194	0.2	0.0	5.5	0.0	0.2	5.5	5.7
DrinkVendingMachine-PT-10	120	111160	1026520	38.0	22.7	6.8	0.1	15.3	6.7	21.9
FamilyReunion-PT-L00200M0020C010P010G005	143908	134279	411469	36.7	0.5	2450.4	0.5	36.1	2450.0	2486.1
FlexibleBarrier-PT-22b	6478	7469	18797	0.1	0.0	1.8	0.0	0.1	1.7	1.8
GlobalResAllocation-PT-5	102	136662	1226388	1.9	0.7	1.0	0.2	1.2	0.8	2.0
HexagonalGrid-PT-816	3391	6174	24696	0.0	0.0	1.4	0.0	0.0	1.4	1.4
HypertorusGrid-PT-d5k3p2b10	7533	24300	97200	0.2	0.0	15.9	0.1	0.2	15.8	16.0
JoinFreeModules-PT-5000	25001	40001	115002	5.2	0.1	56.9	0.1	5.1	56.8	62.0
NeighborGrid-PT-d5n4m1t35	1024	196608	393216	0.6	0.1	1.9	2.1	0.5	-0.3	0.3
NeoElection-PT-8	10062	22266	129195	2.2	1.4	2.9	0.1	0.8	2.8	3.5
NoC3x3-PT-8B	9140	14577	30726	0.4	0.0	5.3	0.0	0.4	5.2	5.6
PhaseVariation-PT-D30CS100	2702	30977	216835	23.7	0.9	0.8	1.5	22.7	-0.7	22.0
Philosophers-PT-10000	50000	50000	160000	195.2	0.2	143.5	0.1	195.0	143.4	338.4
PhilosophersDyn-PT-20	540	17220	140780	3.8	0.8	1.5	0.4	3.0	1.1	4.2
Railroad-PT-100	1018	10506	62728	1.6	0.3	0.9	0.2	1.4	0.6	2.0
RERS17pb113-PT-9	639	31353	125418	5.6	0.5	1.4	0.3	5.1	1.1	6.2
RERS17pb114-PT-9	1446	151085	604252	107.5	6.2	27.1	4.5	101.2	22.6	123.9
RERS17pb115-PT-9	1399	144369	577414	90.8	7.4	30.4	4.5	83.3	25.9	109.3
RwMutex-PT-r2000w0010	6020	4020	52040	1.2	0.0	1.6	0.0	1.2	1.6	2.8
SafeBus-PT-20	1026	10461	77364	1.0	0.5	0.2	0.3	0.6	-0.1	0.5
SharedMemory-PT-200	40601	80200	320000	2992.3	45.1	3821.9	38.6	2947.2	3783.4	6730.6
TokenRing-PT-50	2601	127551	510204	840.1	1.4	3.1	0.8	838.8	2.3	841.1

Table 7.1: Time comparison in seconds between the former (here called old) and the new method to compute enabledness and conflicts.

Experiments were executed on a machine with 32 physical cores running at 2.7 GHz and 1 TB of RAM. All computations were done with no time and memory restrictions. The time was measured with the C++ chrono library using the high resolution clock. We only show experiments where one of the methods needed more than one second. Table 7.1 lists the results of our experiments. It shows that the new method is in almost all cases at least one order of magnitude faster. Although both methods are asymptotically

of the same complexity, it seems that they have different worst case inputs. Our experiments suggest that the ones that slow down the new approach are less frequent in practice than those making the former one slow.
In the remainder of this section, we want to take a closer look at the outcome of our experiments. As Table 7.1 shows for the decreasing computation, both methods needed less than 1 ms for 29 models. In 7 models the former method was faster, whereas in 6 of them the difference amounted to only a couple of milliseconds. In the remaining 64 models the new approach was faster. All in all the new approach needed only 2.86 % of the time compared to the former method. Even if we leave out the biggest outliers the new approach still needs less than 10 % of the time.
The picture for the increasing computation is even better. There are 19 models where both methods needed less than 1 ms. In 8 models the former method was faster, however, in all of these the difference was only a couple of milliseconds. In the remaining 73 models the new method was faster. Altogether the new approach needed only 0.79 % of the time compared to the former method. And if we, again, leave out the biggest outliers the new approach still needs less than 10 % of the time.
There are only three models, where the new method is slower. In two of them, the difference is only a couple of milliseconds. But for the CloudDeployment-PT-7b model the decrease computation needed more than 38 seconds, while the former approach needed not even one. We thoroughly investigated this case, but we could not find any useful hints, why this model performs completely different than the rest. The model has by far not the most places, transitions, or arcs. Further, the ratio between these three are rather ordinary as well.
However, the general faster computation of the new method makes more than up for this outlier. All but one model needed less than 45 seconds to compute $DI(N)$. Furthermore, almost all models needed not even a single second. The most time is needed for "SharedMemory-PT-200" with 83.6 seconds, however, this is still significantly less time compared to the almost 2 hours the former method needed.

7.6 Discussion

In this chapter, we dealt with our research goal to accelerate the *state space exploration*. For this, we have computed the enabledness information of the transitions in every marking. Doing that in a brute force fashion results in evaluating the enabledness information for all transitions. Since the state space usually consists of a large number of markings, reducing the number

of enabledness tests deserves particular attention.
To this end, we used the locality of transition firing. We introduced the notion of increasing and decreasing transitions which are the sets of transitions that might become disabled or enabled, respectively, if a given transition t is fired. Together, they form the data structure $DI(N)$. Our main contribution is a new and faster way to compute $DI(N)$. Results show that the new method is in almost all cases at least one order of magnitude faster than the former method. In general, the performance of the state exploration increases significantly, because in each marking only a subset of all transitions have to update their enabledness information. Thus, we have successfully reached this research goal and with this, we have also *improved model checking efficiency*.
The new approach addresses mainly the runtime performance. In the future, we also plan to use $DI(N)$ for the reduction of memory used to save the decreasing and increasing transitions for each transition. Storing $DI(N)$ requires a lot of space, in worst case $\mathcal{O}(|T^2|)$. An idea to this end, would be to utilize the observation that some transitions t and t' share a significant amount of transitions that they affect, when firing. Storing $DI(N)$ only for t and just a pointer for t' to t would save memory. Finding such suitable candidate transitions without increasing the runtime significantly is on open issue.
Another open question is based on the observation that expanding $H(N)$ can be reduced to matrix multiplication. Using a fast algorithm, like the Strassen algorithm [117], might improve the performance further.
On the implementation level, two issues remain open. First, the computation of $DI(N)$ can be paralleled based on conflict cluster (CC) (see Definition 2.3.3) of a P/T net [144]. CCs can be determined by a union-find-algorithm [120] with effectively constant amortized time complexity [144] which means the effort to compute them is negligible. Since all CC are independent from each other, every CC can be used to build its own part of $DI(N)$ in parallel. At the end, all parts are merged together. Second, the intermediate directed graph $H(N)$ is similar to the original P/T net. The difference is that transitions and there connecting edges are missing, if the transition has no increasing or decreasing effect on any transition. Another difference is that the arc weight of an incoming and an outgoing edge between two nodes is set off against each other. Therefore, it might be possible for a model checker to build $H(N)$ directly while reading the P/T net.

Chapter 8

Formula simplification

This chapter introduces several smaller *formula simplification* techniques for temporal logic formulas. We have published the following techniques with its results in [79]. The following chapter is based on this publication.
It has already been recognized [7] that formulas should be carefully preprocessed before running a model checking procedure. Because the more atomic propositions (i.e., the more places and transitions) are used in a given formula φ, the more behavior is relevant for the validity of φ, and the less efficient other reduction techniques like partial order reduction [49, 97, 127], symmetry [110] or net reduction [4, 8, 92] become. The goal is to find techniques which reduce the size of φ, while preserving its validity. To use such simplification techniques for real applications, they should be easier to compute than the actual verification based on the state space exploration.
Structural reduction rules simplify the structure of a given model and/or a given formula. Using such rules to simplify the specification can have the effect that some parts of the state space become irrelevant and thus can be pruned. Moreover, some subformulas can become invariantly true or false. Such properties can be replaced in the specification directly with true or false and therefore simplify the entire formula.
Several approaches were proposed in the literature to find such properties. Y. Thierry-Mieg proposed in [123] structural reduction rules that jointly reduce a given model and a set of properties expressed as invariants. In [143] K. Wolf introduced a method for detecting duplicate subformulas in a given specification. In [7] Bønneland et al. proposed several methods based on invariants to significantly reduced the size of the formula. We continue this direction by introducing several approaches to reduce the size of the formula even further.
We propose embedded place invariants and an algorithm to easily compute traps [43] for formula reduction. To simplify the specification further, tau-

tologies can be used. There exist a whole range of CTL* tautologies. Many of them are well known in the literature [71] but not all of them are commonly known. In addition, we propose to shift **X**-operators to the beginning of a given formula and to use Boolean operators in one or the other direction.

We have implemented all proposed techniques as proof-of-concept in LoLA. Experiments using the MCC benchmark from 2018 show that all these techniques *improve model checking efficiency*.

The remainder of this chapter is organized as follows. Section 8.1 proposes several ways to simplify the input formula. Section 8.2 is dedicated to an experimental validation of the proof-of-concept implementation. Section 8.3 concludes the chapter with a discussion.

8.1 Formula simplification

Atomic propositions may turn out to be always true or false, proven by the infeasibility of a linear program that can be derived from the proposition and the Petri net state equation (see Definition 2.3.13). Once a proposition has been identified as true or false, a whole subformula may become irrelevant for the specification. In fact, a significant number of formulas can be evaluated without even running a model checker. For other formulas, the model checking problem may become significantly simpler than the original one. In the remainder of this section, we have next to the reduction of the formula size, two other objectives: First, we want to increase the number of situations where we can apply a special routine, e.g., routines discussed in Chapter 6 and Chapter 11. Second, we want to increase the efficiency of partial order reduction that we introduce in Chapter 9 of this thesis. In the remainder, let $N = (P, T, F, W, m)$ be a P/T net. We propose several ways to simplify the input formula.

Tautologies

Firstly, there exist many tautologies in temporal logic. Not all of them are commonly known. This way, an originally complicated formula may automatically be rewritten to a much simpler query. The formula rewriting system of LoLA [142] for example currently contains more than 100 rewrite rules that are based on CTL* tautologies.

Boolean operators

Some tautologies, such as $\mathbf{AG}\,(\psi \wedge \chi) \iff (\mathbf{AG}\,\psi \wedge \mathbf{AG}\,\chi)$ can be applied in both directions. Applying it from right to left decreases the number of temporal operators. However, the single operator on the left applies to a more complicated subformula $\varphi = \psi \wedge \chi$ and for φ, more transitions are visible, meaning that they can change the truth value of φ if they are fired, than for ψ or χ alone. To have fewer so-called visible transitions in a formula is beneficial for partial order reduction. Applying the tautology from left to right is, thus, meaningful too, to obtain a formula with more temporal operators but each operating on fewer visible transitions. Then, partial order reduction potentially works better. Moreover, this way increases the likelihood that one Boolean operator becomes the root of the subformula tree. In that case, we can verify the particular subformula on its own possibly with a specialized routine from Chapter 6 or Chapter 11. Hence, we propose to use an orientation of tautologies that prefers pushing Boolean operators towards the root of the formula tree.

X-operators

In model checking, the **X**-operator is problematic for some reduction techniques, since formulas containing the **X**-operator are in general not stutter-invariant (see Definition 3.3.1). For example, partial order reduction is not applicable to CTL or LTL formulas containing the **X**-operator. Therefore, we try to push **X**-operators towards the root of the formula tree. To this end, we apply tautologies. For example, we apply $\mathbf{EF}\,\mathbf{EX}\,\varphi \iff \mathbf{EX}\,\mathbf{EF}\,\varphi$ from left to right. This way, we increase the likelihood that we finally obtain one of the formulas considered in Section 11.7 which only contain **X**-operators at the beginning of the formula. Moreover, we get larger subformulas that do not contain an **X**-operator. This can be seen in the example: after the **EX**-operator the subformula on the left side contains only φ, whereas the subformula on the right side contains $\mathbf{EF}\,\varphi$. Since CTL preserving partial order reduction techniques work only on **X**-free formulas, partial order reduction is not applicable to a formula containing an **X**-operator as a whole. However, when an **X**-free subformula of a CTL formula is evaluated on some level of recursion in the model checking procedure sketched in Section 4.3, there is no reason not to apply partial order reduction. Hence, the rewriting strategy improves the applicability of partial order reduction.

Traps

When investigating atomic propositions, Frederik Bønneland et al. [7] mainly employed the Petri net state equation. In quite some situations where the state equation is not able to prove a proposition to be invariantly true or false, a trap can help.

Definition 8.1.1 (Trap).
Let $N = (P, T, F, W, m)$ be a P/T net. The set $Q \subseteq P$ is called trap *if $Q^\bullet \subseteq {}^\bullet Q$.*

Hence, a trap is a set Q of places that, if it once contains a token, it always contains a token. This is formally established by requiring that every transition that consumes tokens from any place in Q, also produces a token on some place in Q. For example, consider an atomic proposition $k_1p_1 + \cdots + k_np_n \geq 1$ with $n = |P|$, $p_i \in P$, and all k_i being positive for $1 \leq i \leq n$. If $\{p_1, \ldots, p_n\}$ contains a trap that has at least one token in the initial marking, then the proposition is invariantly true. We show with the following procedure that detecting a trap in $\{p_1, \ldots, p_n\}$ is easy. We start with $\{p_1, \ldots, p_n\}$ and remove every place p in $t^\bullet$ of some transition t that does not produce tokens on any $p_1, \ldots, p_n$. This way, leaves the maximal trap included in $\{p_1, \ldots, p_n\}$. For more information how to use traps for model checking, the reader is referred to [43].

Embedded place invariants

Sometimes, place invariants (see Definition 2.3.16) can be used for simplifying atomic propositions. Place invariants can be found by solving the system of equations $i \cdot C = 0$, where C is the incidence matrix of N.

Lemma 8.1.1 (Weighted sum of tokens [74]). *Let $N = (P, T, F, W, m)$ be a P/T net. If i is a place invariant, then the equation $i(p_1)m(p_1) + \cdots + i(p_n)m(p_n) = im_0$ holds for all reachable markings in N with the set of places $\{p_1, \ldots, p_n\}$.*

A place invariant i assigns a weight $i(p)$ to every place p such that the weighted sum of tokens remains constant for all reachable markings. We propose to systematically compute helpful invariants as the solution of a linear program LP. Consider an atomic proposition of the shape $k_1p_1 + \cdots + k_mp_m + l_1q_1 \cdots + l_nq_n \; op \; k$, where all p_j and q_j are places, all k_j are positive integers, all l_j are negative integers, $op \in \{=, \neq, <, >, \leq, \geq\}$, and k is an integer. LP looks for the largest possible invariant where the coefficients are between 0 and k_i (respectively l_i): Maximize $i(p_1) + \cdots + i(p_m) - i(q_1) - \cdots -$

$i(q_n)$ where $C^T i = 0$, and $0 \leq i(p_j) \leq k_j$ (for $1 \leq j \leq m$), and $l_j \leq i(q_j) \leq 0$ (for $1 \leq j \leq n$). If LP is feasible, subtracting the resulting solution from the atomic proposition may or may not lead to less mentioned places but is guaranteed not to add places to the formal sum of the proposition.
Now, consider, as an example, the proposition $p_1 + 2p_2 + p_3 \geq 2$ and assume that there is a place invariant that yields the equation $p_1 + p_2 = 1$. Then the atomic proposition can be simplified to $p_2 + p_3 \geq 1$. It is not constant but does no longer mention p_1. Consequently, the set of visible transitions, meaning transitions that can change the truth value of the inspected formula, may become smaller since the environment of p_1 does no longer need to be considered as visible (unless transitions still appear in the environment of p_2 or p_3). With a smaller set of visible transitions, better partial order reduction may be expected (in particular for the VIS principle, which will be introduced later in Definition 9.2.4).

8.2 Experimental validation

We implemented the methods discussed in this chapter in our proof-of-concept model checker LoLA. For the evaluation of the methods, we used the benchmark provided by the MCC 2018 [66]. The benchmark consists of 767 P/T nets. We used the formulas provided in the CTL category. For every net, 32 CTL formulas are provided. This results in 24544 individual verification problems.
Experiments were executed on a machine with 32 physical cores running at 2.7 GHz and 1 TB of RAM. All computations were done with no time and memory restrictions.
For 3704 problems (15.1 %), the initial rewriting process yielded a formula that does not contain any temporal operator. Here, sufficiently many atomic propositions have been found to be invariantly true or false due to the formula simplifications described in this section and in [7]. Resulting formulas can be evaluated by just inspecting the initial marking, so no actual run of a model checker is necessary. This relieves a CTL model checker significantly.

8.3 Discussion

We proposed to relieve the model checking algorithms by providing formula simplification. We implemented all proposed techniques in our proof-of-concept model checker LoLA. Experiments show that 15.1 % of all queries could be solved using the introduced methods. Thus, we successfully handled

two of our research challenges: *formula simplification* and *improving model checking efficiency*.
Designing structural reductions is an active research area, because structural reduction is a powerful method to reduce the size of the specification (and the model). One fact that highlights the power of structural reduction techniques is that LoLA (with formula simplification) scored in the reachability category of the MCC 2019 behind TAPAAL and the same LoLA combined with the structural reduction techniques of ITS-Tools proposed by Thierry-Mieg in [123] scored in the MCC 2020 as ITS-LoLA better than TAPAAL. Therefore, finding additional structural reductions is ongoing work.

Part IV

Partial order reduction

This part is solely concerned with partial order reduction (POR) techniques to tackle the state explosion problem for certain classes of temporal logic. POR is based on the observation that concurrent and independent running processes of the model contribute extensively to the state explosion problem, while having only little influence on the property preservation of individual processes.
There exist several instances of partial order reduction: *ample sets* [97], *persistent sets* [49], and *stubborn sets* [127]. The essence of all of them is, while building the state space, to compute for every state a subset of transitions, called *aps set*, and only fire those transitions to reach other states. This reduces the state space.
When using partial order reduction there exist several different approaches to preserve entire classes of properties, such as CTL [48] or LTL [129]. Furthermore, several methods from the literature preserve only certain properties, e.g., deadlocks [127], reachability [70], or liveness and other standard properties [109]. For an introduction and an exhaustive overview of POR methods, the reader is referred to [130].
Partial order reduction appears to be a powerful reduction technique. Our explicit model checker LoLA [142] solves around 70 % of the queries in the reachability category of the MCC benchmark without and around 90 % with partial order reduction techniques. Thus, POR allows to solve ⅔ of the 30 % really hard-to-solve queries. In addition, queries that can be solved without POR, are usually solved faster and with a smaller memory footprint if POR is applied. Thus, it makes sense to push the theoretical limits of POR to reach as good reduction as possible.
The remainder of this part is organized as follows. Chapter 9 recalls the basic notions of the stubborn set method, which we will use in the following chapters. Chapter 10 is dedicated to the introduction of new automata-based stubborn sets for LTL. In chapter 11, we present stubborn sets for special CTL formulas.

Chapter 9

The stubborn set method

For self-containedness, we recall the basic notions of the stubborn set method from the literature in this chapter. More precisely, we repeat the basic principles used for stubborn sets and the selection of principles used for property preservation. We use these notions in the next two chapters where we introduce our stubborn set approaches for LTL and CTL, respectively. The chapter is based on our publications [76] and [79].

Partial order reduction is a technique to reduce the state space size of systems, while preserving certain properties. The stubborn set method [126, 127, 130] is an instance of partial order reduction. Other instances of partial order reduction are *ample sets* [97] and *persistent sets* [49]. The essence of all of them is, while building the state space, to compute for every state a subset of transitions, called *aps set*, and only fire those transitions to reach other states. More precisely, let N be a P/T net and φ a property, the partial order method aims at producing a subgraph G' of the reachability graph G of N such that the evaluation of φ on G' yields the same value as on G. To this end, a set aps(m) of transitions is assigned to every marking m, and only enabled transitions in aps(m) are fired to explore G'.

In the remainder, we are concerned with the stubborn set approach. The reason for this is that stubborn sets push the theoretical limits to reach "as good reduction as possible, while ample and persistent sets have favored straightforward easily implementable conditions and algorithm" [132]. For more details regarding this subject the reader is referred to [132].

Over the years, a consistent systematic has emerged for presenting stubborn set methods. There is a list of *principles* that governs the selection of stubborn sets. Each principle comes with an *algorithmic approach* for the computation of a stubborn set that obeys that principle. Finally, there is a list of *results* stating that, if G' is computed using stubborn sets that meet some selection of principles, all properties of a certain class of properties are

preserved. We only list principles and results that we need for our considerations below. Our focus here is not stubborn set theory as such but LTL and CTL model checking technology. For this reason, we selected principles that are more understandable. For stronger results on stubborn sets, the reader is referred to [48, 70, 130, 132]. We will also completely skip the algorithmic approaches as they are not necessary for our argumentation. For implementation details, the reader is referred to [137].
In this chapter, we start with a motivating example in Section 9.1. In Section 9.2, we formalize and systematize the principles used for stubborn sets. Section 9.3 is concerned with the selection of principles to preserve certain classes of properties. In Section 9.4, we show how to use our data structure $DI(N)$ from Chapter 7 for a faster computation of stubborn sets.

9.1 Motivational example

Let $N = (P, T, F, W, m_0)$ be a P/T net with $T = \{t_1, t_2\}$ and the reachability graph TS illustrated in Figure 9.1. We want to check if marking m_3 satisfies some property φ. There are two independent paths t_1t_2 and t_2t_1 in TS that both reach m_3. This means, if only the reachability of m_3 counts, then the set of transitions to explore in m_0 can be reduced to either t_1 or t_2, respectively. Here we choose t_1, which is the blue solid path in Figure 9.1. Since we do not follow t_2, highlighted by the red dashed path, we reach the intermediate marking m_1 but not m_2. The exploration continues from m_1 with the only enabled transition t_2 and reaches m_3. Because m_2 is never explored the state space is reduced.

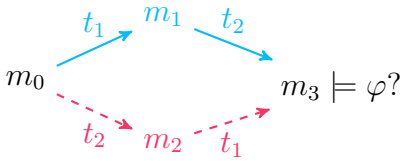


Figure 9.1: Verify property φ in m_3 in the reachability graph of N.

9.2 Principles

In this section, let $N = (P, T, F, W, m_0)$ be an arbitrary, fixed P/T net with reachability graph (M, E) and φ a property. The stubborn set method aims at producing a small subgraph (M', E') of the reachability graph (M, E), such that the evaluation of φ on (M', E') yields the same truth value as on

(M, E). In most cases, we assume that there is a path π in (M, E) (e.g., a witness path or a counterexample) and show that (M', E') contains a path π' that is equally fit with respect to φ. We first define a mechanism to restrict the set of transitions in each marking to a so-called *stubborn set*.

Stubborn set generator.
A function stub : $M \to 2^T$ is called *stubborn set generator* and produces the reduced reachability graph (M', E') such that $(m, m') \in E' \iff m \xrightarrow{t} m'$ for a transition $t \in \text{stub}(m)$. M' is the set of markings, which can be reached from the initial marking by only firing transitions from stub(m) in marking m.

With this, the stubborn set generator allows us to select any subset of transitions in any marking without any restrictions. This is certainly not useful if we want to verify a given property, because a random choice would most likely not preserve the studied property. The goal is to reduce the state space as much as possible, while choosing the subset in a way that the property under investigation is preserved. For this, a set of *principles* (requirements) were introduced in the literature. Each requirement has a specific purpose for property preservation. We first introduce the principles, before we show how to preserve properties in the next section.
The first principle is *commutativity*, which is at the heart of all POR methods.

Definition 9.2.1 (COM: The commutativity principle).
For a stubborn set generator stub : $M \to 2^T$ *and a marking* $m \in M$, stub$(m) \subseteq T$ *satisfies the* commutativity principle *(COM for short) if, for all* $\omega \in (T \setminus \text{stub}(m))^*$ *and all* $t \in \text{stub}(m)$, $m \xrightarrow{\omega t} m'$ *implies* $m \xrightarrow{t\omega} m'$.

The main purpose of COM is that π' may execute transitions in another order than π. The firing of transitions, outside of the stubborn set, can be postponed. Figure 9.2 illustrates this situation. After firing transition t in marking m and reaching m_1, it is still possible to fire ω in m_1. In the end, both paths $t\omega$ and ωt lead from m to m_3.

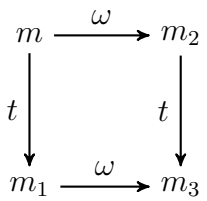


Figure 9.2: Graphical representation of COM.

The next requirement is the *key transition principle*.

Definition 9.2.2 (KEY: The key transition principle).
For a stubborn set generator stub : $M \to 2^T$ *and a marking* $m \in M$, $\text{stub}(m) \subseteq T$ *satisfies the* key transition principle *(KEY for short) if* m *does not enable any transition or it contains a transition* t' *(a* key transition*) such that, for all* $\omega \in (T \setminus \text{stub}(m))^*$, $m \xrightarrow{\omega} m'$ *implies that* t' *is enabled in* m'.

KEY ensures that there is an enabled transition that can be switched to the beginning of a transition sequence. The purpose of KEY (in connection with COM) is that π' may contain transitions that are not occurring in π. In more detail, KEY states that there is a transition t' in the stubborn set which is enabled before and after firing a sequence ω outside of the stubborn set. This is illustrated in Figure 9.3.

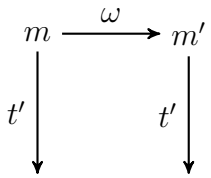


Figure 9.3: Graphical representation of KEY.

Before we continue with the next requirement, we introduce the *invisibility property*.

Definition 9.2.3 (Invisibility property).
A transition t *is* invisible *with respect to a formula* φ *regarding a set of transitions* $T' \subseteq T$, *if*

$$(m \xrightarrow{\omega} m' \wedge m \xrightarrow{t\omega} m'') \Longrightarrow (m' \models \phi \iff m'' \models \phi)$$

holds for all atomic propositions ϕ *occurring in* φ, *all markings* $m \in M$, *and all finite sequences* $\omega \in (T')^*$. *Otherwise, the transition* t *is called* visible *with respect to* φ *regarding* T'.

Transitions that are able to change the truth value of an atomic proposition ϕ occurring in φ are visible. Transitions that can not change any ϕ are invisible. This means, firing a sequence of invisible transitions will have no effect on the truth value of φ. Using the invisibility property, we are able to introduce the *visibility principle*.

Definition 9.2.4 (VIS: The visibility principle).
For a stubborn set generator stub : $M \to 2^T$ *and a marking* $m \in M$,

stub$(m) \subseteq T$ *satisfies the* visibility principle *(VIS(φ) for short) for a property* φ *if* stub(m) *contains only invisible transitions with respect to* φ *regarding* $(T \setminus \text{stub}(m))$*, or all transitions.*

The main purpose of VIS(φ) is that if visible transitions appear in π', then they appear in the same order as in π. VIS(φ) ensures the same ordering of transitions, but it can introduce stuttering (see Definition 3.3.2). As an example, consider the property $\varphi = \mathbf{F}(p_2 > 0)$ which should hold in the P/T net of Figure 9.4. Transition t_1 can change the truth value of φ. More precisely, if t_1 fires, it produces one token on p_2 and with this, φ is satisfied. This means, there are two valid stubborn sets in m_0. The first one is $\{t_2\}$ and contains only invisible transitions. The second one is $\{t_1, t_2\}$. If we choose $\{t_2\}$ as stubborn set, we reach the same marking m_0 again after firing t_2. We could now infinitely often choose t_2 as stubborn set. This results in infinite stuttering, while we ignore t_1 indefinitely.

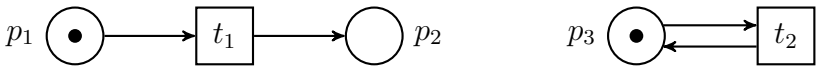


Figure 9.4: Example P/T net for the graphical representation of VIS(φ) and IGN.

To avoid infinite stuttering, we introduce the *non-ignoring principle.*

Definition 9.2.5 (IGN: The non-ignoring principle).
A stubborn set generator stub $: M \to 2^T$ *satisfies the* non-ignoring principle *(IGN for short) if every cycle in the reduced reachability graph contains a marking where all enabled transitions are explored.*

IGN is used to ensure that all transitions of π are eventually occurring in π'. The concept behind IGN is that if a marking m is the start of a cycle in the reachability graph, then in at least one marking m' in the cycle all enabled transitions have to be explored. In other words, stub(m) consists of all enabled transitions in m' that can leave the cycle.
The *branching principle* is concerned with the preservation of the correct branching structure of a CTL formula.

Definition 9.2.6 (BRA: The branching principle).
For a stubborn set generator stub $: M \to 2^T$ *and a marking* $m \in M$*,* stub(m) *satisfies the* branching principle *(BRA for short) if* stub(m) *contains a single enabled transition, or all enabled transitions of* m*.*

BRA is applied to ensure that visible transitions are not swapped with branches in the state space other than branches that are introduced by concurrency. The branching principle enables reduction only in markings where just one (invisible) enabled transition is sufficient to meet all other principles. The last principle we are considering is state oriented and is called the up-set principle.

Definition 9.2.7 (UPS: The up-set principle).
For a marking m and a CTL property φ such that $m \not\models \varphi$, U is an up-set if every path from m to a marking that satisfies φ contains an element of U. For a stubborn set generator $\text{stub} : M \to 2^T$ *and a marking* $m \in M$, $\text{stub}(m)$ *satisfies the* up-set principle *with respect to φ if $m \models \varphi$ or $U \subseteq \text{stub}(m)$, for some up-set U.*

The up-set principle is useful in the following situation: Let us consider that we have a set M of states. Then the up-set principle should preserve M in the sense that the reduced state space contains elements of M if and only if the original state space does. For a particular marking m, this means that if a marking m' is reachable from m in the full reachability graph, then at least one transition sequence leading from m to $m'' \in M$ should be part of the reduced reachability graph. This means, the up-set principle ensures that the stubborn set at m will always contain a transition of π.
For example, consider the property $\varphi = p < k$. If we are in a marking m and want to reach $\{m' | m' \models \varphi\}$, then the up-set consists of all transitions in the post-set of p, because these transitions are the only ones that can reduce the number of tokens on p to satisfy φ.
There are some more principles, but they are not needed for this thesis. See [48, 70, 130, 132] for the details.

9.3 Property preservation

Combining the principles from the previous chapter in a specific way, results in the preservation of certain property classes. In this section, let φ be the investigated property, $N = (P, T, F, W, m_0)$ an arbitrary, fixed P/T net with reachability graph (M, E), and (M', E') the reduced reachability graph.
The first property we look into is the *deadlock* property.

Proposition 9.3.1 (Preservation of deadlocks, [127]). *If the principles COM and KEY are satisfied then (M', E') contains all deadlocks and at least one infinite path of the original reachability graph.*

The next property we consider are terminal strongly connected components. For the preservation of TSCCs, we need to avoid infinite stuttering and therefore non-ignoring is needed.

Proposition 9.3.2 (Preservation of TSCCs, [131]). *If the principles COM, KEY, and IGN are satisfied then (M', E') contains at least one marking of every TSCC of the original reachability graph.*

For the preservation of an LTL formula φ, we need COM, KEY, IGN, and the visibility principle. Also, we have to restrict LTL to stutter-invariant formulas.

Proposition 9.3.3 (Preservation of $\mathrm{LTL}_{-\mathbf{X}}$, [97, 130]). *Let φ be an LTL property not using the $\mathbf{X}$-operator. If the principles COM, KEY, IGN, and VIS(φ) are satisfied then φ is preserved.*

We call the previous proposition the *conventional $LTL_{-\mathbf{X}}$* POR method.
To preserve a CTL property φ, we add the branching principle to the mix.

Proposition 9.3.4 (Preservation of $\mathrm{CTL}_{-\mathbf{X}}$, [48]). *Let φ be a CTL property not using the $\mathbf{X}$-operator. If the principles COM, KEY, IGN, VIS(φ), and BRA are satisfied then φ is preserved.*

To preserve reachability, we only need COM and the up-set principle. This is because UPS preserves target markings for (M', E') if and only if they are reachable in (M, E).

Proposition 9.3.5 (Preservation of reachability, [70, 109]). *Let φ be a CTL formula without temporal operators. If the principles COM and UPS(φ) are satisfied then $\mathbf{EF}\,\varphi$ is preserved.*

9.4 Using $DI(N)$ for stubborn set computations

Even though, the focus in this thesis does not lie on implementing stubborn sets, we still show how to speed up the computation of stubborn sets with the data structure $DI(N)$ introduced in Chapter 7
In practice, there are basically two methods to compute stubborn sets, although there exist some more theoretical approaches [137]. The deletion method [128, 137] starts with all transitions as stubborn candidate and then iteratively deletes unnecessary transitions, until the set is minimal. Here, minimal means that no subset of its enabled transitions, fulfilling all requirements, can be the set of enabled transitions of any other possible stubborn set. The second one is the incremental method [126, 137], which starts with

a single transition as stubborn candidate and adds more transitions until a proper stubborn set is found.
The deletion method has its strength when applied to negative (not reachable) formulas, because the resulting state space is in general smaller. On the other hand, goal-oriented stubborn sets [70] based on the incremental method tend to perform better in case of positive (reachable) formulas. The reason for this is that this version of stubborn sets can be computed extremely fast and thus, many more states can be explored in a certain time [141]. In addition, the goal-orientation, introduced by UPS steers the state space exploration towards the witness state faster and produces short witness paths in many examples. Experiments have shown that the performance of goal-oriented stubborn sets is in general better than the performance of other methods [141].
For the computation of goal-oriented incremental stubborn sets, $DI(N)$ can be used for acceleration. More precisely, to differentiate between transitions in and outside of a stubborn set stub(m) of a marking m, which is necessary to satisfy the commutative principle, the set of *conflicting transitions*, defined as $\{({}^\bullet t)^\bullet\}$ regarding transitions $t \in \text{stub}(m)$, have to be computed. For this, the closure or the marking-dependent relation [132] between transitions needs to be built.

Definition 9.4.1 (Closure of transitions [136]).
Let $N = (P, T, F, W, m_0)$ be a P/T net, $t \in T$ a transition, m a marking, and stub(m) *a stubborn set in m.*

- *If t is enabled in* stub(m)*, then include every so-called* conflicting transition $t' \in ({}^\bullet t)^\bullet$ *into* stub(m) *as well.*
- *If t is disabled in* stub(m)*, then choose $p_t \in {}^\bullet t$ such that $m(p_t) < W(p_t, t)$ and include every $t' \in {}^\bullet p_t$ into* stub(m) *as well.*

Note, if more than one p_t with $m(p_t) < W(p_t, t)$ exist, only one is randomly chosen. The performance does not depend on the choice [136]. The set of conflicting transitions for a transition t is frequently needed in stubborn set computations. Every time we add an enabled transition to the stubborn set, we have to check if there are conflicting transitions. However, the conflicting transitions can be directly read from $DI(N)$ and therefore, after $DI(N)$ has been preprocessed, the computational time is greatly reduced.

Chapter 10

Automata-based partial order reduction for LTL

In the previous chapter, we recalled stubborn set theory as an instance of partial order reduction from the literature. This chapter introduces our new Büchi automata-based POR for LTL. It is a generalization and extension of ideas proposed in [64] and [75]. Some remarks in this chapter are inspired by [64]. We have published this method with its results in [76]. The following chapter is based on this publication.
In conventional LTL model checking with POR the state space is first reduced, and then together with the Büchi automaton B of the (negated) formula a product automaton B^* is built. The actual verification is then carried out in B^*. In [75] Lehmann et al. proposed to first build B^* with the original state space and subsequently reduce B^* with the additional information available from B. To the best of our knowledge, the idea of using information from the Büchi automaton to reduce the state space was first presented by Peled et al. in [98]. However, the idea has received little attention since then. We propose a new automata-based stubborn set method that uses the additional information available from B. The main idea is to focus the reduction of B^* to the current Büchi state q, i.e., all considerations regarding the formula are done locally around q. This gives rise to the main principle we use to achieve additional reduction power: all transition sequences, which do not use transitions from the stubborn set, cannot leave the current Büchi state. As long as the outgoing formulas from q are not fulfilled, B and thus B^* remain in q and it can be reduced. With this, we are able to weaken or drop several requirements used in conventional LTL POR. Often, Büchi automata only consist of very few states compared to the number of states of the system. The formulas in the MCC, although artificial, rarely have more than 5 – 10 Büchi states. This is also consistent with our experience of real

world use cases using our model checker LoLA [142].
The remainder of this chapter is organized as follows. Section 10.1 updates the list of principles. We introduce a new principle and drop or weaken some existing ones. Section 10.2 continues with the introduction of our new automata-based stubborn set method. In Section 10.3, we show the effectiveness of the newly introduced method with several examples. We analyze the reduction power and compare it with the reduction efficiency of conventional POR. We conclude this chapter with some remarks regarding related work and some thoughts on future work in Section 10.4.

10.1 Updated principles

In this chapter, let $N = (P, T, F, W, m_0)$ be an arbitrary, fixed P/T net with reachability graph (M, E), and $B = (Q, Q_0, \delta, \lambda, Q_F)$ be the Büchi automaton of a negated input LTL formula φ. Remember that Proposition 4.4.2 stated that φ can be transformed to B and that B accepts exactly those paths which violate φ. B has at most 2^{φ} states [134].
A stubborn set generator used in conventional $\text{LTL}_{-\mathbf{X}}$ POR is a function $\text{stub} : M \to 2^T$. This means, the state space of a P/T net is reduced first and then the product automaton B^* is built with the reduced state space. Compared to this, Lehmann et al. [75] proposed to use B for the stubborn set generator, too. Therefore, they change the stubborn set generator. Then, $\text{stub} : M \times Q \to 2^T$. The extra information from B can be used to weaken some stubborn set requirements. With weaker requirements, we have more options to choose the respective transition set and thus, the chance of better reduction. The procedure is to first build the product automaton on-the-fly with the full reachability graph and then to reduce the product automaton with the additional information available from the Büchi automaton.
However, the stubborn set method from Lehmann et al. was only able to solve a certain class of Büchi automata, namely elementary Büchi automata. Elementary Büchi automata consist of a non-branching sequence of states, which may or may not have self loops and in addition, the last state is a final state. Although there are interesting properties in this class, such as $\mathbf{G}(\varphi \implies \mathbf{F}\,\psi)$, it is still a restriction.
As starting point for our following considerations, we use the stubborn set generator $\text{stub} : M \times Q \to 2^T$ and conventional LTL POR (see proposition 9.3.3) that uses the principles COM, KEY, IGN, and VIS(φ) to preserve $\text{LTL}_{-\mathbf{X}}$. Our main idea is to restrict the scope of the formula under investigation to the current Büchi state q. All considerations regarding the formula are done locally around q.

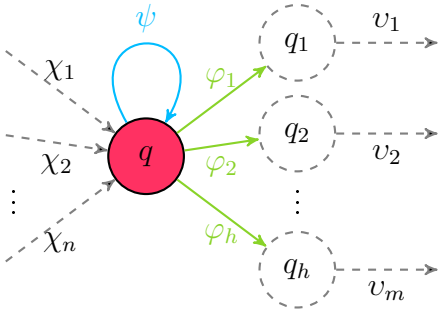


Figure 10.1: The scope of the formula is restricted to the current Büchi state q, its retarding formula ψ, and its progressing formulas $\varphi_i, i \in [1, h]$.

To illustrate this, let us consider the Büchi automaton from Figure 10.1. Assume we are in state q. We restrict the scope of the formula to:

1. the self loop ψ leading from q to q, and
2. the progressing (outgoing) formulas $\varphi_i, i \in [1, h]$ leading from q to new Büchi states q_i.

When we are in q, we do not care about all the other parts of the formula, i.e., $\chi_j, j \in [1, n]$, and $\upsilon_k, k \in [1, m]$. This leads us to the main principle for the reduction. We call it the *non-leaving principle*:

> All transition sequences, which do not use transitions from the stubborn set, cannot leave the current Büchi state.

Let us formalize this. The formula φ can consist of several atomic subformulas. Let $q \in Q$ be a Büchi state and $\psi = \lambda(q, q)$ be the formula of the self loop $q \to q$, which we call the *retarding formula*. Furthermore, let h be the number of outgoing arcs from q that are progressing to new Büchi states, and let $\varphi_i, i \in [1, h]$ be the formulas of the outgoing arcs, which we call *progressing formulas*. Now we are able to formalize the main principle for the reduction.

Definition 10.1.1 (NLG: The non-leaving principle).
For a stubborn set generator $\text{stub} : M \times Q \to 2^T$*, a marking* $m \in M$ *and a Büchi state* $q \in Q$*,* $\text{stub}(m, q) \subseteq T$ *satisfies the* non-leaving principle *(NLG for short) if* $m \xrightarrow{\omega} m' \implies m' \not\models \varphi_i$ *for all* $i \in [1, h]$ *and for all* $\omega \in (T \setminus \text{stub}(m, q))^+$.

It holds that no transition sequence, from the set of transitions which are outside of the stubborn set, can satisfy a progressing formula. For example,

consider the progressing formula $\varphi_i = p < 1$, which means that place p must have less than one token on it. Then each transition t that is in the post-set of p must be part of the stubborn set, since t can remove tokens from p and thus might change the truth value of φ_i from false to true.
As long as the outgoing formulas from q are not fulfilled, the Büchi automaton and thus the product system remain in state q and it can be reduced. With NLG some requirements from the conventional LTL POR can be weakened or even dropped, resulting in several advantages of the new stubborn set method.

Invisibility principle

Since all considerations regarding the formula φ are done locally around the current Büchi state, all other parts of φ can be ignored. Hence, the invisibility property can be simplified. In the general version of the property, the influence of atomic subformulas ϕ regarding the result of φ is dependent on the temporal progression and not further known. This means, the truth value of ϕ must be preserved in both directions, $(m \models \phi \iff m' \models \phi)$. In the automata-based reduction, nevertheless, all temporal operators are expressed in the state transitions of B and with this, the influence of ϕ is determined. A satisfied formula allows a progression to another state q', however, it has no influence on other state transitions. Thus, we can simplify the invisibility property to an "uni-directional" implication, $(m \models \phi \implies m' \models \phi)$. This means, a transition, which changes ϕ from true to false, is not allowed, but a transition, which makes ϕ only "more true", is admissible.

Definition 10.1.2 (Semi-invisibility property).
A transition $t \in T$ is called semi-invisible *with respect to an LTL formula φ regarding a set of transitions $T' \subseteq T$, if*

$$(m \xrightarrow{\omega} m' \wedge m \xrightarrow{t\omega} m'') \implies (m' \models \phi \implies m'' \models \phi)$$

holds for all markings $m \in M$, all finite transition sequences $\omega \in (T')^*$ and all atomic subformulas ϕ of φ. Otherwise the transition t is semi-visible with respect to ϕ regarding T'.

As a consequence, we can weaken the visibility principle and introduce the *semi-invisibility principle.*

Definition 10.1.3 (S-INV: Semi-invisibility principle).
For a stubborn set generator $\mathrm{stub} : M \times Q \to 2^T$*, a marking $m \in M$ and a Büchi state $q \in Q$,* $\mathrm{stub}(m, q) \subseteq T$ *satisfies the* semi-invisibility principle *(S-INV for short) for an LTL formula φ, if all enabled transitions $t \in \mathrm{stub}(m, q)$*

are semi-invisible with respect to φ regarding $(T \setminus \text{stub}(m, q))$, or all transitions $t \in (T \setminus \text{stub}(m, q))$ are semi-invisible with respect to φ regarding the empty set.

Non-ignoring principle and LTL$_{-\mathbf{X}}$

Since the reduction is only applied within a Büchi state q, stuttering becomes irrelevant. Finite stuttering within q does not change the accepting behavior, and infinite stuttering can always be avoided, if this was possible in the original product automaton due to the reduction principle. Consequently, the non-ignoring principle can be dropped. In addition, due to the irrelevance of stuttering, the restriction to LTL$_{-\mathbf{X}}$ can be dropped as well. Our new method preserves the full class of LTL. Because of the additional information available from B and because the reduction is only applied to q, we know if q is transient or not. In fact, transient Büchi states can only appear, if an atomic proposition is checked without any temporal operators, or as representatives of **X**-operators of φ.

Key-transition principle

In accepting states, infinite stuttering is possible and even desired. However, it always needs to be ensured that there exists an enabled transition to prolong the path to an infinite path. Hence, KEY has to hold only in accepting states. If the current Büchi state q is not accepting, then only transition sequences from the stubborn set can leave q. And since q is not an accepting state, this implies that we will change the Büchi state at some point in the future of the considered path. It follows, for non-accepting states that KEY does not have to hold, because each transition sequence leaving the current Büchi state contains a transition from the stubborn set. This follows directly from the non-leaving principle.

To get the advantage of the weakened or dropped requirements, we have to uphold the main principle, that all transition sequences, which do not use transitions from the stubborn set, cannot leave the current Büchi state. As long as the progressing formulas are not satisfied, we are staying in the same Büchi state and can reduce further. This is also useful since the Büchi automaton for the LTL formula is usually very small, compared to the transition system.

10.2 Automata-based stubborn sets for LTL

After we have introduced the main ideas for our new automata-based stubborn set method for LTL, we can present the main result of this chapter.

Theorem 10.2.1 (Büchi automata-based partial order reduction)**.** *Given is a Büchi automaton* $B = (Q, q_s, \delta, \lambda, Q_F)$ *and a P/T net* $N = (P, T, F, W, m_s)$ *with reachable markings* M*, let* $\text{stub} : M \times Q \to 2^T$ *be a stubborn set generator with the following properties:*

1. $\text{stub}(m, q)$ *satisfies COM;*
2. $\text{stub}(m, q)$ *satisfies S-INV with respect to* $\psi = \lambda(q, q)$*;*
3. $\text{stub}(m, q) = T$ *or* $\forall t \in \text{stub}(m, q) : m \xrightarrow{t} m' \implies m' \models \psi$*;*
4. $\text{stub}(m, q)$ *satisfies NLG;*
5. *If* $q \in Q_F$*, then* $\text{stub}(m, q)$ *satisfies KEY.*

There exists an infinite accepting path in the reduced product system $\underline{\mathcal{P}}$ *(see Definition 4.4.5), which is generated by* stub*, if and only if there exists an infinite accepting path in the original product system* $\mathcal{P}$*.*

Proof. $\implies$: The reduced state space $\underline{\mathcal{P}}$ is a subsystem of the original state space $\mathcal{P}$. This means, accepting paths in $\underline{\mathcal{P}}$ trivially imply the same accepting paths in $\mathcal{P}$.
$\impliedby$: Let $\pi \in (M \times Q)^\omega$ be an infinite accepting path in $\mathcal{P}$. Assume $\underline{\mathcal{P}}$ has no infinite accepting path. We can separate π in $\pi_1\pi_2$ such that π_1 is the largest prefix that can be executed in $\underline{\mathcal{P}}$. Let (m, q) be the last state in π_1. Furthermore, let $(m_i, q_i), i \in \mathbb{N}$ be the state sequence in π_2 and $t_0 t_1 \ldots$ the sequence of transitions that induce π_2 from (m, q) such that $m \xrightarrow{t_0 \ldots t_i} m_i$.

$$\underbrace{q_s \longrightarrow (m_s, q_s) \longrightarrow (m, q)}_{\pi_1} \underbrace{\xrightarrow{t_0} (m_0, q_0) \xrightarrow{t_1 \ldots t_i} (m_i, q_i)}_{\pi_2}$$

Since (m, q) is the last state in π_1, it follows that $t_0 \notin \text{stub}(m, q)$. Because $t_0 \notin \text{stub}(m, q)$, requirement 4 (NLG) implies that $q = q_0$. We consider two cases. First, q is part of the accepting set of B and we do not change the Büchi state any more. Second, the Büchi state is changed at least once more on the remaining path.
Case 1: $\forall i \in \mathbb{N} : q_i = q$*:*

All states in π_2 remain in the same Büchi state q, i.e., $\forall i \in \mathbb{N} : q_i = q$. This means, q is in the accepting set of B and all markings in π_2 satisfying the retarding formula, i.e., $\forall i \in \mathbb{N} : m_i \models \psi$.

$$\underbrace{q_s \longrightarrow (m_s, q_s) \longrightarrow (m, q)}_{\pi_1} \underbrace{\xrightarrow{t_0} (m_0, q) \xrightarrow{t_1 \dots t_i} (m_i, q)}_{\pi_2}$$

We consider two sub-cases. First, we assume that there exists a transition t_i in $\text{stub}(m, q)$. And second, we assume that there is no such t_i in $\text{stub}(m, q)$.
Case 1.1: *$\exists t_i \in \text{stub}(m, q)$:*
Assume there is a transition $t_i \in \text{stub}(m, q)$. We choose the smallest such i, which means that $m \xrightarrow{t_0 \dots t_i} m_i$ with $t_i \in \text{stub}(m, q)$ and $t_0, \dots, t_{i-1} \notin \text{stub}(m, q)$. Using requirement 1 (COM), we switch t_i to the front of the path and it holds that $m \xrightarrow{t_i t_0 \dots t_{i-1}} m_i$.

$$\underbrace{q_s \longrightarrow (m_s, q_s) \longrightarrow (m, q)}_{\pi_1} \underbrace{\xrightarrow{t_i t_0 \dots t_{i-1}} (m_i, q)}_{\pi_2}$$

The following two cases show that requirement 4 (NLG) implies that all transitions $t_0, \dots, t_{i-1}$ are satisfying the retarding formula ψ.
Case 1.1.1: *t_i is semi-invisible with respect to ψ:*
Requirement 2 (S-INV) ensures, if t_i is semi-invisible regarding ψ, then all states along the new path $t_i t_0 \dots t_{i-1}$ are satisfying ψ.
Case 1.1.2: *$m \not\models \psi$:*
Otherwise, requirement 2 (S-INV) states that all transitions $t_0, \dots, t_{i-1}$ are semi-invisible regarding ψ and together with the fact that requirement 3 ensures that t_i satisfies ψ, all states along the new path $t_i t_0 \dots t_{i-1}$ are satisfying ψ.
In both cases 1.1.1 and 1.1.2 the path π_1 is extended by one state and has an infinite accepting continuation in $\mathcal{P}$.

$$\underbrace{q_s \longrightarrow (m_s, q_s) \longrightarrow (m, q) \xrightarrow{t_i} (m', q)}_{\pi_1} \underbrace{\xrightarrow{t_0 \dots t_{i-1}} (m_i, q)}_{\pi_2}$$

Case 1.2: *$\nexists t_i \in \text{stub}(m, q)$:*
Assume there is no transition $t_i \in \text{stub}(m, q)$. Since q is in the accepting set of B, requirement 5 (KEY) ensures that there exists a key-transition $t' \in \text{stub}(m, q)$ such that $m \xrightarrow{t' t_0 \dots}$ is executable in N.

$$\underbrace{q_s \longrightarrow (m_s, q_s) \longrightarrow (m, q)}_{\pi_1} \underbrace{\xrightarrow{t'} (m^*, q) \xrightarrow{t_0 \dots}}_{\pi_2}$$

It follows the same argumentation as in cases 1.1.1 and 1.1.2. The following two cases show that requirement 4 (NLG) implies that $t_0, \ldots$ are all satisfying the retarding formula ψ.

Case 1.2.1: *t' is semi-invisible with respect to ψ:*

Requirement 2 (S-INV) ensures, if t' is semi-invisible regarding ψ, then all states along the new path $t't_0 \ldots$ are satisfying ψ.

Case 1.2.2: *$m \not\models \psi$:*

Otherwise, requirement 2 (S-INV) states that all transitions $t_0, \ldots$ are semi-invisible regarding ψ and together with the facts that $t' \in \text{stub}(m, q)$ and that requirement 3 ensures that t' satisfies ψ, all states along the new path $t't_0 \ldots$ are satisfying ψ.

In both cases 1.2.1 and 1.2.2 the path π_1 is extended by one state and has an infinite accepting continuation in $\mathcal{P}$.

$$\underbrace{q_s \longrightarrow (m_s, q_s) \longrightarrow (m, q) \xrightarrow{t'} (m', q)}_{\pi_1} \underbrace{\xrightarrow{t_0} (m_0, q) \xrightarrow{\cdots}}_{\pi_2}$$

This construction can now be repeated as often as necessary to get an infinite accepting path in $\mathcal{P}$, contradicting the assumption that π_1 is the longest executable prefix.

Case 2: *$\exists n \in \mathbb{N}\ \exists (m_n, q_n) : q_n \neq q$:*

There exists a state (m_n, q_n) which is leaving the current Büchi state, that is $q_n \neq q$. Let q_n the first such state, then it holds that $\forall i < n : q_i = q$ and $m_n \models \varphi_j$, for a $j \in [1, h]$. Requirement 4 (NLG) ensures that there exists a t_i where $i \in [0, n]$ with $t_i \in \text{stub}(m, q)$. Using requirement 1 (COM) we switch t_i to the front of the path.

$$m \xrightarrow{t_i t_0 \ldots t_{i-1}} m_i \xrightarrow{t_{i+1} \ldots t_n} m_n$$

It follows the same argumentation as in cases 1.1.1 and 1.1.2. The following two cases show that requirement 4 (NLG) implies that all transitions $t_0, \ldots, t_{i-1}$ are satisfying the retarding formula ψ.

Case 2.1: *t_i is semi-invisible with respect to ψ:*

Requirement 2 (S-INV) ensures, if t_i is semi-invisible regarding ψ, then all states along the new path $t_i t_0 \ldots t_{i-1}$ are satisfying ψ.

Case 2.2: *$m \not\models \psi$:*

Otherwise, requirement 2 (S-INV) states that all transitions $t_0, \ldots, t_{i-1}$ are semi-invisible regarding ψ and together with the fact that requirement 3 ensures that t_i satisfies ψ, all states along the new path $t_i t_0 \ldots t_{i-1}$ are satisfying ψ.

Hence, the continuation $m \xrightarrow{t_i t_0 \ldots t_{i-1}} m_i \xrightarrow{t_{i+1} \ldots t_n}$ describes an infinite accepting path in $\mathcal{P}$ with the same sequence of traversed Büchi states, where now at

least one more state is executable in $\underline{\mathcal{P}}$. This construction can be repeated at most n-times to find an infinite accepting path in which all transitions $t_0 \dots t_n$ are executable in $\underline{\mathcal{P}}$, but possibly in a different order. This infinite accepting path leaves q and changes to q_n. For the new Büchi state q_n we can apply the construction according to case 1 or case 2 again such that an executable infinite accepting path in $\underline{\mathcal{P}}$ is formed. This contradicts the initial assumption on the choice of π. □

10.3 Comparison

This section compares the new automata-based approach with the conventional LTL$_{-\mathbf{X}}$ POR. We demonstrate for each weakened requirement potential gains in the reduction. For simplicity, we assume that the different formulas φ_i are already negated.
The first restriction, we can drop, is the restriction to stuttering invariant formulas. A model checker using the conventional LTL$_{-\mathbf{X}}$ POR cannot apply it for formulas containing the **X**-operator. In the benchmark from the 2019 edition of the MCC [68] the **X**-operator occurred in more than 25 % of the formulas of the LTL category. These formulas are now accessible by POR using our new automata-based stubborn set approach.

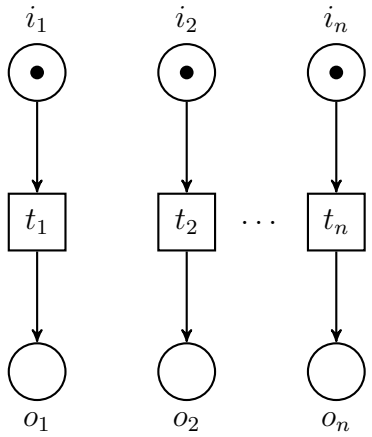


Figure 10.2: A system modeled as P/T net with n concurrent processes.

For the next comparisons, we use the P/T net in Figure 10.2. It models a system of $n \in \mathbb{N}$ concurrent processes. Each process has one transition $t_j, j \in [1, n]$, one input pre place i_j, and one output post-place o_j. The state space consists of 2^n markings with $2n2^n$ edges. Note, if one utilizes on-the-fly strategies for the verification the following first two examples have only n

states, regardless of whether and which POR method is used. Nevertheless, we choose these examples due to their simplicity and the fact that they emphasize the possible reduction power of the new method.
In this section, let B^* be the corresponding product automaton.

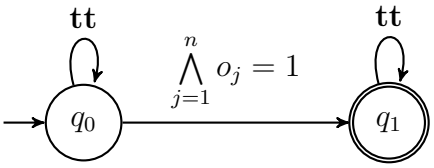


Figure 10.3: Büchi automaton for LTL formula:
$\varphi_1 = \mathbf{F}\left(\bigwedge_{j=1}^{n} o_j = 1\right)$.

As first example, we consider the LTL formula $\varphi_1 = \mathbf{F}\left(\bigwedge_{j=1}^{n} o_j = 1\right)$. The corresponding Büchi automaton is shown in Figure 10.3. The original visibility property based on Definition 9.2.3 states that each transition is visible to φ_1, i.e., each transition t_j produces a token on o_j. This means, there is no reduction possible. The reduced state space contains 2^n markings and is the same as the original state space. Therefore, B^* has $2^n + 2$ reachable states: the initial state, 2^n states in q_0, and one state in q_1.
However, in the automata-based approach each transition is semi-invisible in both Büchi states q_0 and q_1. The reason for this is that based on Definition 10.1.3 (S-INV), we only care for the retarding formula which in this case is always true. We see here the sizable effect of the weakened visibility principle. In q_0 any singleton set $\{t_j\}$ is a valid stubborn set, as long as o_j does not contain a token. Without firing t_j the q_0 cannot be left. And besides this, all other transitions are independent of each other. The reduced product automaton has only $n + 3$ reachable states: the initial state, $n + 1$ chain-shaped states in q_0, and one state in q_1.
The exponential reduction from $2^n + 2$ to $n + 3$ is based on the restriction of the visibility.
LTL formula $\varphi_2 = \left(\bigwedge_{j=1}^{n} o_j = 0\right) \mathbf{U} \left(\mathbf{G}\left(\bigvee_{j=1}^{n} o_j = 1\right)\right)$ is our second example and has the corresponding Büchi automaton shown in Figure 10.4. Again, according to Definition 9.2.3 each transition in φ_2 is visible and, as such, no reduction can be applied. This leaves B^* with $2^n + 1$ reachable states: the initial state, one state in q_0, and $2^n - 1$ states in q_1.
Using the new approach, no reduction is possible in q_0 based on the S-INV, because each transition can leave q_0. But in q_1 reduction is possible, since all transitions are semi-invisible. With this, any singleton stubborn set $\{t_j\}$ is

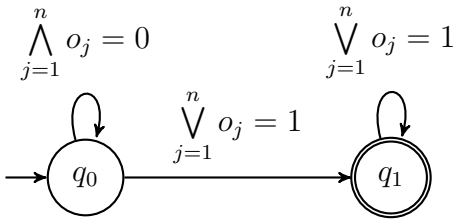


Figure 10.4: Büchi automaton for LTL formula:
$\varphi_2 = \left(\bigwedge_{j=1}^{n} o_j = 0\right) \mathbf{U} \left(\mathbf{G} \left(\bigvee_{j=1}^{n} o_j = 1\right)\right)$.

valid, as long as o_j is empty. The reduced B^* has only $\frac{n(n+1)}{2} + 2$ reachable states: the initial state, one state in q_0, and $\frac{n(n+1)}{2}$ states in q_1.

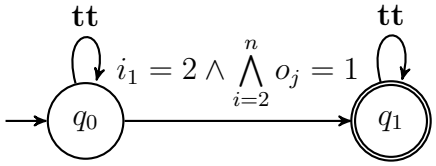
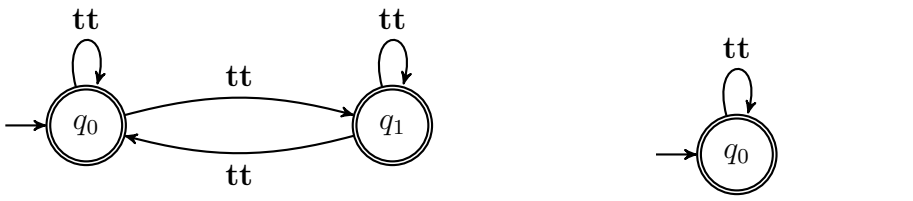


Figure 10.5: Büchi automaton for LTL formula:
$\varphi_3 = \mathbf{F} \left(i_1 = 2 \wedge \bigwedge_{j=2}^{n} o_j = 1\right)$.

Our third example is the LTL formula $\varphi_3 = \mathbf{F} \left(i_1 = 2 \wedge \bigwedge_{j=2}^{n} o_j = 1\right)$ and its corresponding Büchi automaton is illustrated in Figure 10.5. The conventional definition of the visibility property states that t_1 is invisible to φ_3 with respect to T and all other transitions $t_2, \ldots, t_n$ are visible to φ_3 with respect to the empty set. This means, in the initial state T can be reduced to the valid stubborn set $\{t_1\}$. As consequence, the state space is slightly reduced to $2^{n-1} + 1$ markings. Accordingly, B^* also reduces to $2^{n-1} + 2$ reachable states: the initial state, and $2^{n-1} + 1$ states in q_0. Note, in q_1 no state is reachable because $i_1 = 2$ can never be satisfied.
But with the automata-based approach, we can get a substantially better reduction. Requirement 5 (KEY) states that an enabled transition has to be part of the stubborn set only in an accepting Büchi state. Since q_0 is not an accepting state, the empty set is a valid stubborn set. Although there are enabled transitions in the initial state, none of them is able to leave q_0. As mentioned before, the reason is that $i_1 = 2$ can never be satisfied. Independent from n, the reduced product automaton always has just two states: the initial state, and one state in q_0.
One thing that should be mentioned is that the reduction power strongly dependents on the formula and the used Büchi automaton. With some effort,



(a) Unfavorable Büchi automaton. (b) Favorable Büchi automaton.

Figure 10.6: Büchi automata for LTL formula: $\varphi_4 = \textbf{false}$.

we can construct unfavorable Büchi automata where the conventional method performs equally well or even better than the new method. Figure 10.6a shows such an unfavorable example for the LTL formula $\varphi_4 = \textbf{false}$, a formula that accepts any path. The problem is that it is possible to switch between the Büchi states in any state of B^* and therefore no reduction can be applied based on the retarding formula. Although each transition is invisible and thus independent of any other transition, we cannot exploit that, here.
With conventional POR, B^* has a size of $2n+3$ states, while B^* built with the new approach has $2 \cdot 2^n + 1$ states. The good news is that this happens only because the Büchi automaton has been chosen unwisely. The LTL formula φ_4 can also be represented with just one Büchi state. For this, we simply remove one Büchi state from Figure 10.6a to get the equivalent Büchi automaton shown in Figure 10.6b. With this Büchi automaton, the problem vanishes and the reduction power is equivalent for both POR methods and leads to $n + 2$ states.

Property/principle	**Old**	**New**
stubborn set generator	$M \to 2^T$	$M \times Q \to 2^T$
X-operator	not supported	improved: supported
VIS	✓	improved: replaced by weaker S-INV
IGN	✓	improved: not required
KEY	✓	improved: required only in accepting states
NLG	not required	✓

Table 10.1: Comparison of the conventional LTL POR and the newly introduced automata-based POR.

We conclude with Table 10.1, which summarizes the advantages and disadvantages of the conventional LTL$_{-\mathbf{X}}$ POR (old) compared to the new automata-based POR (new) method.

10.4 Discussion

Stubborn sets promise remarkable state space reduction for concurrent systems. Concurrency is one of the main sources for the state explosion problem, but it is also the most important factor for commutativity. Therefore, stubborn sets tackle directly a major source of the state space explosion.
Most POR approaches for LTL focus only on the transition system and ignore additional information from the Büchi automaton. To the best of our knowledge, there exist just two previous approaches that use this information. Peled et al. [98] use the additional information to relax the original invisibility property to those propositions that lie ahead of the current Büchi state and Lehmann et al. [75] use it to restrict the invisibility property to only one proposition at a time, and only for retarding formulas. Furthermore, Lehmann et al. introduced the idea to first build B^* with the original state space and then reduce B^*, using the additional information from the Büchi automaton. The drawback of their method is that it only works for elementary Büchi automata.
Our contribution is a generalization and extension of the ideas presented by Lehmann et al. [75] and the ideas described in [64]. We introduced a new semi-invisibility principle, that only has to hold in one direction. In addition, we introduced a new non-leaving principle for the reduction. As a consequence, we can weaken or drop some requirements of the conventional LTL$_{-\mathbf{X}}$ POR method. This results in several advantages and *increases the reduction efficiency.*
One drawback of the conventional approach is that it only works on stutter-invariant formulas, that is, formulas that do not contain the **X**-operator. Kan et al. [62] introduced an approach that is able to do POR with the **X**-operator. In the process of translating the LTL formula to a Büchi automaton, they use a heuristic to detect and extend stutter-invariant components of the Büchi automaton. These components are then used to guide the reduction of the state space. In contrast, our reduction is only applied within a single Büchi state and thus, stuttering becomes irrelevant. In fact, finite stuttering within a Büchi state does not change the accepting behavior, and infinite stuttering can always be avoided, if this was possible in the original product automaton. This happens due to the NLG principle of the reduction. Therefore, we can drop the restriction to LTL$_{-\mathbf{X}}$ and verify any LTL formula. Lehmann et

al. [75] also uses additional information from the Büchi automaton. But their reduction principle was different and they could not drop the **X**-operator.
Since stuttering is irrelevant, the non-ignorance principle can be dropped as well. Furthermore, the key-transition principle can be reduced to states that are part of the acceptance set.
To summarize, we have weakened or dropped several requirements compared to other LTL POR methods. Compared to the conventional method the reduction power can be exponentially better. This is shown by our examples, e.g., $n+3$ vs. 2^n+2 states, or 2 vs. $2^{n-1}+2$ states. However, the reduction power depends on the formula, or to be exact, on the corresponding Büchi automaton.
In future work, we want to extend our new method to other formalisms than P/T nets. Furthermore, we want to explore whether the new automata-based approach can be tuned to be always better or at least equally good as the conventional method. In addition to this, we want to implement our new method in our proof-of-concept model checker LoLA [142].

Chapter 11

Stubborn sets for special CTL formulas

This chapter introduces stubborn sets for simple and frequently occurring CTL formulas. We have published the following methods with its results in [79]. The following chapter is based on this publication. Some of the choices made in this chapter are justified by empirical evaluations based on the MCC benchmark.

In recent years, CTL has been the category where most queries were left unsolved in the yearly model checking contest [66–68]. Consequently, improving CTL model checking technology deserves particular attention with the aim of keeping pace with LTL and reachability checking. At present, leading CTL model checkers such as TAPAAL [34] and LoLA [142] use explicit model checking algorithms. One of their main technique for alleviating state explosion is the stubborn set method [127]. As shown in Proposition 9.3.4, CTL preserving POR has severe restrictions: we either find, in a given marking, a *singleton* set consisting of an invisible transition that satisfies all other conditions for a stubborn set, or we have to fire all transitions enabled in this marking. This condition is necessary for CTL preservation since otherwise the position of visible transitions with respect to branching points may not be preserved which in turn would jeopardize preservation of the branching time logic CTL.

Many CTL queries have a rather simple structure in the sense that they contain only few temporal operators. In the MCC, this might be an artifact of the formula generation mechanism. However, we share the same experience with the users of our tool LoLA. Even if complicated CTL formulas occasionally occur, they are subject to several simplification approaches as we have seen in Chapter 8). Firstly, there exist many tautologies in temporal logic. Not all of them are commonly known. This way, an originally complicated

formula may automatically be rewritten to a much simpler query [7]. The formula rewriting system of LoLA currently contains more than 100 rewrite rules that are based on CTL* tautologies. For P/T nets, secondly, linear programming techniques employing the Petri net state equation can be applied to the atomic propositions in the formula [7], sometimes proving them to be invariantly true or false. This way, whole subformulas of a query may collapse, enabling further rewriting based on tautology. Boolean combinations of queries can be simplified by checking the subformulas separately (thus having queries with less visible transitions in each run which propels POR). Thirdly, complicated queries may be replaced by simpler queries through modifications in the system under investigation. A simple example is the verification of relaxed soundness [36] for workflow nets. For every transition t, we have to show that there is a path to a given final place f that includes the occurrence of t. In CTL, this reads as $\mathbf{EF}\,(t \text{ occurs} \wedge \mathbf{EF}\,(f \geq 0))$. Inserting a fresh post-place p to t, the query can be simplified to $\mathbf{EF}\,(p \geq 0 \wedge f \geq 0)$. The most systematic approach of this kind is LTL model checking as a whole. As shown in Section 4.4, the explicit verification of an LTL formula φ is done by modifying the system under investigation, namely to construct the product system with the Büchi automaton for $\neg\varphi$. In the modified system, we only need to verify $\neg\mathbf{GF}$ *accepting-state* instead of the arbitrarily complicated φ. We conclude that explicit CTL model checking can be substantially improved through a special treatment of as many as possible of the most simple queries. Special treatment means that we apply a specific verification procedure to such queries thus avoiding the application of the generic CTL model checking routines. This approach has two obvious advantages. Firstly, some of the queries may permit the use of completely different verification technology. For example, for properties like $\mathbf{EF}\,\varphi$ or $\mathbf{AG}\,\varphi$, with φ assumed not to contain additional temporal operators, we may employ the Petri net state equation for verification [140]. We used this approach in Chapter 6.7 to supplement the verification of certain properties with an additional quick check that can be run in parallel. Secondly, a verification technique dedicated to just one class C of simple CTL queries may use a better partial order reduction: we only need to preserve C rather than whole CTL.

In this chapter, we focus on the second item. We identify several classes of simple CTL queries for which specific search routines enable the use of POR methods better than CTL preserving ones. These POR methods are already known in some cases. So the actual contribution of this chapter is to show that the systematic separation of simple queries from general CTL routines can indeed improve CTL model checking. In all reported cases, we will be able to drop the very limiting BRA principle that enables reduction only in markings where just one (invisible) enabled transition is sufficient to

meet all the other principles. In addition, less restrictive conditions, i.e., a smaller set of principles to be met, leads to potentially smaller stubborn sets and thus to better reduction. We used this approach In the MCC of 2018 where almost 70 % of the CTL queries were transferred to specific routines for simple queries in our tool LoLA. Employing these methods, LoLA could solve more than 50 % of the queries that could not be solved with the generic CTL model checking algorithm.
Additionally, we introduce a specialized search routine for the CTL formula $\mathbf{EG}\,\mathbf{EF}\,\varphi$ in this chapter. It is preferable to use a routine that is specialized for a specific formula type because, instead of general CTL or LTL, they do not need to consider every detail of the entire temporal logic. For example, the check for $\mathbf{EG}\,\mathbf{EF}\,\varphi$ can be folded into a single depth-first search instead of using the entire recursive CTL model checking algorithm.
The simple problems discussed below appear as pairs of an existentially and a universally quantified formula. These two formulas can be reduced to each other by negation. Hence, they permit the application of the same verification techniques and we may restrict subsequent considerations to existentially quantified formulas. In the sequel, let φ and ψ be CTL formulas without temporal operators, and $N = (P, T, F, W, m_0)$ be an arbitrary, fixed P/T net.
The remainder of this chapter is organized as follows. Section 11.1 presents stubborn set dialects for $\mathbf{EF}\,\varphi$ and $\mathbf{AG}\,\varphi$. We continue in Section 11.2 with $\mathbf{EG}\,\varphi$ and $\mathbf{AF}\,\varphi$ preserving stubborn sets. Section 11.3 describes how to solve $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$ and $\mathbf{A}\,(\varphi\,\mathbf{R}\,\psi)$ formulas with stubborn sets. Section 11.4 presents a single depth-first search for $\mathbf{EG}\,\mathbf{EF}\,\varphi$. Section 11.5 is dedicated to stubborn sets for $\mathbf{EF}\,\mathbf{EG}\,\varphi$ and $\mathbf{AG}\,\mathbf{AF}\,\varphi$. Section 11.6 continues with stubborn sets for properties tightly related to TSCCs of the reachability graph. Section 11.7 proposes an approach for handling formulas starting with the **X**-operator. Section 11.8 is dedicated to Boolean combinations. In Section 11.9, we introduce stubborn sets for a larger class of CTL formulas, namely for single-path formulas. We show experimental results in Section 11.10. Finally, Section 11.11 concludes this chapter with a discussion and some directions for future research.

11.1 $\mathbf{EF}\,\varphi$, $\mathbf{AG}\,\varphi$

For the reachability problem $\mathbf{EF}\,\varphi$, we may use stubborn set as suggested by Proposition 9.3.5, or a relaxed version [70]. Both techniques have specific advantages. The first method works much better if $\mathbf{EF}\,\varphi$ is true while the second method has advantages if $\mathbf{EF}\,\varphi$ is false. Any of the methods, however,

is much more powerful than the CTL$_{-\mathbf{X}}$ preserving method.
The ability of LoLA to solve far beyond 90 % of the queries in the reachability category of the MCC, compared to less than 70 % if only a CTL model checker is applied to the CTL category, clearly confirms the conclusion to separate reachability queries from CTL model checking.

11.2 $\mathbf{EG}\,\varphi$, $\mathbf{AF}\,\varphi$

The CTL formula $\mathbf{AF}\,\varphi$ is equivalent to the LTL formula $\mathbf{F}\,\varphi$. The universal path quantifier is implicitly present in LTL, too, since a system satisfies an LTL formula if *all* its paths do. That is, we may apply automata-based LTL stubborn sets proposed in Chapter 10, or conventional LTL$_{-\mathbf{X}}$ preserving stubborn sets instead of CTL$_{-\mathbf{X}}$ preserving ones. Without the BRA principle, LTL$_{-\mathbf{X}}$ preserving stubborn sets are more powerful. More than 90 % of the queries in the LTL category are solved by LoLA, compared to less than 70 % success if CTL$_{-\mathbf{X}}$ preserving stubborn sets are applied to all of the CTL category. Additionally, we may completely drop the IGN principle for visible transitions.

Lemma 11.2.1. *If N is a P/T net and G its reachability graph, then a reduced reachability graph G' obeying the principles COM, KEY, and VIS(φ) preserves* $\mathbf{EG}\,\varphi$.

We sketch a proof for $\mathbf{EG}\,\varphi$.

Proof. If there is no witness path, i.e., an infinite path where φ permanently holds, in G, then there cannot be one in G' which is a subgraph. If there is an (infinite) witness path π, then by COM, KEY, and VIS(φ), there is an infinite path π' in G' such that visible transitions of π' occur in the same order as in π. Invisible transitions in π' do not alter the value of φ. That is, π' witnesses $\mathbf{EG}\,\varphi$ as well since otherwise there would be a prefix of π where φ is violated, contradicting the assumption that π is a witness path. □

When only COM, KEY, and VIS(φ) need to be established in stubborn set computation, we can often find much smaller stubborn sets and achieve much better state space reduction.

11.3 $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$, $\mathbf{A}\,(\varphi\,\mathbf{R}\,\psi)$

To satisfy $\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$, we need to use stubborn sets that preserve two properties: first, the reachability of ψ, and second, the non-violation of φ. Combining the discussion for reachability (**EF**) and non-violation (**EG**), we propose

the following combination of principles for the stubborn sets to be used: COM, UPS(ψ), and VIS(φ).

Lemma 11.3.1. *If N is a P/T net and G its reachability graph, then a reduced reachability graph G' obeying the principles COM, UPS(ψ), and VIS(φ) preserves $\mathbf{E}(\varphi \,\mathbf{U}\, \psi)$.*

We sketch the arguments for correctness of this setting.

Proof. Assume G contains a witness path π. By UPS(ψ), this path contains a transition that is in the stubborn set used in the initial marking. By COM, we can shift the first such transition to the front of the path. By VIS(φ), this modification does not change the order of transitions visible for φ. At least the first transition of the modified path can be replayed in G'. By induction, a witness path in G' is established. □

We obtain a combination of principles where the harmful BRA principle is absent and VIS can disregard ψ. In addition, the UPS principle preserves a shortest witness path. This accelerates the positive effect of on-the-fly model checking in all situations where $\mathbf{E}(\varphi \,\mathbf{U}\, \psi)$ turns out to be true.

11.4 EG EF φ, AF AG φ

For this pair of formulas, we do not have a dedicated version of stubborn sets, so we apply CTL preserving stubborn sets for state space reduction. However, the verification of the pair of temporal operators can be folded into a single depth-first search. We present the approach for **EG EF** φ. A witness path π for the **EG** operator is a maximal path (i.e., infinite or ending in a deadlock).
If the path ends in a deadlock, the deadlock marking has to satisfy φ since this is the only way for φ to be reachable from that marking. If the deadlock satisfies φ, all markings on the path automatically satisfy **EF** φ, so this case can be easily implemented. An infinite path appears in a model checker as a cycle that is reachable from m_0. For satisfying **EG EF** φ, it is necessary and sufficient that, from one of the markings m on the cycle, a marking m' is reachable that satisfies φ. Necessity follows immediately from the definition of the semantics of CTL. Sufficiency follows from the fact that m is reachable from all markings in π, so m' is reachable as well from all markings in π.
We record, for every marking visited in depth-first search, whether a marking satisfying φ can be reached. To this end, every marking that satisfies φ itself is marked as "can reach φ". In addition, whenever depth-first search backtracks from a marking that can reach φ, the predecessor marking is

marked as well as "can reach φ". For detecting cycles, we use the well-known fact from [56] that every cycle in a state space contains an edge from some marking m to a marking m' such that, at some stage of depth-first search, m is the top element of the search stack and m' is on the search stack as well (such an edge is called *backward edge*). During the search, we maintain information whether or not the search stack contains such m'. If this is the case while the marking on top of the stack can reach φ, we return *true*. If we reach a deadlock satisfying φ, we return *true* as well. If the search is completed without having returned true, we return *false*.

Lemma 11.4.1. *The procedure sketched above correctly evaluates* $\mathbf{EG}\,\mathbf{EF}\,\varphi$.

Proof. If we reach a deadlock satisfying φ, $\mathbf{EG}\,\mathbf{EF}\,\varphi$ is trivially true. If we return *true* in any other situation, we have a marking m on the search stack that is member of some cycle reachable from m_0. From m, the top element m' of the stack is reachable and, from m', a marking satisfying φ can be reached. Hence, $\mathbf{EG}\,\mathbf{EF}\,\varphi$ is true. For the other direction, assume that $\mathbf{EG}\,\mathbf{EF}\,\varphi$ is true and consider a witness path π for the $\mathbf{EG}$ operator. If this is a finite path, the final marking must be a deadlock satisfying φ. Otherwise, π is infinite. The set of markings that are visited infinitely often in π is strongly connected, hence contained in an SCC C of the reachability graph. The root m^* of C (i.e., the marking of C entered first by the search) is member of some cycle (by strong connectivity). As m^* is the first marking of C entered by the search, it is target of a backward edge. This is recognized before m^* is finally left by depth-first search. Depth-first search explores all markings reachable from m^* before finally leaving m^*. That is, in the moment we are about to finally leave m^*, we know that m^* is target of a backward edge and can reach φ. Hence, we return true (if we have not returned true much earlier). □

11.5 $\mathbf{EF}\,\mathbf{EG}\,\varphi$, $\mathbf{AG}\,\mathbf{AF}\,\varphi$

Before we introduce dedicated stubborn sets for $\mathbf{EF}\,\mathbf{EG}\,\varphi$, we present our verification approach for $\mathbf{EF}\,\mathbf{EG}\,\varphi$. We check the property by nested depth-first search. The approach uses ideas from [28, 29, 47, 56] that are concerned with the similar problem of finding accepting cycles in Büchi automata. Outer search proceeds through markings that have already proven not to be part of a φ-cycle or a φ-deadlock, i.e., a deadlock state in which φ holds. This includes markings that do not satisfy φ and markings where inner search has already been run. Inner search proceeds only through φ-markings and tries to find a cycle or a deadlock. By definition, $\mathbf{EF}\,\mathbf{EG}\,\varphi$ holds if and only

if a φ-cycle or a φ-deadlock is reachable from m_0. We start with outer search. Whenever we encounter a fresh φ-marking m, we switch to inner search. If inner search terminates without having found a cycle or deadlock, we resume outer search in m.
This procedure is very similar to the general CTL model checking algorithm. However, we may apply dedicated stubborn sets. In outer search, we distinguish markings that satisfy φ from markings that do not satisfy φ. If m does not satisfy φ, we use stubborn sets that satisfy COM and UPS(φ). If m satisfies φ, we have two correct combinations of principles. We can use stubborn sets that satisfy COM and UPS($\neg\varphi$), or stubborn sets that satisfy COM, KEY, and VIS(φ). In inner search, we use stubborn sets satisfying COM, KEY, and VIS(φ).

Lemma 11.5.1. *If N is a P/T net and G its reachability graph, then a reduced reachability graph G' graph obeying the principles stated above preserves* **EF EG** φ.

Proof. Let $m_1^* \dots m_n^*$ be a φ-cycle or a φ-deadlock (then: $n = 1$). Let $m_1 m_2 \dots m_k$ be a path such that m_1 has been visited in outer search G', and $m_k = m_i^*$, for some i $(1 \leq i \leq n)$. Consider first the case where all m_j with $1 \leq j \leq k$ satisfy φ. Then inner search from m_1 will find a φ-cycle or φ-deadlock since the path

$$\pi = m_1 \dots \left(m_k = m_i^* m_{i+1}^* \dots m_n^* m_1^* \dots m_{i-1}^*\right)^*$$

witnesses **EG** φ and **EG** φ is preserved by stubborn sets with COM, KEY, and VIS(φ) (see Section 11.2).
Second, consider the case where m_1 does not satisfy φ. Since $m_k = m_i^*$ satisfies φ, the path from m_1 to m_k contains a transition of the up-set used in m_1, and, by the UPS principle, elements of the stubborn set used in m_1. Applying COM, we obtain an alternative path where the first transition is in the stubborn set used in m_1. Its successor meets the same properties in m_1 but with a smaller value for k.
It remains to consider the case where m_1 satisfies φ and the first case is not applicable. Then, for at least one q with $2 \leq q \leq k$, m_q violates φ. If we apply stubborn sets satisfying COM and UPS($\neg\varphi$), we argue as in the second case. This yields a continuation for the witness path in the reduced reachability graph. If we obey COM, KEY, and VIS(φ) instead, we argue as follows. If a transition of the stubborn set used in m_1 occurs in π, COM yields a continuation of the path in G'. Otherwise, by VIS(φ), the stubborn set in m_1 contains only invisible transitions . Choose a key transition t' in the stubborn set for m_1 which is available via KEY. By KEY, t' is never disabled

in π. By COM, all transitions in π can still be executed after having fired t'. The t'-successor m' of m_1 occurs in G'. The third case is applicable only a finite number of times since m' satisfies φ but there is no φ-cycle reachable in inner search from m_1. □

11.6 EF AG φ, EF AG EF φ, AG EF φ, AG EF AG φ

These properties are tightly related to TSCCs of the reachability graph. For

- **EF AG** φ, there must exist a TSCC where all markings satisfy φ;
- **EF AG EF** φ, a TSCC must exist where at least one marking satisfies φ;
- **AG EF** φ, every TSCC must contain a marking satisfying φ;
- **AG EF AG** φ, all markings in all TSCCs must satisfy φ.

By Proposition 9.3.2, stubborn sets obeying COM, KEY, and IGN preserve access to all TSCCs of the reachability graph. Adding UPS(φ) for **AG EF** φ and **EF AG EF** φ (or UPS($\neg\varphi$) for the other two cases) preserves at least inside the TSCCs the properties under investigation. There are several strategies for implementing UPS in the TSCCs. We can either require it for all markings (then KEY may be dropped) [112], or enforce a relaxed version of UPS in all markings (see [70] for details), or we may launch a depth first search using stubborn sets with COM and UPS whenever we encounter a TSCC in the reduced graph with respect to COM, KEY, and IGN.
The proposed procedure has two advantages. First, we proceed in a single depth-first search compared to the recursive approach of a CTL model checker. Second, we can drop the very problematic BRA principle. Being able to drop the visibility principle as well, the stubborn set method can achieve substantial reduction even in cases where φ is a property that refers to a large number of places, and causes many transitions to be visible.

11.7 Formulas starting with EX and AX

This section is concerned with formulas of the shape:

- **EX EF** φ
- **EX EG** φ
- **EX E** $(\varphi\,\mathbf{R}\,\psi)$
- **EX E** $(\varphi\,\mathbf{U}\,\psi)$

- $\mathbf{EX}\,\mathbf{EG}\,\mathbf{EF}\,\varphi$
- $\mathbf{EX}\,\mathbf{EF}\,\mathbf{EG}\,\varphi$
- $\mathbf{AX}\,\mathbf{AF}\,\varphi$
- $\mathbf{AX}\,\mathbf{AG}\,\varphi$
- $\mathbf{AX}\,\mathbf{A}\,(\varphi\,\mathbf{R}\,\psi)$
- $\mathbf{AX}\,\mathbf{A}\,(\varphi\,\mathbf{U}\,\psi)$
- $\mathbf{AX}\,\mathbf{AG}\,\mathbf{AF}\,\varphi$
- $\mathbf{AX}\,\mathbf{AF}\,\mathbf{AG}\,\varphi$

We explicitly discuss the existentially quantified ones.

Lemma 11.7.1. *Verification of the properties stated above can be traced back to the respective formula without the leading* **EX***-operator, with the addition to explore all enabled transitions of m_0, and not store m_0.*

This means, whenever m_0 is visited during the search, it is treated as a fresh marking and a stubborn set can be used. Other than this, the same stubborn set approaches as discussed earlier are applicable.

11.8 Boolean combinations

If a CTL formula is a Boolean combination of subformulas, we may check the subformulas individually. Doing that, the subformulas often have a smaller set of visible transitions, so some of the stubborn set principles are stronger for a subformula than for the whole formula. Some subformulas may contain the **X**-operator, so the stubborn set method can be applied at least to the subformulas not containing the **X**-operator. Some subformulas may fall into any of the classes considered above, so their verification may be accelerated. In a setting with distributed memory, the subformulas can be verified in parallel. With shared memory, a parallel execution is not necessarily recommendable since the individual verification procedures compete for memory which may lead to memory exhaustion in all procedures while verification could have been successful if the whole memory were available for either of the procedures.
To get the most out of our accelerated procedures in a shared memory setting, subformulas can be rated according to their simplicity. Then, the simplest formulas are checked first. This way, one gets an increased probability that the result of the Boolean combination can already be determined (by a true subformula of a disjunction or a false subformula of a conjunction) before the procedures for the most complicated formulas have been launched.
Our rating works as follows. The simplest category consists of subformulas that do not contain temporal operators. They are true, false, or can be evaluated by just inspecting the initial marking. The second category consists

of formulas that contain only **X**-operators. They can be verified by exploring the state space to a very limited depth. Then follow categories for the simple cases studied above. The simplicity of these categories is mainly influenced by our experience concerning their performance in the MCC. Then follow the categories in the following order $\text{LTL}_{-\mathbf{X}}$, $\text{CTL}_{-\mathbf{X}}$, LTL, and CTL. For the last categories, applicability of stubborn sets is the distinguishing feature.

11.9 Single-path formulas

In this section, we discuss a larger class of CTL formulas. We aim to apply LTL model checking instead of CTL model checking. This way, the BRA principle may be skipped. Switching to an LTL model checker is actually a good idea, given the better success rate of our tool LoLA in the LTL category of the MCC. According to Clarke and Draghicescu [18], removing the path quantifiers of a CTL formula yields the only candidate to be an equivalent LTL formula. But this candidate may or may not turn out to be indeed equivalent. The ACTL formulas where equivalence can be achieved can be characterized [86]. We chose to apply the approach to a collection of CTL formulas that can be more easily be recognized by a rewriting system.
LTL is a linear time temporal logic. That is, a counterexample for an LTL formula is always a single maximal path of the system. In contrast, CTL is a branching time temporal logic. This means that the counterexample is a subtree of the computation tree (see Definition 3.1.4). For instance, a witness for $\mathbf{EG}\,\mathbf{EF}\,\varphi$ consists of a a maximal path where, for each marking a finite path to a state satisfying φ branches off. Even with the observations made in Section 11.4, the structure remains more complicated than a single path. However, in several cases, the branching structure collapses into a single path. Consider $\mathbf{EF}\,\mathbf{EG}\,\varphi$. Here, we only need a finite path to the first state of a φ-cycle or φ-deadlock, extended with the cycle itself. It is precisely a counterexample for the LTL formula $\mathbf{GF}\,\neg\varphi$ that is obtained by negating $\mathbf{EF}\,\mathbf{EG}\,\varphi$ to $\mathbf{AG}\,\mathbf{AF}\,\neg\varphi$ and then dropping the universal path quantifiers. In the sequel, we shall exhibit a class of CTL formulas and define inductively, where this approach is applicable. We call them *single-path* formulas. They may contain only existential path quantifiers or only universal path quantifiers. In the next definition, let a state predicate be a CTL formula without any temporal operator.

Definition 11.9.1 (Existential single-path formula).
If φ and ψ are existential single-path formulas and ω is a state predicate, then the following formulas are existential single-path formulas:

- ω;
- $\mathbf{EG}\ \omega$;
- $\mathbf{EF}\,\varphi$;
- $\mathbf{E}\,(\omega\,\mathbf{U}\,\varphi)$;
- $\mathbf{E}\,(\varphi\,\mathbf{U}\,\omega)$;
- $\varphi \vee \psi$;
- $\varphi \wedge \omega$.

Universal single-path formulas are defined accordingly:

Definition 11.9.2 (Universal single-path formula).
If φ and ψ are universal single-path formulas and ω is a state predicate, then the following formulas are universal single-path formulas:

- ω;
- $\mathbf{AF}\,\omega$;
- $\mathbf{AG}\,\varphi$;
- $\mathbf{A}\,(\omega\,\mathbf{R}\,\varphi)$;
- $\mathbf{A}\,(\varphi\,\mathbf{U}\,\omega)$;
- $\varphi \wedge \psi$;
- $\varphi \vee \omega$.

This class is similar to the class of finite-single-path formulas form Definition 6.5.1 but are not limited to finite paths. The class of single-path formulas covers several cases discussed earlier in this paper. However, the results above are stronger then the results we shall obtain now, so the separate treatment is indeed justified. It is easy to see that the negation of an existential single-path formula is indeed a universal single-path formula and vice versa. That is, we may restrict subsequent considerations to universal single-path formulas.
For a universal single-path formula φ, let LTL(φ) be the formula obtained from φ by removing all path quantifiers. We claim:

Lemma 11.9.1. *Let φ be a universal single-path formula and N a P/T net. Then N satisfies φ if and only if N satisfies LTL(φ).*

Proof. We show that violation of φ implies violation of LTL(φ) and violation of LTL(φ) implies violation of φ. We proceed by induction, according to Definition 11.9.2.

Case ω *(state predicate):* In both CTL and LTL, a state predicate is violated if it does not hold in the initial marking.
Case $\mathbf{AF}\,\omega$***:*** In both CTL and LTL, a counterexample is a maximal path where all markings violate ω. Since ω is a state predicate, it directly refers to the markings on the path.
Case $\mathbf{A}\,(\omega\,\mathbf{R}\,\varphi)$***:*** A counterexample for $\mathbf{A}\,(\omega\,\mathbf{R}\,\varphi)$ is a finite path to a marking where all but the last marking m_f violate ω and m_f violates φ. As ω is a state predicate, the intermediate markings as such violate ω. Hence, the path, extended by a counterexample path for φ at m_f, which exists by induction hypothesis, yields a path that is a counterexample for LTL($\mathbf{A}\,(\omega\,\mathbf{R}\,\varphi)$). For the other direction, consider a counterexample for LTL($\mathbf{A}\,(\omega\,\mathbf{R}\,\varphi)$). It must have a suffix serving as a counterexample for LTL(φ). Hence, using once more the induction hypothesis, the first marking of that path violates φ. The markings that are not part of the considered suffix violate ω, so the full path is a counterexample for $\mathbf{A}\,(\omega\,\mathbf{R}\,\varphi)$.
Case $\mathbf{A}\,(\varphi\,\mathbf{U}\,\omega)$***:*** A counterexample can either be a maximal path where ω is violated in every marking, then the argument of Case $\mathbf{AF}\,\omega$ applies, or a path where ω is violated until both ω and φ are violated, then the argument of case $\mathbf{A}\,(\omega\,\mathbf{R}\,\varphi)$ applies.
Case $\mathbf{AG}\,\varphi$***:*** This case can be traced back to Case $\mathbf{A}\,(\omega\,\mathbf{R}\,\varphi)$ using the tautology $\mathbf{AG}\,\varphi \iff \mathbf{A}\,(\mathtt{FALSE}\,\mathbf{R}\,\varphi)$).
Case $\varphi \wedge \psi$***:*** If φ is violated, there is a counterexample for φ for which the induction hypothesis may be applied. Otherwise, there is a counterexample for ψ for which again the induction hypothesis applies.
Case $\varphi \vee \omega$***:*** In this case, φ and ω are violated. Since ω is a state predicate, only the initial marking of the path is concerned. Hence, the induction hypothesis applied to φ yields the desired result. □

Using Lemma 11.9.1 the considered fragment of CTL can be verified using an LTL model checker. As another option, we may use a CTL model checker but apply LTL preserving stubborn sets. Existential single-path formulas can be verified by checking their negation.

11.10 Experimental validation

We implemented the methods discussed in this chapter in our proof-of-concept model checker LoLA ($\mathbf{EX}\,\mathbf{EF}\,\mathbf{EG}$, $\mathbf{EX}\,\mathbf{EG}\,\mathbf{EF}$, and their universal counterparts are not yet covered by implementation). For evaluating the methods, we use the benchmark provided by the MCC 2018 [66]. We used the formulas provided in the CTL category. While the nets of the MCC are

contributed by the community, the formulas are actually generated automatically, and are to a certain degree random.

Technique	**All (24544)**	
	#	%
Specialized routines	13366	54.5
Formula simplification	3704	15.1
Quick checks	305	1.2
CTL model checker	7169	29.2

Table 11.1: Techniques used to solve CTL formulas.

The benchmark consists of 767 P/T nets. For every net, 32 CTL formulas are provided. This results in 24544 individual verification problems. For 3704 problems (15.1 %), the initial rewriting process yielded a formula that does not contain any temporal operator. Here, sufficiently many atomic propositions have been found to be invariantly true or false due to formula simplifications introduced in Section 8.1 and in [7]. Resulting formulas can be evaluated by just inspecting the initial marking, so no actual run of a model checker is necessary. 13366 problems (54.5 %), after rewriting, fall into some of the categories mentioned in the previous sections of this chapter and an additional 305 problems (1.2 %) are solved by quick checks from Section 6.7. All in all, we need to run the generic CTL model checker only for 29.2 % of the CTL problems in the benchmark.
Although only 1.2 % of the queries were answered by quick checks, it is still worthwhile to run them, because their cost is insignificant. The reason for this is that there are usually enough CPU cores available for model checking, but only a limited amount of memory. This means insufficient memory is usually the problem. Since structural methods use only polynomial space, due to their connection to NP-complete problems, it is useful to run quick checks in parallel to the state space exploration.
For the 13366 problems where application of a special stubborn sets is possible, we compared the proposed routine with a run of the generic CTL model checking procedure. To this end, we used 300 seconds of execution time and unlimited memory for every problem instance. Experiments were executed on our machine Ebro. It has 32 physical cores running at 2.7GHz and 1 TB of RAM. Memory overflow was no issue within the 300 seconds given to each instance.
Table 11.2 lists the results of our experiments. It shows that specialized stubborn sets in total are more successful than the CTL model checking procedure. For the formulas where specialized stubborn sets have been found, we

Formula type	Count	CTL		Special		Difference	
	#	#	%	#	%	#	%
$\mathbf{EF}\,\varphi$, $\mathbf{AG}\,\varphi$	2471	1438	58.2	2300	93.1	862	34.9
$\mathbf{EG}\,\varphi$, $\mathbf{AF}\,\varphi$	1767	1625	92.0	1670	94.5	45	2.5
$\mathbf{E}\,(\varphi\,\mathbf{R}\,\psi)$, $\mathbf{A}\,(\varphi\,\mathbf{U}\,\psi)$	168	157	93.5	160	95.2	3	1.8
$\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$, $\mathbf{A}\,(\varphi\,\mathbf{R}\,\psi)$	318	187	58.8	198	62.3	11	3.5
$\mathbf{EG}\,\mathbf{EF}\,\varphi$, $\mathbf{AF}\,\mathbf{AG}\,\varphi$	385	276	71.7	277	71.9	1	0.3
$\mathbf{EF}\,\mathbf{EG}\,\varphi$, $\mathbf{AG}\,\mathbf{AF}\,\varphi$	515	340	66.0	431	83.7	91	17.7
$\mathbf{EF}\,\mathbf{AG}\,\varphi$, $\mathbf{AG}\,\mathbf{EF}\,\varphi$	884	286	32.4	343	38.8	57	6.4
$\mathbf{EF}\,\mathbf{AG}\,\mathbf{EF}\,\varphi$ and $\mathbf{AG}\,\mathbf{EF}\,\mathbf{AG}\,\varphi$	13	3	23.1	6	46.2	3	23.1
$\mathbf{EX}\,\mathbf{EF}\,\varphi$, $\mathbf{AX}\,\mathbf{AG}\,\varphi$	353	193	54.7	319	90.4	126	35.7
$\mathbf{EX}\,\mathbf{EG}\,\varphi$, $\mathbf{AX}\,\mathbf{AF}\,\varphi$	197	177	89.8	178	90.4	1	0.5
$\mathbf{EX}\,\mathbf{E}\,(\varphi\,\mathbf{R}\,\psi)$ and $\mathbf{AX}\,\mathbf{A}\,(\varphi\,\mathbf{U}\,\psi)$	19	17	89.5	18	94.7	1	5.3
$\mathbf{EX}\,\mathbf{E}\,(\varphi\,\mathbf{U}\,\psi)$ and $\mathbf{AX}\,\mathbf{A}\,(\varphi\,\mathbf{R}\,\psi)$	33	20	60.6	24	72.7	4	12.1
Boolean	5822	4250	73.0	5239	90.0	989	17.0
Single-Path	421	275	65.3	295	70.1	20	4.8
All	13366	9244	69.2	11458	85.7	2214	16.6

Table 11.2: Comparison between CTL model checking procedure (CTL) and specialized routines (Special).

increased the success rate from 69.2 % to 85.7 %. In other words, specialized routines are able to solve more than half of the cases where a generic CTL model checker was not successful. This means that the proposed approach proved to be effective.

The table also shows that success is very unevenly distributed over the various formula types. The big success of reachability ($\mathbf{EF}\,\varphi$) is of course to be expected and can be quoted to the large portfolio that included search with very powerful stubborn sets and the state equation approach (see Chapter 6).

In case of $\mathbf{EX}\,\mathbf{EG}\,\varphi$, the CTL model checker left only 20 problems open. That is, there is not much room for improvement. Problems in the MCC can be separated into the categories "easy enough for everybody", "too hard for everybody", and "battleground". The first category refers to nets with rather small state space. Here, every approach is able to get a result in time. In the second category, we have nets with very large state spaces and dense dependencies between transitions. At least explicit model checkers that depend on the reduction power of the stubborn set method and other

reduction techniques like net reduction [4, 8, 92], have no chance to verify such systems. This means that progress in model checking mainly refers to the battleground category. We should aim at covering the problems in this category as much as possible. Returning to the $\mathbf{EX}\,\mathbf{EG}\,\varphi$ category, the little success may very well be due to the fact that only one of the 20 formulas left open by the CTL model checker actually fell into the battleground category. Consequently, we do *not* conclude that the special routine for $\mathbf{EX}\,\mathbf{EG}\,\varphi$ is ineffective as such. Given the fact that we may apply more powerful stubborn sets, we have reason to believe that the procedure would be more effective on a different benchmark, with more $\mathbf{EX}\,\mathbf{EG}$ formulas in the battleground category.
From the 385 $\mathbf{EG}\,\mathbf{EF}\,\varphi$ queries, the CTL model checker could solve 261 satisfiable and 15 unsatisfiable queries. The specialized routine could solve one more satisfiable query. Even though, the performance of the new specialized routine is not really highlighted by the experiments, we still consider the new method faster. Almost all queries, namely 103, from the remaining 108 unsolved queries are unsatisfiable. In these cases, both the specialized routine and the CTL model checker would have to search the entire state space. In general, we observed that for satisfiable queries the new method will answer faster than the CTL model checker.
In consequence, the large bandwidth of success rates in the different formula types does not jeopardize the general conclusion in favor of using specialized routines.

11.11 Discussion

We proposed to relieve the CTL model checker by providing specialized stubborn sets for a large set of simple CTL queries. Special treatment permits the use of much more powerful stubborn set dialects. In the benchmark, specialized stubborn sets are applicable to more than half of the problems. In the introduction, we argued that a significant percentage of simple queries has to be expected in practice, too.
With our approach we increased the success rate for simple formulas by 16.6 % in the benchmark. Over half of the simple problems left unsolved by the CTL model checker can now be solved. The performance demonstrated here with the MCC benchmark can be repeated in other situations with meaningful formulas. Thus, with the *state space reduction* due to the powerful stubborn sets and the *improved model checking efficiency*, we have successfully managed two of our research goals,
Finding specialized search routines for simple and frequently occurring for-

mulas is an open problem. Since specialized routines do not need to consider every detail to preserve entire temporal logics, they have the potential to increase the performance of model checking significantly.
Offering the new methods, LoLA unfortunately does not yet reach the performance of TAPAAL [34], the 2018, 2019, and 2020 winner of the MCC CTL category. TAPAAL offers some techniques that have not yet been used by LoLA in the MCC. For instance, TAPAAL uses sophisticated net reduction [4, 8, 92] as another form of preprocessing [34]. LoLA has only recently added the ability to use net reduction as well. Thus, the upcoming 2021 edition of the MCC might give a better comparison between both tools. The largest difference is the handling of frequently occurring formulas vs. TAPAAL's usage of dependency graphs [31, 41].
An open question is, how to build stubborn sets for formula types that contain different path quantifiers and which are not related to TSCCs such as $\mathbf{EG}\,\mathbf{AF}\,\varphi$. In general, future work includes finding in a systematically way more formula types that permit any improvement in verification. In addition, some of the ideas of this chapter could be integrated into a CTL model checker itself. For instance, treating $\mathbf{AG}\,\mathbf{EF}$, or even $\mathbf{AG}\,\mathbf{EF}\,\mathbf{AG}$, in a single depth-first search should be possible even if that pair of operators occurs in the middle of a more complex CTL formula. In addition, the proposed stubborn set dialects may not necessarily be the optimal ones for the respective formula type. Finding alternative stubborn set methods for larger classes of formulas, we may ultimately be able to have a dedicated dialect of stubborn sets for every subformula of a CTL query.

Part V

Conclusions

Chapter 12

Conclusion

The contributions of this thesis are all about the improvement of model checking efficiency both in theory and in practice. To this end, we introduced several techniques to improve vital components of the model checking procedure. Figure 12.1 repeats the model checking procedure from Chapter 1 and highlights our contributions in the corresponding components.

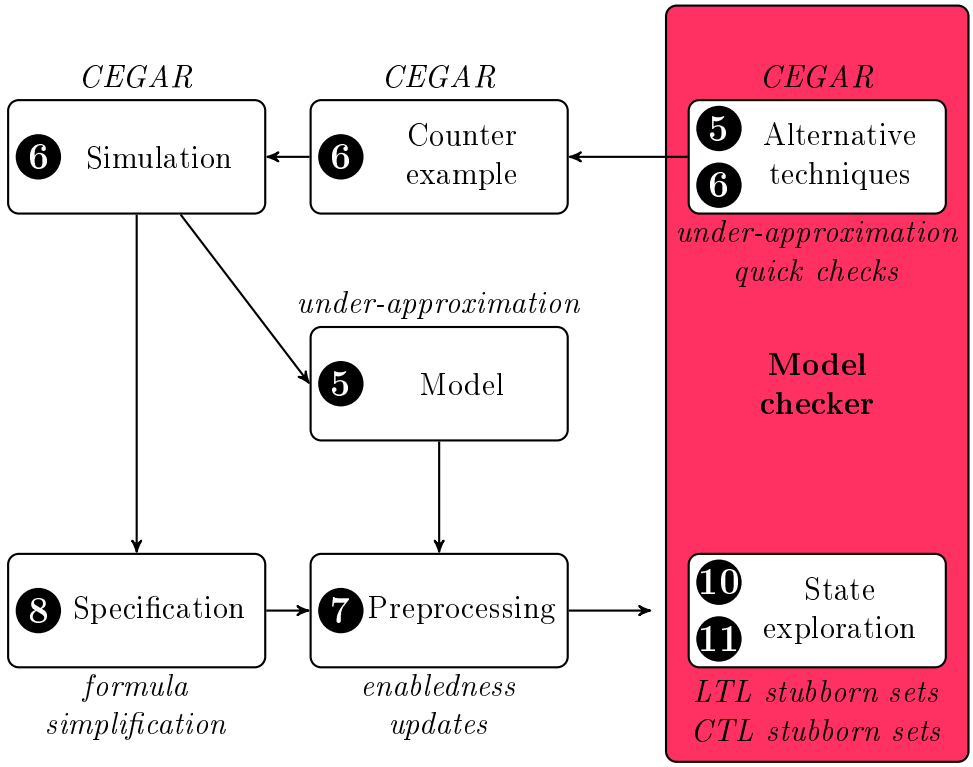


Figure 12.1: Model checking procedure and compatibility of results.

We met the research challenges identified in the beginning of this thesis with the following techniques.

1. *Model simplification.*
   To simplify a P/T net $N = (P, T, F, W, m_0)$, previous approaches reduced P, T, F, W but not m_0. We have shown for the case of token-scaling models that reducing the number of tokens on the initially marked places makes sense and results in a significantly smaller state space.

2. *Specification simplification.*
   We introduced several techniques such as embedded place invariants, traps and a multitude of tautologies with the effect that specifications become essentially shorter without losing any semantics.

3. *State space reduction.*
   For the state space reduction, we introduced new stubborn set dialects. In case of LTL, our new method relies not only on the reachability graph but also uses the available information from the given Büchi automaton to generate the stubborn set. For CTL, we focused on frequently occurring formulas and introduced according dedicated stubborn set methods.

4. *State space exploration.*
   We described a new and faster preprocessing algorithm for our acceleration data structure $DI(N)$. Recall that $DI(N)$ is used to reduce the number of enabledness tests in each marking, which improve the computation of the reachability graph.

5. *Alternative reduction techniques.*
   For actual verification, we have advocated a portfolio approach, running several incomplete but fast techniques in parallel. Each technique works well for one special input case. This thesis introduced a CEGAR approach for a set of finite-single-path formulas. We point out that this was the first time that a CEGAR approach has been used for a class of temporal logic formulas other than reachability.

   In addition, we introduced several quick checks for certain necessary or sufficient conditions. These quick checks are based on easy to detect structural properties and can therefore run in parallel to other verification approaches.

6. *Performance enhancement of LoLA.*
   We implemented most of the introduced techniques into our proof-of-concept model checker LoLA. Using the data from the annual MCC as

a benchmark, we demonstrated how well model checking benefits from our improvements. Moreover, each of these implemented techniques is now a module of LoLA and increases the model checking efficiency of this publicly available tool.

In the remaining sections of the thesis, we address some final selected topics that we believe deserve some remarks. Section 12.1 discusses the combined use of the techniques introduced in this thesis. Finally, Section 12.2 is dedicated to open problems and future work.

12.1 Compatibility

As we have seen, the introduced techniques concern different parts of the model checking procedure. Consequently, whenever possible, it is reasonable to combine them to achieve the best possible performance.

Contribution	**CTL**	**LTL**	**Notes**
formula simplification	✓	✓	
under-approximation	(✓)	✓	ACTL* and ECTL*
enabledness tests	✓	✓	
CEGAR for FLP formulas	(✓)	(✓)	Only certain formula types
quick checks	(✓)	(✓)	Only certain formula types
LTL stubborn sets	(✓)	✓	Only CTL formulas that are LTL formulas
CTL stubborn sets	(✓)	(✓)	Only certain formula types

Table 12.1: Compatibility of contributions regarding CTL and LTL.

Table 12.1 shows the compatibility of the introduced techniques regarding CTL and LTL. A checkmark in brackets means that the technique is only applicable to a subset of CTL or LTL, respectively. The specific subset can be found in the description of the respective formula types in the corresponding chapters.
Due to our experience from the model checking contests and real world use cases, we found the following rules of thumb for combining the introduced techniques. Formula simplification and acceleration of enabledness updates should always be used. The additional runtime in the preprocessing is compensated many times over due to the method advantages. First, a simplified formula may fall into one of the supported classes for which we have dedicated algorithms. Second, a shorter formula has less visible transitions which increases the power of stubborn sets. And third, with $DI(N)$ the speed of state space exploration increases significantly since the number of enabledness tests in each marking is significantly reduced.

The under-approximation for token-scaling models should be applied with care, because it requires a separate state space search. When used in parallel to other verification techniques, as supposed to, under-approximation should be killed first when memory becomes exhausted. The freed memory can then be used for the remaining verification tasks.
The CEGAR approach and the quick checks can be run in parallel at all times. They are based on structural methods and therefore only need limited memory. Stubborn set methods should also be always applied. The overhead computation decreases the runtime only marginally but is more than compensated by the reduced state space which saves both space and time.
As an example for the compatibility of the introduced techniques, consider a token-scaling model and the CTL formula $\varphi = \mathbf{E}\,(((\mathbf{AG}\,\phi) \wedge \psi)\,\mathbf{U}\,\chi)$ where ϕ, ψ, χ are atomic propositions. Assume that, due to the introduced formula simplifications, ϕ is invariantly true. Hence, $\varphi = \mathbf{E}\,(((\mathbf{AG}\,\mathtt{TRUE}) \wedge \psi)\,\mathbf{U}\,\chi)$ becomes $\varphi = \mathbf{E}\,(\psi\,\mathbf{U}\,\chi)$. With this, φ is now one of the specialized formulas that can be solved using the CEGAR approach. In addition, we apply in the state space exploration the introduced dedicated stubborn set dialect for this formula type. During this exploration, we further use the reduced enabledness tests provided by $DI(N)$ and which we have preprocessed in a shorter time with our new method. Furthermore, we can run the under-approximation method in parallel, since the formula is an ECTL* formula and the model scales over tokens. And finally, we also apply the necessary quick check $\mathbf{EF}\,(\chi)$, which, if it fails, saves the resources of an entirely useless verification.
All in all, the introduced techniques support each other or even make each other possible in the first place.

12.2 Open problems and future work

The work in this thesis can be continued and improved in several aspects, which we wish to lay out in the following listing:

- *Verification with under-approximation.*
  In this approach, we either only get a necessary or only a sufficient condition for the verification of token-scaling-models. This makes dependent on a witness or a counterexample. We believe that it is possible to verify properties for token-scaling models with a reduced number of tokens that do not depend on producing a witness or a counterexample. This would result in the ability to verify other formula types with this procedure.

– *CEGAR for finite-single-paths formulas.*
An open question here is, how to solve the remaining four basic CTL operators, namely $\mathbf{EG}\,\varphi$, $\mathbf{E}\,(\varphi\,\mathbf{R}\,\psi)$, and their universal counterparts. However, this requires to deal with infinite paths, which are difficult because they do not have a final marking. One way to overcome this issue would be T-invariants. However there are easily exponentially many T-invariants and even for a single T-invariant x, we would need to find a marking which can execute x. If n is the number of transitions in x, then there are n markings that can be used as starting point. For this difficulties, handling infinite paths remains an open problem.

– *Accelerating the state space computation.*
In the future, we want to, beside accelerating the computation of $DI(N)$, also reduce the memory footprint of $DI(N)$. Storing $DI(N)$ requires a lot of space, in worst case $\mathcal{O}(|T^2|)$. An idea to this end would be to utilize that, in our experience, transitions t and t' often share an overlapping environment E. Storing E only for t and a pointer to E for t' would save memory. Finding such suitable overlapping environments without increasing the runtime significantly is an open issue.

– *Simplifying the specification.*
Often we observe that a specification contains equivalent subformulas. There are many reasons for this, but one main driver seems to be the unfolding of colored Petri nets [59] into low-level Petri nets. As future work, we want to find more of these equivalent subformulas and get rid of them.

– *Specialized algorithms and POR dialects for specific formula types.*
We will continue the work on dedicated algorithms and stubborn set methods for small classes of properties. Furthermore, we want to find more quick checks. A natural starting point would be frequently occurring LTL formulas such as stabilization ($\mathbf{FG}\,\varphi$), immortality ($\mathbf{GF}\,\varphi$), and leads-to-formulas ($\mathbf{G}\,(\varphi \implies \mathbf{F}\,\psi)$).

– *Implementation in LoLA.*
In the future, we are going to implement the CEGAR approach for finite-single-paths formulas and the automata-based partial order reduction into LoLA. While the CEGAR approach for reachability is implemented in an external tool, Sara, which is currently just controlled by LoLA, extending this approach, requires a rigorous reimplementa-

tion of Sara in LoLA, which also means to convert Sara's data structures into LoLA's. Only then can we integrate our extensions. Conversely, extending Sara directly, is also not very meaningful, since we want to utilize stubborn sets from LoLA in the realization of CEGAR.

Model checking is a multifaceted challenge and there are many more possible further directions for future work.
Another open problem is the question whether and how structural methods such as traps, siphons, conflict clusters, linear algebra, and others can support the state space exploration. Are structural methods able to cut off certain parts of the state space or can they be used to guide the search? The CEGAR approach is a good example for this challenge and it conveys the hope that even more can be gained from this area.
Moreover, since modern computers represent multi-core architectures, it is only natural to look for more techniques to run in parallel supporting the portfolio approach. This also raises the possibility to introduce more incomplete methods, which we want to explore more closely, in the future.
Further, we have completely neglected multi-core algorithms. In [3] Barnat et al. presented multi-core model checking algorithms for LTL and in [72] Laarman and Wijs showed that the performance of partial order reduction for LTL can be increased with a multi-core approach. So we see it as an open question how the presented approaches can benefit from multi-core machines.
As a final remark, notice that in this thesis, we focused solely on P/T nets. Some of the introduced approaches might be applicable to other formalisms. For example, the automata-based POR for LTL does not include any specific P/T net theory and is therefore a good candidate for a translation to other formalisms.

Bibliography

[1] H. R. Andersen. Model checking and boolean graphs. *Theor. Comput. Sci.*, 126(1):3–30, 1994.

[2] T. Babiak, F. Blahoudek, M. Kretínský, and J. Strejcek. Effective translation of LTL to deterministic rabin automata: Beyond the (f, g)-fragment. In D. V. Hung and M. Ogawa, editors, *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, volume 8172 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2013.

[3] J. Barnat, V. Bloemen, A. Duret-Lutz, A. Laarman, L. Petrucci, J. van de Pol, and E. Renault. Parallel model checking algorithms for linear-time temporal logic. In Y. Hamadi and L. Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 457–507. Springer, 2018.

[4] G. Berthelot and Lri-Iie. Checking properties of nets using transformations. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, pages 19–40, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[5] G. Berthelot and R. Terrat. Petri nets theory for the correctness of protocols. In C. A. Sunshine, editor, *Protocol Specification, Testing and Verification, Proceedings of the IFIP WG6.1 Second International Workshop on Protocol Specification, Testing and Verification, Idyllwild, CA, USA, 17-20 May, 1982*, pages 325–342. North-Holland, 1982.

[6] U. Boker. Why these automata types? In G. Barthe, G. Sutcliffe, and M. Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, pages 143–163. EasyChair, 2018.

[7] F. Bønneland, J. Dyhr, P. G. Jensen, M. Johannsen, and J. Srba. Simplification of CTL formulae for efficient model checking of petri

nets. In V. Khomenko and O. H. Roux, editors, *Application and Theory of Petri Nets and Concurrency - 39th International Conference, PETRI NETS 2018, Bratislava, Slovakia, June 24-29, 2018, Proceedings*, volume 10877 of *Lecture Notes in Computer Science*, pages 143–163. Springer, 2018.

[8] F. M. Bønneland, J. Dyhr, P. G. Jensen, M. Johannsen, and J. Srba. Stubborn versus structural reductions for petri nets. *J. Log. Algebraic Methods Program.*, 102:46–63, 2019.

[9] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In A. W. Mazurkiewicz and J. Winkowski, editors, *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[10] A. Brandstädt. *Graphen und Algorithmen.* Leitfäden und Monographien der Informatik. Teubner, 1994.

[11] J. L. Briz and J. M. Colom. Implementation of weighted place/transition nets based on linear enabling functions. In *Proc. PETRI NETS, LNCS 815*, pages 99–118, 1994.

[12] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.

[13] J. R. Büchi. On a decision method in restricted second order arithmetic. *International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

[14] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 428–439. IEEE Computer Society, 1990.

[15] Christel and J. Katoen. *Principles of model checking.* MIT Press, 2008.

[16] S. Christensen, L. M. Kristensen, and T. Mailund. A sweep-line method for state space exploration. In T. Margaria and W. Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001*

Genova, Italy, April 2-6, 2001, Proceedings, volume 2031 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2001.

[17] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods Syst. Des.*, 19(1):7–34, 2001.

[18] E. M. Clarke and I. A. Draghicescu. Expressibility results for linear-time and branching-time logics. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings*, volume 354 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 1988.

[19] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[20] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[21] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[22] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[23] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[24] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 2001.

[25] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[26] E. M. Clarke, W. Klieber, M. Novácek, and P. Zuliani. Model checking and the state explosion problem. In B. Meyer and M. Nordio, editors, *Tools for Practical Software Verification, LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*, volume 7682 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2011.

[27] F. Commoner. *Deadlocks in Petri Nets*. Applied Data Research, Inc., Wakefield, Massachusetts, Report CA-7206-2311, 1972.

[28] C. Courcoubetis, M. Y. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods Syst. Des.*, 1(2/3):275–288, 1992.

[29] J. Couvreur, A. Duret-Lutz, and D. Poitrenaud. On-the-fly emptiness checks for generalized büchi automata. In P. Godefroid, editor, *Model Checking Software, 12th International SPIN Workshop, San Francisco, CA, USA, August 22-24, 2005, Proceedings*, volume 3639 of *Lecture Notes in Computer Science*, pages 169–184. Springer, 2005.

[30] J. Couvreur and Y. Thierry-Mieg. Hierarchical decision diagrams to exploit model structure. In F. Wang, editor, *Formal Techniques for Networked and Distributed Systems - FORTE 2005, 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5, 2005, Proceedings*, volume 3731 of *Lecture Notes in Computer Science*, pages 443–457. Springer, 2005.

[31] A. E. Dalsgaard, S. Enevoldsen, P. Fogh, L. S. Jensen, P. G. Jensen, T. S. Jepsen, I. Kaufmann, K. G. Larsen, S. M. Nielsen, M. C. Olesen, S. Pastva, and J. Srba. A distributed fixed-point algorithm for extended dependency graphs. *Fundam. Inform.*, 161(4):351–381, 2018.

[32] G. B. Dantzig and M. N. Thapa. *Linear Programming 1: Introduction*. Springer-Verlag, Berlin, Heidelberg, 1997.

[33] D. Das, P. P. Chakrabarti, and R. Kumar. Functional verification of task partitioning for multiprocessor embedded systems. *ACM Trans. Design Autom. Electr. Syst.*, 12(4):44, 2007.

[34] 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 C. Flanagan and B. König, editors, *Tools*

and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings, volume 7214 of *Lecture Notes in Computer Science*, pages 492–497. Springer, 2012.

[35] R. De La Briandais. File searching using variable length keys. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), page 295–298, New York, NY, USA, 1959. Association for Computing Machinery.

[36] J. Dehnert and P. Rittgen. Relaxed soundness of business processes. In K. R. Dittrich, A. Geppert, and M. C. Norrie, editors, *Advanced Information Systems Engineering, 13th International Conference, CAiSE 2001, Interlaken, Switzerland, June 4-8, 2001, Proceedings*, volume 2068 of *Lecture Notes in Computer Science*, pages 157–170. Springer, 2001.

[37] S. Demri and P. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Inf. Comput.*, 174(1):84–103, 2002.

[38] J. Desel and J. Esparza. *Free choice Petri nets.* Cambridge tracts in theoretical computer science 40. Cambridge University Press, Cambridge, 1995.

[39] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer, 1980.

[40] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited: On branching versus linear time. In J. R. Wright, L. Landweber, A. J. Demers, and T. Teitelbaum, editors, *Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 1983*, pages 127–140. ACM Press, 1983.

[41] S. Enevoldsen, K. G. Larsen, and J. Srba. Abstract dependency graphs and their application to model checking. In T. Vojnar and L. Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part*

of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I, volume 11427 of *Lecture Notes in Computer Science*, pages 316–333. Springer, 2019.

[42] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2000.

[43] J. Esparza and S. Melzer. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods Syst. Des.*, 16(2):159–189, 2000.

[44] D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data Knowl. Eng.*, 70(5):448–466, 2011.

[45] A. Finkel. The minimal coverability graph for petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 1993, Papers from the 12th International Conference on Applications and Theory of Petri Nets, Gjern, Denmark, June 1991*, volume 674 of *Lecture Notes in Computer Science*, pages 210–243. Springer, 1991.

[46] A. Gaiser and S. Schwoon. Comparison of algorithms for checking emptiness on büchi automata. In P. Hlinený, V. Matyás, and T. Vojnar, editors, *Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, MEMICS 2009, November 13-15, 2009, Prestige Hotel, Znojmo, Czech Republic*, volume 13 of *OASICS*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[47] J. Geldenhuys and A. Valmari. More efficient on-the-fly LTL verification with tarjan's algorithm. *Theor. Comput. Sci.*, 345(1):60–82, 2005.

[48] R. Gerth, R. Kuiper, D. A. Peled, and W. Penczek. A partial order approach to branching time logic model checking. In *Third Israel Symposium on Theory of Computing and Systems, ISTCS 1995, Tel Aviv, Israel, January 4-6, 1995, Proceedings*, pages 130–139, 1995.

[49] P. Godefroid and P. Wolper. A partial approach to model checking. In *Proceedings of the Sixth Annual Symposium on Logic in Computer*

Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991, pages 406–415, 1991.

[50] R. Gorrieri. *Process Algebras for Petri Nets : The Alphabetization of Distributed Systems.* Monographs in Theoretical Computer Science. An EATCS Series. Springer International Publishing, Cham, 2017. 1 Online-Ressource (307 pages).

[51] O. Grumberg and D. E. Long. Model checking and modular verification. In *Proc. CONCUR, LNCS 527*, pages 250–265, 1991.

[52] M. H. T. Hack. Analysis of Production Schemata by Petri Nets. Master's thesis, MIT, Dept. Electrical Engineering,, Cambridge, Mass, 1972.

[53] Á. Hajdu, A. Vörös, and T. Bartha. New search strategies for the Petri net CEGAR approach. In *Application and Theory of Petri Nets and Concurrency - 36th International Conference, PETRI NETS 2015, Brussels, Belgium, June 21-26, 2015, Proceedings*, pages 309–328, 2015.

[54] Á. Hajdu, A. Vörös, T. Bartha, and Z. Mártonka. Extensions to the CEGAR approach on Petri nets. *Acta Cybern.*, 21(3):401–417, 2014.

[55] M. Heiner. GPPP. `https://mcc.lip6.fr/pdf/GPPP-form.pdf`, 2016. Accessed: 2020-08-15.

[56] G. J. Holzmann, D. A. Peled, and M. Yannakakis. On nested depth first search. In J. Grégoire, G. J. Holzmann, and D. A. Peled, editors, *The Spin Verification System, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, August, 1996*, volume 32 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 23–31. DIMACS/AMS, 1996.

[57] J. E. Hopcroft and J. D. Ullman. *Formal languages and their relation to automata.* Addison-Wesley series in computer science and information processing. Addison-Wesley, 1969.

[58] J. F. Jensen, T. Nielsen, L. K. Oestergaard, and J. Srba. TAPAAL and reachability analysis of P/T nets. *Trans. Petri Nets Other Model. Concurr.*, 11:307–318, 2016.

[59] K. Jensen and L. M. Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems.* Springer, 2009.

[60] P. G. Jensen, K. G. Larsen, J. Srba, M. G. Sørensen, and J. H. Taankvist. Memory efficient data structures for explicit verification of timed systems. In J. M. Badger and K. Y. Rozier, editors, *NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings*, volume 8430 of *Lecture Notes in Computer Science*, pages 307–312. Springer, 2014.

[61] A. Kaiser, D. Kroening, and T. Wahl. Dynamic cutoff detection in parameterized concurrent programs. In T. Touili, B. Cook, and P. B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 645–659. Springer, 2010.

[62] S. Kan, Z. Huang, Z. Chen, W. Li, and Y. Huang. Partial order reduction for checking LTL formulae with the next-time operator. *J. Log. Comput.*, 27(4):1095–1131, 2017.

[63] R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

[64] C. Koch. Weiterentwicklung von Methoden der Partial Order Reduction. Master's thesis, Universität Rostock, 2013.

[65] F. Kordon. RobotManipulation. `https://mcc.lip6.fr/pdf/RobotManipulation-form.pdf`, 2017. Accessed: 2020-08-15.

[66] F. Kordon and T. L. et al. Complete Results for the 2018 Edition of the Model Checking Contest. http://mcc.lip6.fr/2018/results.php, 06 2018. Accessed: 2020-08-15.

[67] F. Kordon and T. L. et al. Mcc'2017 - the seventh model checking contest. *Trans. Petri Nets Other Model. Concurr.*, 13:181–209, 2018.

[68] F. Kordon and T. L. et al. Presentation of the 9th edition of the model checking contest. In D. Beyer, M. Huisman, F. Kordon, and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III*, volume 11429 of *Lecture Notes in Computer Science*, pages 50–68. Springer, 2019.

[69] J. Kretinský and J. Esparza. Deterministic automata for the (f, g)-fragment of LTL. In P. Madhusudan and S. A. Seshia, editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2012.

[70] L. M. Kristensen, K. Schmidt, and A. Valmari. Question-guided stubborn set methods for state properties. *Formal Methods Syst. Des.*, 29(3):215–251, 2006.

[71] F. Kröger and S. Merz. *Temporal Logic and State Systems*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2008.

[72] A. Laarman and A. Wijs. Partial-order reduction for multi-core LTL model checking. In E. Yahav, editor, *Hardware and Software: Verification and Testing - 10th International Haifa Verification Conference, HVC 2014, Haifa, Israel, November 18-20, 2014. Proceedings*, volume 8855 of *Lecture Notes in Computer Science*, pages 267–283. Springer, 2014.

[73] D. Lau. *Algebra und Diskrete Mathematik 1 : Grundbegriffe der Mathematik, Algebraische Strukturen 1, Lineare Algebra und Analytische Geometrie, Numerische Algebra*. Springer-Lehrbuch. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, zweite, korrigierte und erweiterte auflage edition, 2007. Online-Ressource, v.: digital.

[74] K. Lautenbach. Linear algebraic techniques for place/transition nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, Germany, 8-19 September 1986*, volume 254 of *Lecture Notes in Computer Science*, pages 142–167. Springer, 1986.

[75] A. Lehmann, N. Lohmann, and K. Wolf. Stubborn sets for simple linear time properties. In *Application and Theory of Petri Nets - 33rd International Conference, PETRI NETS 2012, Hamburg, Germany, June 25-29, 2012. Proceedings*, pages 228–247, 2012.

[76] T. Liebke. Büchi-automata guided partial order reduction for LTL. In M. Köhler-Bußmeier, E. Kindler, and H. Rölke, editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering co-located with 41st International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2020), Paris, France,*

June 24, 2020 (due to COVID-19: virtual conference), volume 2651 of *CEUR Workshop Proceedings*, pages 147–166. CEUR-WS.org, 2020.

[77] T. Liebke and C. Rosenke. Faster enabledness-updates for the reachability graph computation. In M. Köhler-Bußmeier, E. Kindler, and H. Rölke, editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering co-located with 41st International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2020), Paris, France, June 24, 2020 (due to COVID-19: virtual conference)*, volume 2651 of *CEUR Workshop Proceedings*, pages 108–117. CEUR-WS.org, 2020.

[78] T. Liebke and K. Wolf. Solving E (φUψ) using the CEGAR approach. In D. Moldt, E. Kindler, and M. Wimmer, editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2019), co-located with the 40th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2019 and the 19th International Conference on Application of Concurrency to System Design ACSD 2019 and the 1st IEEE International Conference on Process Mining Process Mining 2019, Aachen, Germany, June 23-28, 2019*, volume 2424 of *CEUR Workshop Proceedings*, pages 47–56. CEUR-WS.org, 2019.

[79] T. Liebke and K. Wolf. Taking some burden off an explicit CTL model checker. In S. Donatelli and S. Haar, editors, *Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019, Aachen, Germany, June 23-28, 2019, Proceedings*, volume 11522 of *Lecture Notes in Computer Science*, pages 321–341. Springer, 2019.

[80] T. Liebke and K. Wolf. Verification of token-scaling models using an under-approximation. In M. Köhler-Bußmeier, E. Kindler, and H. Rölke, editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering co-located with 41st International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2020), Paris, France, June 24, 2020 (due to COVID-19: virtual conference)*, volume 2651 of *CEUR Workshop Proceedings*, pages 1–9. CEUR-WS.org, 2020.

[81] T. Liebke and K. Wolf. Using approximation for the verification of token-scaling models. *Trans. Petri Nets Other Model. Concurr.*, 16, submitted to, invited as an extension for one of the best papers of PNSE'2020.

[82] T. Liebke and K. Wolf. Solving finite-linear-path CTL-formulas using the CEGAR approach. *Trans. Petri Nets Other Model. Concurr.*, 15, to appear.

[83] R. J. Lipton. The reachability problem requires exponential space. *Research Report*, 62, 1976.

[84] X. Liu and S. A. Smolka. Simple linear-time algorithms for minimal fixed points (extended abstract). In K. G. Larsen, S. Skyum, and G. Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 53–66. Springer, 1998.

[85] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods Syst. Des.*, 6(1):11–44, 1995.

[86] M. Maidl. The common fragment of CTL and LTL. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 643–652. IEEE Computer Society, 2000.

[87] M. Mäkelä. Optimising enabling tests and unfoldings of algebraic system nets. In *Proc. PETRI NETS, LNCS 2075*, pages 283–302, 2001.

[88] M. Mihail and C. H. Papadimitriou. On the random walk method for protocol testing. In D. L. Dill, editor, *Computer Aided Verification, 6th International Conference, CAV '94, Stanford, California, USA, June 21-23, 1994, Proceedings*, volume 818 of *Lecture Notes in Computer Science*, pages 132–141. Springer, 1994.

[89] R. Milner. *Communication and concurrency.* PHI Series in computer science. Prentice Hall, 1989.

[90] A. W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In A. Skowron, editor, *Computation Theory - Fifth Symposium, Zaborów, Poland, December 3-8, 1984, Proceedings*, volume 208 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 1984.

[91] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[92] T. Murata and J. Koh. Reduction and expansion of live and safe marked graphs. *IEEE Transactions on Circuits and Systems*, 27(1):68–71, 1980.

[93] L. Napione, D. Manini, F. Cordero, A. Horváth, A. Picco, M. D. Pierro, S. Pavan, M. Sereno, A. Veglio, F. Bussolino, and G. Balbo. On the use of stochastic petri nets in the analysis of signal transduction pathways for angiogenesis process. In P. Degano and R. Gorrieri, editors, *Computational Methods in Systems Biology, 7th International Conference, CMSB 2009, Bologna, Italy, August 31-September 1, 2009. Proceedings*, volume 5688 of *Lecture Notes in Computer Science*, pages 281–295. Springer, 2009.

[94] O. Oanea, H. Wimmel, and K. Wolf. New algorithms for deciding the siphon-trap property. In J. Lilius and W. Penczek, editors, *Applications and Theory of Petri Nets, 31st International Conference, PETRI NETS 2010, Braga, Portugal, June 21-25, 2010. Proceedings*, volume 6128 of *Lecture Notes in Computer Science*, pages 267–286. Springer, 2010.

[95] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[96] E. Pastor, O. Roig, J. Cortadella, and R. M. Badia. Petri net analysis using boolean manipulation. In R. Valette, editor, *Application and Theory of Petri Nets 1994, 15th International Conference, Zaragoza, Spain, June 20-24, 1994, Proceedings*, volume 815 of *Lecture Notes in Computer Science*, pages 416–435. Springer, 1994.

[97] D. A. Peled. All from one, one for all: on model checking using representatives. In *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, pages 409–423, 1993.

[98] D. A. Peled, A. Valmari, and I. Kokkarinen. Relaxed visibility enhances partial order reduction. *Formal Methods in System Design*, 19(3):275–289, 2001.

[99] D. A. Peled and T. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Inf. Process. Lett.*, 63(5):243–246, 1997.

[100] J. L. Peterson. *Petri net theory and the modeling of systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981. X, 290 S., graph. Darst.

[101] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962.

[102] N. Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Log. Methods Comput. Sci.*, 3(3), 2007.

[103] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[104] J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[105] W. Reisig. Petri nets and algebraic specifications. *Theor. Comput. Sci.*, 80(1):1–34, 1991.

[106] W. Reisig. *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.

[107] S. Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 319–327. IEEE Computer Society, 1988.

[108] K. Schmidt. [lola] wird pfadfinder. In J. Desel and A. Oberweis, editors, *6. Workshop Algorithmen und Werkzeuge für Petrinetze (AWPN'99), Frankfurt, Germany, October 11. - 12., 1999*, volume 26 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1999.

[109] K. Schmidt. Stubborn sets for standard properties. In S. Donatelli and H. C. M. Kleijn, editors, *Application and Theory of Petri Nets 1999, 20th International Conference, ICATPN '99, Williamsburg, Virginia, USA, June 21-25, 1999, Proceedings*, volume 1639 of *Lecture Notes in Computer Science*, pages 46–65. Springer, 1999.

[110] K. Schmidt. How to calculate symmetries of petri nets. *Acta Inf.*, 36(7):545–590, 2000.

[111] K. Schmidt. Lola: A low level analyser. In *Proc. PETRI NETS, LNCS 1825*, pages 465–474, 2000.

[112] K. Schmidt. Stubborn sets for model checking the EF/AG fragment of CTL. *Fundam. Informaticae*, 43(1-4):331–341, 2000.

[113] P. Schnoebelen. The complexity of temporal logic model checking. In P. Balbiani, N. Suzuki, F. Wolter, and M. Zakharyaschev, editors, *Advances in Modal Logic 4, papers from the fourth conference on "Advances in Modal logic," held in Toulouse, France, 30 September - 2 October 2002*, pages 393–436. King's College Publications, 2002.

[114] M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67 – 72, 1981.

[115] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[116] C. Stahl, W. Reisig, and M. Krstic. Hazard detection in a GALS wrapper: A case study. In *Fifth International Conference on Application of Concurrency to System Design (ACSD 2005), 6-9 June 2005, St. Malo, France*, pages 234–243. IEEE Computer Society, 2005.

[117] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[118] C. Talcott and D. Dill. The pathway logic assistant. *Third International Workshop on Computational Methods in Systems Biology*, 2005.

[119] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[120] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[121] D. Taubner. On the implementation of petri nets. In *Advances in Petri Nets, LNCS 340*, volume 340, pages 418–434, 1987.

[122] Y. Thierry-Mieg. Symbolic model-checking using its-tools. In C. Baier and C. Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 231–237. Springer, 2015.

[123] Y. Thierry-Mieg. Structural reductions revisited. In R. Janicki, N. Sidorova, and T. Chatain, editors, *Application and Theory of Petri Nets and Concurrency - 41st International Conference, PETRI NETS 2020, Paris, France, June 24-25, 2020, Proceedings*, volume 12152 of *Lecture Notes in Computer Science*, pages 303–323. Springer, 2020.

[124] A. A. Tovchigrechko. *Efficient symbolic analysis of bounded Petri nets using interval decision diagrams*. PhD thesis, Brandenburg University of Technology, Cottbus - Senftenberg, Germany, 2008.

[125] Tricentis. Software Fails Watch 5th edition. `https://www.tricentis.com/wp-content/uploads/2019/01/Software-Fails-Watch-5th-edition.pdf`, 2020.

[126] A. Valmari. Error detection by reduced reachability graph generation. In *9th International European Workshop on Application and Theory of Petri Nets, Venice, Italy, June 1988, Proceedings*, pages 95–112, 1988.

[127] A. Valmari. Stubborn sets for reduced state space generation. In G. Rozenberg, editor, *Advances in Petri Nets 1990 [10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany, June 1989, Proceedings]*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer, 1989.

[128] A. Valmari. *State space generation: Efficiency and practicality*. PhD thesis, Tampere University Of Technology, 1990.

[129] A. Valmari. A stubborn attack on state explosion. *Formal Methods Syst. Des.*, 1(4):297–322, 1992.

[130] A. Valmari. The state explosion problem. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer, 1996.

[131] A. Valmari. Stubborn set methods for process algebras. In D. A. Peled, V. R. Pratt, and G. J. Holzmann, editors, *Partial Order Methods in Verification, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, July 24-26, 1996*, volume 29 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 213–231. DIMACS/AMS, 1996.

[132] A. Valmari and H. Hansen. Stubborn set intuition explained. *Trans. Petri Nets Other Model. Concurr.*, 12:140–165, 2017.

[133] T. van Dijk and J. van de Pol. Sylvan: Multi-core decision diagrams. In C. Baier and C. Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 677–691. Springer, 2015.

[134] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 332–344, 1986.

[135] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 332–344. IEEE Computer Society, 1986.

[136] K. Varpaaniemi. Efficient detection of deadlocks in petri nets. WorkingPaper 26, Helsinki University of Technology, 1993.

[137] K. Varpaaniemi. *On the Stubborn Set Method in Reduced State Space Generation.* PhD thesis, Helsinki University of Technology, 1998.

[138] J. Vautherin. Parallel systems specitications with coloured petri nets and algebraic specifications. In G. Rozenberg, editor, *Advances in Petri Nets 1987, covers the 7th European Workshop on Applications and Theory of Petri Nets, Oxford, UK, June 1986*, volume 266 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 1986.

[139] B. Vergauwen and J. Lewi. A linear local model checking algorithm for CTL. In E. Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 447–461. Springer, 1993.

[140] H. Wimmel and K. Wolf. Applying CEGAR to the petri net state equation. *Log. Methods Comput. Sci.*, 8(3), 2012.

[141] K. Wolf. Running lola 2.0 in a model checking competition. *Trans. Petri Nets Other Model. Concurr.*, 11:274–285, 2016.

[142] K. Wolf. Petri net model checking with LoLA 2. In V. Khomenko and O. H. Roux, editors, *Application and Theory of Petri Nets and Concurrency - 39th International Conference, PETRI NETS 2018, Bratislava, Slovakia, June 24-29, 2018, Proceedings*, volume 10877 of *Lecture Notes in Computer Science*, pages 351–362. Springer, 2018.

[143] K. Wolf. A simple abstract interpretation for petri net queries. In D. Moldt, E. Kindler, and H. Rölke, editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'18), co-located with the39th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2018 and the 18th International Conference on Application of Concurrency to System Design ACSD 2018, Bratislava, Slovakia, June 24-29, 2018*, volume 2138 of *CEUR Workshop Proceedings*, pages 163–170. CEUR-WS.org, 2018.

[144] K. Wolf. How petri net theory serves petri net model checking: A survey. *Trans. Petri Nets Other Model. Concurr.*, 14:36–63, 2019.

[145] K. Wolf. Portfolio management in explicit model checking. In M. Köhler-Bußmeier, E. Kindler, and H. Rölke, editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering co-located with 41st International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2020), Paris, France, June 24, 2020 (due to COVID-19: virtual conference)*, volume 2651 of *CEUR Workshop Proceedings*, pages 10–28. CEUR-WS.org, 2020.

[146] R. Zurawski and M. Zhou. Petri nets and industrial applications: A tutorial. *IEEE Trans. Ind. Electron.*, 41(6):567–583, 1994.

All links were last followed on 11.12.2020.

Abbreviations

ACTL* universal computational tree logic*. 43, 46

BFS breadth first search. 49

CC conflict cluster. 112

CEGAR counterexample guided abstraction refinement. 17, 70, 73

CTL computational tree logic. 18, 43, 45

CTL* computational tree logic*. 39, 43

DFS depth-first search. 49

ECTL* existential computational tree logic*. 43, 46

ILP integer linear programming. 73

LTL linear time temporal logic. 19, 43, 44

LTS labeled transition system. 26

MCC model checking contest. 13

P/T net place/transition net. 29

POR partial order reduction. 19, 120

SAT Boolean satisfiability problem. 48

SCC strongly connected component. 28

TSCC terminal strongly connected component. 28

List of symbols

2^M the power set of M. 24

C_N the incidence matrix of N. 34

$DI(N)$ decrease-increase graph of N. 102

N^r the reduced net. 66

N a place/transition net (low-level Petri net). 29

R_N the reachability graph of P/T net N. 32

T_i the set of transitions that can change s_i. 83

$T_{i,\psi}$ the set of increasing transitions in T_i w.r.t. ψ. 83

$X \times Y$ the Cartesian product of X and Y. 25

$\Delta_{t,s}$ the delta of t regarding sum s. 81

Γ the set of jump and increment constraints. 79

Σ the simulation relation. 65

δ_i the max. arc weight of t in $T_{i,\psi}$. 84

$\mathbb{B}$ the boolean domain. 24

$\mathbb{Z}$ the set of integers. 24

$\mathbb{N}_1$ the set of natural numbers excluding 0. 24

$\mathbb{N}$ the set of natural numbers including 0. 24

$\nearrow(p)$ the set of increasing edges of p. 104

$x^\bullet$ the post-set of x. 29

${}^\bullet x$ the preset of x. 29

$\searrow(p)$ the set of decreasing edges of p. 104

σ the maximal firing sequence of a solution. 79

$\mathbf{ILP}_\varphi$ the initial ILP-problem for finite-single-path CTL formulas. 88

θ_i the offset of consumable tokens in m_0. 84

$\wp(w)$ the Parikh vector. 34

m_0^r the reduced initial marking. 66

r the remainder of a solution. 80

$s \xrightarrow{a} s'$ a transition relation leading with a from s to s'. 26

$t \nearrow t'$ transition t increases transition t'. 102

$t \searrow t'$ transition t decreases transition t'. 101

$v_s(m)$ the integer number of s in m. 80

$|M|$ the cardinality of set M. 24

Index