Time For Stubborn Game Reductions

Bønneland, Frederik Meyer

DOI (link to publication from Publisher):
10.54337/aau429765826

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

TIME FOR STUBBORN GAME REDUCTIONS

BY
FREDERIK MEYER BØNNELAND

DISSERTATION SUBMITTED 2021

AALBORG UNIVERSITY
DENMARK

Time For Stubborn Game Reductions

Ph.D Dissertation
Frederik Meyer Bønneland

Dissertation submitted February, 2021

Dissertation submitted:	February 2021
PhD supervisors:	Professor Jiri Srba Aalborg University
	Professor Kim Guldstrand Larsen Aalborg University
PhD committee:	Professor Kristian Torp (chairman) Aalborg University
	Professor Karsten Wolf University of Rostok
	Professor Doron Peled Bar Ilan University
PhD Series:	Technical Faculty of IT and Design, Aalborg University
Department:	Department of Computer Science

ISSN (online): 2446-1628
ISBN (online): 978-87-7210-907-7

Published by:
Aalborg University Press
Kroghstræde 3
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2021

Abstract

Computers have grown to encompass almost every aspect of human life. We expect that these systems operate seamlessly and without error, but this can sometimes be unrealistic. While testing is widely used to provide some assurance of fault-freeness, it is an incomplete approach that does not guarantee the absence of errors. For safety-critical systems, we require fault-freeness to safely deploy them in the real world, supported by formal methods such as model checking. However, high computational complexity caused by the state-state explosion problem encumbers model checking.

This thesis seeks to alleviate the state-space explosion and improve model checking's practical applicability by extending the theory of static partial order reductions to timed systems and games, and exploring orthogonal, complementary, and combined techniques.

Partial order reductions prune redundant interleavings of system actions. We extend static partial order reductions to timed systems by exploiting urgent states where time cannot elapse. We develop partial order reductions that preserves winning reachability strategies regardless of antagonistic environment actions. Lastly, we combine these two techniques into a single framework for timed games. We ensure the techniques are correct for general labelled transition systems, and instantiate the techniques to Petri net variants to novelly show significant time and space reductions with limited overhead on a set of case studies.

Next, we refine structural reductions for Petri nets and compare it with static partial order reductions. We show that the two technique are synergistic and allows for verifying more model checking problems.

To improve both structural reductions and static partial order reductions, we extend state equations for Petri nets to formula simplification of CTL formulae. Simplification significantly increases the applicability of both static partial order reductions and structural reductions on the database of models from the 2017 Model Checking Contest.

Resume

Computere er en del af næsten alle aspekter af vores moderne liv. Vi stoler på at disse systemer fungerer sømløst og uden fejl. Dette er dog ikke sandt i praksis. Test bruges omfattende i industrien til at give nogle garantier for fejlfrihed, men det er en ufuldstændig tilgang og kan ikke garanterer at systemer er fejlfri. For sikkerhedskritiske systemer har vi brug for stærkere garantier for at kunne sikkert indsætte dem i den virkelige verden. Formelle metoder såsom model checking er en tilgang til at give disse garantier. Problemet er at model checking er dyrt i forhold til tid og plads på grund af tilstandsrum eksplosionen.

Denne afhandling ønsker at lindre eksplosionen og forbedre den praktiske anvendelighed af model checking. Dette gøres ved at udvide statiske partial order reduktioner og undersøge supplerende metoder. Partial order reduktioner formindsker tilstandsrummet ved at fjerne overflødige sammenfletninger af samtidige system handlinger.

Vi udvider statiske partial order reduktioner til tidssystemer ved at udnytte tidspressede tilstande og udvikler partial order reduktioner som sikre at bevare vindende tilgængeligheds og sikkerheds strategier. Derudover, så kombinerer vi disse to tilgange til en samlet metode for tidsindstillede spil. Vi beviser at vores methoder er korrekt for generelle markeret transitions systemer, og viser på varianter af Petri nets væsentlige tids- og hukommelsesbesparelser.

Vi forfiner strukturelle reduktioner for Petri nets og sammenligner med statiske partial order reduktioner. Vi observere at de to metoder forstærker hinanden og giver mulighed for at verificere flere model checking problemer.

Vi forbedre både strukturelle reduktioner og statiske partial order reduktioner ved at udvide tilstandsligninger for Petri nets til formelforenkling af CTL formler. Formelforenkling øger betydeligt anvendeligheden af både statiske partial order reduktioner og strukturelle reduktioner. Vi viser dette på en model database fra Model Checking Contest 2017.

Contents

Preface

Beyond a way to make a living and the PhD project as a career path, it has offered me a lot more than what can be quantified. It has offered me self-realisation and a level of self-confidence I have not experienced before. For that, I am forever grateful for the opportunity.

I want to thank my supervisors Kim G. Larsen and Jiří Srba for the opportunity, their supervision, and for putting up with my awkwardness.

For the sparring and technical expertise, I want to thank my colleagues Marco and Peter in particular. I believe the work would have been a lot less impressive without your help.

For their accommodation and collaboration during my external stay at Uppsala University in the fall of 2019 I want to thank Bengt Jonsson, Parosh A. Abdulla, Mohamed F. Atig, Konstantinos Sagonas, Sarbojit Das, and the rest of the program verification group.

For their support through good and bad, I want to thank my parents Janna Bønneland and Torben Meyer Bønneland and my younger brother Søren Meyer Bønneland. I am very grateful that my father was able to see me obtain my masters degree. May he rest in peace.

I want to thank Mads Johannsen and Jakob Dyhr for the environment they provided during the last semester of my master studies. I believe I would not have been able to graduate in 2017 with a masters degree in computer science without this support.

For being my friend since HTX in Aarhus, I want to thank Martin Thøgersen. Even though we spend less time together when compared to before the PhD days, I never grow tired of that time.

Part I

Introduction

All aspects of modern society have rapidly been permeated by computers since the inception of the first digital computers in the twentieth century; from our production lines to our stovetops to our social interaction. Due to this, and commonly to all systems, we want to ensure that the systems adhere to their specifications and ideally are free of bugs and unintended behaviour. For this purpose, the strenuous task of *testing* is widely used in software development to ensure software conforms to their specification and detect the presence of bugs. However, while testing is effective at systemically detecting the presence of bugs, it is unable to prove the complete absence of bugs [46]. Many systems do not require such a strong guarantee of bug-freeness, such as the Google File System [59], where fault tolerance rather than fault-free is a central design choice. In the instances where the cost and risk of failure exceed what is worth or legal, fault-free behaviour is needed. In other words, *safety-critical systems* where there is a real risk of monetary losses or human lives in the case of errors, such as assembly lines or medical equipment.

Formal verification [116, 125, 109] is the act of rigorously proving that a system adheres to its specification. Recently, *formal methods* and *verification* have been used by, for example, Amazon [104, 7] and Facebook [31, 105] to verify the correctness of their massive and concurrent systems. *Model checking* [10, 37, 113, 36, 34, 82, 4] is one such approach to formal verification where, typically, a *transition system* is exhaustively generated and explored by a *model checker* program to prove that some *property* is verified. A transition system is a directed graph consisting of states connected with (possibly labelled) edges. States correspond to the possible configurations of the modelled system, and edges describe the system's behaviour. The model checker derives the transition system from a *model* that captures the system's behaviour. Examples of models can range from an abstract representation of the system, such as a *Petri net* [111] or a *timed automaton* [5, 86] to the source code of the system itself [2, 3, 9].

However, several challenges limit the viability of model checking. The modelling process itself can suffer from human errors, or the modelling formalism is insufficiently detailed. Even with an appropriate model, the computational complexity and decidability results of model checking are not encouraging [10, 50]. For example, consider a system that controls the lights in a railway network where the objective is to ensure that all trains reach their destinations safely. The actions of each light and train are at most points in the system *independent* of each other, but

each possible interleaving of actions is a part of the transition system and subject for exploration; a common situation when model checking is used for *distributed computing*. Furthermore, it is appropriate that we model this system as a real-time system that induces additional states for each possible configuration of time. Together this creates the so-called *state space explosion* problem. The explosion is further worsened if the system requires a detailed model to represent the problem correctly. Both timing and the train driver's uncontrollable actions modelled as an antagonistic opponent in a game have to be taken into account. This serves as the main hindrance to the viability of model checking as the exploration of the transition system becomes practically intractable. Approaches such as *randomised* [85, 64] and *statistical model checking* [90, 19, 27, 44] can provide probabilistic guarantees, but not absolute guarantees of bug-freeness which are required for safety-critical systems.

There have been several techniques suggested to alleviate the state-space explosion problem, which can broadly be put into three categories; *symbolic methods*, *abstractions*, and *structural methods* [34].

Symbolic methods [133, 135, 89, 66, 34] compress the state-space into representative sets of states (symbolic states), provide efficient operations on these sets, and avoid explicitly generating any concrete states of the state-space. Historically, *binary decision diagrams* (BBDs) [28, 29, 30] have been closely coupled with symbolic methods and used in symbolic verification tools such as NuSMV [33] and libraries such as CUDD [123] or BuDDy [40]. *Difference bounded matrices* (DBMs) [47, 14] have been used by tools such as UPPAAL [87, 86] and KRONOS [45] to canonically represent the symbolic states and efficiently verify timed automata [5, 86] with multiple clocks. Furthermore, *clock difference diagrams* (CDDs) [88, 12] are also used by UPPAAL to ensure the symbolic states are closed under set-union [13] when compared to DBMs. More recently, SAT-based model checking [16, 68, 75, 124, 76, 132] and satisfiability modulo theories (SMT) solvers [11, 32, 16] have grown popular using efficient SAT solvers to enhance verification. An example of SAT-based model checking includes *bounded model checking* [17], where a system is generated up to some number of execution steps based on a property and then passed to an SAT solver.

Abstractions [42] are approaches where a model is converted into a new form, carrying over specific properties of the original model depending on the given abstraction. Most notably, *counter-example guided abstraction refinement* (CEGAR) [42, 73, 55] is a popular algorithmic

abstraction method to model checking. An abstraction is refined based on generated counter-examples s.t. the refined abstraction covers the example. Eventually, either a true counter-example is generated, or every counter-example has been exhausted, implying the original model is verified. Examples of CEGAR applications include ICE (*Implications, Counter-examples, and Examples*) [54, 51, 55] which is a framework for invariant synthesis, and Sara [135] which is a tool for checking the reachability of Petri net markings. *Interpolation* is an approach for generating counter-examples by generalising proofs [96, 67, 97, 98].

Structural methods make use of the structure of the system, either its model or the derived transition system, to improve performance. *Symmetry reductions* [39, 49, 117] is a structural method that groups equivalent states of the transition system into a single representative state. Other structural methods include *parametric verification* for proving correctness disregarding the number of processes in the system [6, 1, 77, 56], *on-the-fly* state-space exploration where the state-space is generated lazily as the exploration proceeds [58, 41, 108, 128], or *compositional assume-guarantee reasoning* where a global property is divided into several smaller properties of smaller components or processes [115, 106, 84, 99, 74, 112, 93, 52, 78].

A family of structural methods that exploits independent actions are *partial order reductions* [60, 107, 130, 53]. The fact that the pioneers of the method Patrice Godefroid, Doron Peled, Antti Valmari, and Pierre Wolper were awarded the CAV award at the 2014 edition of the International Conference on Computer-Aided Verification [83] reflects the importance of partial order reductions. Since its conception in the nineties, there have been several variations to partial order reductions. Examples include stubborn sets [130, 127, 128], ample sets [107, 108], and persistent sets [60, 53, 62, 63] as the major variations. While there are several differences between these three approaches—such as stubborn sets allowing for disabled actions while persistent sets do not—they all have in common that they generate a representative subset of actions to consider in each state that is sufficient to preserve the correctness of a given model checking question.

Static partial order reductions refer to the classical approach to partial order reductions [62, 130, 107, 119]. In this approach, we generate the representative subset for a given state upon first visiting the state, usually by overapproximation [80, 21, 22, 130]. Optimal reductions are rare for static partial order reductions as only the syntax of the model

and the state itself are available to generate the subset. On the contrary, this limitation allows us to compute the subsets in an on-the-fly manner. Further analysis is usually counterproductive to the goal of making model checking more tractable. Alternatively, it is possible to improve static partial order reduction by reducing the model's complexity with techniques such as structural reductions for Petri nets [100, 102, 69, 126], or formula simplification [21, 136].

In contrast, *dynamic partial order reductions* [53] have been shown to generate optimal subsets [2, 3, 9]. In the dynamic paradigm, we add actions to the representative subset of a state as the descendants of the state are explored in a depth-first manner. Suppose an action leading to an descendant is shown to be conflicting with (or dependent on) a previously explored action. In that case, a backtracking point in the form of an action in a representative subset is added to a common ancestor. As the exploration backtracks to the state the subset is guaranteed to be representative and optimal. It is optimal in the sense that we only explore one trace from each class of *Mazurkiewicz traces* [95, 3]. This comes at the cost of limiting the applicability to acyclic systems.

While partial order reductions operate on individual states of the state space, *structural reductions* [100, 102, 69, 126] for Petri nets operate on the model itself and its syntax. Structural reductions for Petri nets consist of a set of rules s.t. if the precondition of a rule is satisfied, then several places, transitions, or tokens can be collapsed or pruned. This results in a smaller model, but more importantly, a smaller and more sparse state space. Furthermore, we can precompute the structural reductions before the computationally heavy state-space exploration.

So far, we have primarily considered model checking and partial order reduction of systems with full control of the system and without time constraints. However, both of these aspects are important for us to create an appropriate model. An example is the previously mentioned train scheduling problem. For example, lights have to be manipulated with certain timing constraints to avoid collisions, the lights are the controllable part, and the trains are the uncontrollable part. Respectively, this corresponds to *timed systems* [5, 65, 20] and *games* [18].

Timed systems in distributed computing have the disadvantage that time has a relevant effect on all actions of all system components as a global synchroniser, practically making all actions dependent. This was, until recently, reflected historically in the limited practical success of partial orders reductions for timed systems [15, 26, 131, 122, 92].

In particular, this is true for static partial order reduction [15] as we have no way to discern the effect of time from the vantage point of a single state. Recently, static partial order reduction approaches for timed automata with abstractions [66] and Timed-Arc Petri nets [24] have shown promising practicability by exploiting urgency.

Games in model checking involve synthesising a strategy that guarantees the property is satisfied. In other words, in each state, the strategy proposes a controllable action that ensures that the property is satisfied regardless of any possible uncontrollable action. For games, static partial order reductions have not received much attention. Partial order reductions for modal μ-calculus [134, 114] allow us in general to preserve the correctness of games. However, these reductions are general μ-calculus preserving reductions that may preserve properties not required in the game setting, limiting the reduction effectiveness. Recently, static partial order reductions have begun to show promise for Petri net games [25, 23] and parity games [103].

1 State-Space Reduction Example

In the following, we present an example of a Petri net model [111] and how we can use partial order reductions in conjunction with structural reductions [100, 102] and state equations [100, 101] for Petri nets. Petri nets is a popular mathematical and graphical modelling formalism for distributed systems [111]. This section is to provide intuition to the central topics of models, state-space representations, independence, and standard techniques for state-space reduction and verification.

1.1 Model

We start with some fundamental definitions and notations. Let $\mathbb{N}^0 = \mathbb{N} \cup \{0\}$ be the set of natural numbers and 0.

Definition 1 (Labelled Transition System)**.** *A Labelled Transition System (LTS) is a tuple $T = (\mathcal{S}, A, \rightarrow)$ where $\mathcal{S}$ is a set of states, A is a finite set of actions, and $\rightarrow \subseteq \mathcal{S} \times A \times \mathcal{S}$ is a transition relation.*

Whenever $(s, a, s') \in \rightarrow$ we write $s \xrightarrow{a} s'$ and say that a is *enabled* at s yielding s' when *executed*. The set of enabled actions in a state $s \in \mathcal{S}$ is given by $en(s) = \{a \in A \mid \exists s' \in \mathcal{S}.\ s \xrightarrow{a} s'\}$. If a is not enabled at s then we say that a is disabled at s. If $en(s) = \emptyset$ then s is a *deadlock*.

For a sequence of actions $w = a_1 a_2 \cdots a_n \in A^*$ we write $s \xrightarrow{w} s'$ if there exists $s_1, \ldots, s_{n-1}, s' \in \mathcal{S}$ s.t. $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s'$.

Definition 2 (Petri Net)**.** *A Petri Net is a tuple $N = (P, T, W)$ where P and T are finite sets of places and transitions, respectively, s.t. $P \cap T = \emptyset$, and $W : (P \times T) \cup (T \times P) \to \mathbb{N}^0$ is a weight function.*

A *marking* M is a function $M : P \to \mathbb{N}^0$ and $\mathcal{M}(N)$ denotes the set of all markings for N. A Petri net $N = (P, T, W)$ defines an LTS $T(N) = (\mathcal{S}, A, \to)$ where $\mathcal{S} = \mathcal{M}(N)$, $A = T$, and $M \xrightarrow{t} M'$ whenever for all $p \in P$ we have $M(p) \geq W(p,t)$ and $M'(p) = M(p) - W(p,t) + W(t,p)$.

As an example of a model using a Petri net, consider Figure 1. The circles are the places, and the rectangles are the transitions of the net. The net progresses by transitions *firing*, consuming tokens from all places with an incoming arc and producing tokens to all places with an outgoing arc. The number of tokens consumed and produced is given by the weight function W. For example, the transition t_1 consumes a single token from the place p_1, and produces a token at the p_5 and p_6 places each and two tokens at the p_2 place. A marking of a net is a possible allocation of tokens to the places of the net. For example, for the marking M shown in Figure 1 we have $M(p_1) = 1$ and $M(p_2) = 0$. Solid dots represent tokens within places, and in the initial marking there is only one token in the place p_1.

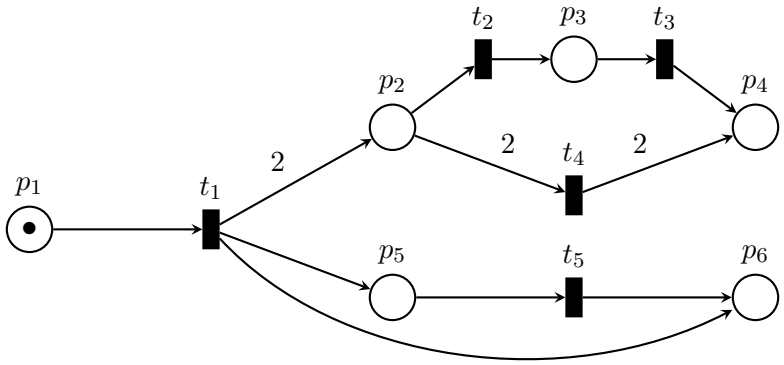


Figure 1: Example of a system modelled as a Petri net

In Figure 2 we have the *state-space* of our Petri net model seen in Figure 1 represented as a labelled transition system. The rectangles are the states, and each state is a marking that is reachable from the

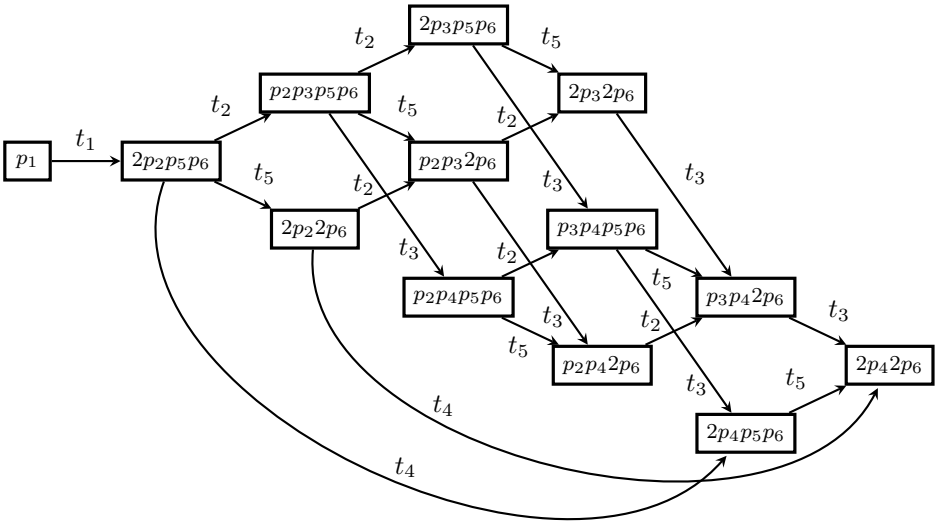


Figure 2: The state-space of Figure 1 as a labelled transition system

initial marking, which corresponds to the state annotated with p_1. The annotation within states is the tokens of the markings, where for example, $2p_2p_5p_6$ denotes a marking where there are two tokens in p_2, one token in p_5, and one token in p_6. We label each edge with the name of the appropriate transition.

1.2 Logic

The Petri net in Figure 1 is a model of a simple system where we initiate two processes when the transition t_1 is fired. The specification of this system is that it should be possible to reach a configuration where both processes terminate. We define the termination condition as when there are two tokens in both of the places p_4 and p_6. We can specify this as a formula in the *computation tree logic* (CTL) [38].

Let AP be a set of atomic propositions and let $v : \mathcal{S} \to 2^{AP}$ be a function s.t. $v(s)$ denotes the the set of atomic propositions satisfied in $s \in \mathcal{S}$. We fix the set of atomic propositions $\alpha \in AP$ for Petri nets as follows [79, 8]:

$\alpha ::= t \mid e_1 \bowtie e_2$

$e ::= \ c \mid p \mid e_1 \oplus e_2$

where $t \in T$, $c \in \mathbb{N}^0$, $\bowtie \in \{<, \leq, =, \neq, >, \geq\}$, $p \in P$, and $\oplus \in \{+, -, *\}$. The function v for a marking $M \in \mathcal{M}(N)$ is given as:

$$v(M) = \{t \in T \mid t \in en(M)\} \cup \{e_1 \bowtie e_2 \mid eval_M(e_1) \bowtie eval_M(e_2)\}$$

where $eval_M(e)$ is an evaluation function s.t. $eval_M(c) = c$, $eval_M(p) = M(p)$ and $eval_M(e_1 \oplus e_2) = eval_M(e_1) \oplus eval_M(e_2)$.

As an example, we have a marking M' for the Petri net in Figure 1 where $M'(p_2) = 2$. The expression $(p_2 * 2) - 1$ evaluates to $eval_{M'}((p_2 * 2) - 1) = 3$ and we have $M' \models (p_2 * 2) - 1 > 2$ and $M' \not\models (p_2 * 2) - 1 > 3$.

We present for simplicity and presentation purposes a subset of the CTL *syntax* and *semantics* in the following definition (where $\alpha \in AP$):

Definition 3 (Computation Tree Logic)**.** *A (subset of a) computation tree logic (CTL) formula is a formula φ with the following grammar:*

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid \alpha \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid EF\varphi \mid E(\varphi_1 U \varphi_2)$$

The semantics of a CTL formula φ is given as follows:

$s \models \mathit{true}$

$s \not\models \mathit{false}$

$s \models \alpha$ *iff* $\alpha \in v(s)$

$s \models \varphi_1 \wedge \varphi_2$ *iff* $s \models \varphi_1$ *and* $s \models \varphi_2$

$s \models \varphi_1 \vee \varphi_2$ *iff* $s \models \varphi_1$ *or* $s \models \varphi_2$

$s \models \neg\varphi$ *iff* $s \not\models \varphi$

$s \models EF\varphi$ *iff* $\exists w \in A^*$ *s.t.* $s \xrightarrow{w} s'$ *and* $s' \models \varphi$

$s \models E(\varphi_1 U \varphi_2)$ *iff* $\exists a_1 \cdots a_n \in A^*$ *s.t.* $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s'$ *and* $s' \models \varphi_2$ *and for all* i, $1 \leq i < n$, *we have* $s_i \models \varphi_1$

A *reachability formula* is a CTL formula $EF\varphi'$ where φ' can only include conjunction, disjunction, negation, α, *true*, and *false* of the CTL syntax and semantics.

As an example, for the initial marking M of Figure 2 consider the CTL formula $\varphi := E((p_1 = 1 \vee p_5 = 1) U (p_5 = 1 \wedge p_4 = 1))$. We have that $M \models \varphi$ is true due to the existence of the transition sequence $t_1 t_2 t_3$. First, we have to ensure that $p_1 = 1 \vee p_5 = 1$ is true until $p_5 = 1 \wedge p_4 = 1$ becomes true. In the initial marking we have that $p_1 = 1$ is true. After

t_1 is executed we remove the token p_1 and instead place a token in p_5 which ensures $p_5 = 1$. Next, after firing t_2, we remove a token from p_2 and produce a token in p_3. If we executed t_4 instead and produced two tokens p_4 then φ would not be satisfied since we cannot remove tokens from p_4. Lastly, we execute t_4 which places a token in p_4, which ensures $p_5 = 1 \wedge p_4 = 1$, and hence $M \models \varphi$.

Another specification expressed in CTL says that eventually the places p_4 and p_6 contains two tokens corresponds to the formula $\varphi := EF(p_4 = 2) \wedge (p_6 = 2)$. We read this as "there exists a run of the system where eventually there are two tokens in p_4 and two tokens in p_6". From the marking shown in Figure 1 it is possible to fire the transition sequence $t_1 t_4 t_5$ to reach a marking which places exactly two tokens in both p_4 and p_6, and φ is satisfied. A formula such as $EF(p_4 = 3)$ is not satisfied since there does not exist a reachable marking where p_4 contains three tokens.

To verify our formulae, we have to traverse the LTS of Figure 2. This traversal involves sufficiently exploring the reachable states of the LTS until a state is found that verifies our formula φ. In this case, the Petri net in Figure 1 is a bounded net, which results in the finite state-space seen in Figure 2 and a finite number of reachable states. However, it is possible that the state-space of even a simple net can have an infinite number of possible reachable states. In general, even for bounded nets, the state-space can be exponentially larger than the model it is derived from. Therein lies the challenge central to this thesis: How can we reduce the size of state-spaces?

1.3 Static Partial Order Reductions

We start with an introduction to the concept of *conditional independence* [34, 61]. Let $T = (\mathcal{S}, A, \rightarrow)$ be an LTS for the remainder of the section.

Definition 4 (Conditional Independence [61]). *A relation $I \subseteq A \times A \times \mathcal{S}$ is a* conditional independence relation *if for all $(a_1, a_2, s) \in I$ we have the following:*

- *If $a_1 \in en(s)$ and $s \xrightarrow{a_1} s'$ then $a_2 \in en(s)$ iff $a_2 \in en(s')$.*
- *If $a_1, a_2 \in en(s)$ then $s \xrightarrow{a_1 a_2} s'$ and $s \xrightarrow{a_2 a_1} s'$.*

If $(a_1, a_2, s) \in I$ then we say that a_1 and a_2 are *independent* at s, and otherwise they are *dependent* at s. The first condition states that two independent actions cannot disable nor enable each other at s, and the

second condition states that executing independent actions in any order results in the same state. In other words, they commute.

We note how the transition t_5 is independent of any of the t_2 and t_3 transitions. The actions of the two processes do not interfere with the operation of the other. In Figure 1, we can see this as the places t_5 consume a token from are disjoint from any place the other transitions consume from. In Figure 2, we can see this as the distinctive diamond shapes of the state-space. For example, from the state $2p_2p_5p_6$ firing t_2 and t_5 in any order will always produce exactly the state $p_2p_32p_6$ and neither will disable nor enable the other from being fired. Intuitively, independence implies that as long we explore one transition of a pair of independent transitions, then the effect of the other transition is not lost and can still be explored. In this specific case, the transitions t_2 and t_5 are always independent in any conceivable marking since neither disable nor enable the other. For each marking, we then define a representative set of actions to consider at that marking. For this purpose, we can either use *persistent sets* [61], *ample sets* [107], or *stubborn sets* [130]. We present for each of these approaches a basic definition. For persistent and ample sets we assume the LTS T is deterministic, i.e. for a state $s \in \mathcal{S}$ and action $a \in A$ if $s \xrightarrow{a} s'$ and $s \xrightarrow{a} s''$ then $s' = s''$.

Definition 5 (Persistent Set [61]). *A set $pers(s) \subseteq en(s)$ at $s \in \mathcal{S}$ is a* persistent set *of s iff for all $a_1, \ldots, a_n \in A \setminus pers(s)$ s.t. $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$ we have a_n is independent in s_n with respect to all transitions in $pers(s)$.*

Definition 6 (Ample Set [34]). *A set $amp(s) \subseteq en(s)$ at $s \in \mathcal{S}$ is a* ample set *of s iff for all $a_1, \ldots, a_n \in A$ s.t. $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$ we have if a_n is dependent on some $a \in amp(s)$ then there exists i, $1 \leq i < n$, s.t. $a_i \in amp(s)$.*

Definition 7 (Stubborn Set [130]). *A set $stub(s) \subseteq A$ at $s \in \mathcal{S}$ is a* stubborn set *of s iff for all $w \in (A \setminus stub(s))^*$ and all $a \in stub(s)$ if $s \xrightarrow{wa} s'$ then we have $s \xrightarrow{aw} s'$.*

Intuitively, all of these partial order reduction definitions appear similar. Both persistent and ample sets use independence to define $pers(s)$ and $amp(s)$, while for stubborn sets, we define the set $stub(s)$ slightly differently. Instead, stubborn actions can commute with any sequence of non-stubborn actions. Furthermore, stubborn sets can include disabled actions, while persistent and ample sets do not. Common for all

three variants, we have that an action is explored at a state only if it is both enabled and included in the persistent, ample, or stubborn set. We annotate the transition relation $\rightarrow$ as $\xrightarrow[pers]{}$ (and similarly for *amp* and *stub*) and have $(s, a, s') \in \xrightarrow[pers]{}$ if and only if $a \in en(s) \cap pers(s)$. If we restrict the exploration of the state-space to $\xrightarrow[pers]{}$ then this generates the *reduced state-space*. We say that a state s' is *reachable* from s if there exists $w \in A^*$ s.t. $s \xrightarrow{w} s'$, and similarly it is reachable in the reduced state-space if there exists $w' \in A^*$ s.t. $s \xrightarrow[pers]{w'} s'$.

The empty set is trivially a valid persistent, ample, and stubborn set for all of the above definitions. Therefore, we require additional conditions that ensure non-emptiness and, more importantly, preserve desired properties in the reduced state-space. As an example, we consider the reachability preservation of all deadlock states. For persistent and ample sets, it is sufficient to ensure that $pers(s)$ and $amp(s)$ are non-empty if s is not a deadlock, i.e. if $en(s) \neq \emptyset$ then $pers(s) \neq \emptyset$ and similarly for $amp(s)$. We call such persistent and ample sets for non-empty persistent and ample sets. For stubborn sets we require that if $en(s)$ is non-empty then $stub(s)$ must include a *key action*.

Definition 8 (Key Action [129])**.** *An action $a \in en(s) \cap stub(s)$ is a* key action *of $stub(s)$ for a state $s \in \mathcal{S}$ iff for all $w \in (A \setminus stub(s))^*$ where $s \xrightarrow{w} s'$ we have $a \in en(s')$.*

In other words, a key action cannot be disabled by non-stubborn actions. We call stubborn sets that includes a key action for *weak stubborn sets*. We now have the following theorem.

Theorem 1 (Deadlock Preservation [129])**.** *Let $s \in \mathcal{S}$ be a state. For all $w \in A^*$ s.t. $s \xrightarrow{w} s'$ and $en(s') = \emptyset$, if a reduced state-space is generated with $\xrightarrow[pers]{}$ where for all $s'' \in \mathcal{S}$ we have $pers(s'')$ is a non-empty persistent set then there exists $w' \in A^*$ s.t. $s \xrightarrow[pers]{w'} s'$. Similarly for non-empty ample sets and weak stubborn sets.*

Interestingly, we do not preserve deadlocks with stubborn sets unless key actions are included, while with persistent and ample sets, we preserve deadlocks in every non-trivial case. This implies that the definition above of stubborn sets is slightly weaker than persistent and ample sets [129]. A variation of stubborn sets called *strong stubborn sets*, where every enabled action in the stubborn set is required to be a key action,

corresponds to the above definitions of persistent and ample sets on deterministic systems (such as Petri nets) [60, 129]. However, for nondeterministic systems, stubborn sets are suitable while persistent and ample sets are not [129]. We prefer stubborn sets, and we consider it the chosen static partial order variant hereon.

Next we give intuition and an example for how a stubborn set can be generated for the marking $2p_2p_5p_6$ in Figure 2 with the formula $EF(p_4 = 2) \wedge (p_6 = 2)$. We select an initial set of *interesting* [22] transitions that have to be executed in order to reach a marking where $(p_4 = 2) \wedge (p_6 = 2)$ is satisfied. This can, for example, be the set $\{t_3, t_4\}$ since these are the only transitions that increase the number of tokens in p_4. We then iteratively add the transitions that violate Definition 7 to the set. We add t_2 because it can enable t_3, which is exemplified by the executable transition sequence $t_1t_2t_3$ whereas $t_3t_1t_2$ is not executable. We then have that the set $\{t_2, t_3, t_4\}$ satisfies Definition 7 and $\{t_2, t_3, t_4\}$ is a stubborn set. Alternatively, the set $\{t_5\}$ is also a viable stubborn set for this marking. For example, the transition t_5 commutes with any possible sequence consisting of any number of t_2 and t_3 transitions. The sequence $t_2t_3t_2$ followed by t_5 is equivalent to the sequence t_5 followed by $t_2t_3t_2$ in the sense that both leads to the same marking, and so on. It is therefore sufficient to only explore t_5 from the state $2p_2p_5p_6$ to cover all behaviour of the transitions t_2 and t_3.

Starting from the initial marking we can generate stubborn sets for states as we explore them. The space state can then be pruned as follows in an on-the-fly manner. If there is an outgoing arc labelled with a transition in the stubborn set, then explore it. Otherwise, we ignore it. The dashed states and edges in Figure 3 are the states and edges that we can prune using stubborn sets in this example. A total of five out of thirteen states are pruned from the full state-space, resulting in a 38,5% reduction.

As explained, stubborn sets are computed by iteratively adding transitions to a set of initially chosen transitions. These transitions are the ones that are relevant to verifying whether a formula is satisfied or not. For example, for the formula $\varphi := EF(p_4 = 2) \wedge (p_6 = 2)$ this corresponds to the transitions t_1, t_3, t_4, or t_5, since they change the number of tokens that the places p_4 and p_6 contain. These transitions are *visible* [60, 110, 121, 107, 127, 81] with respect to φ since there exists a state where firing them changes the evaluation of the conjunction. Conversely, every other transition is *invisible*. While including all visible transitions

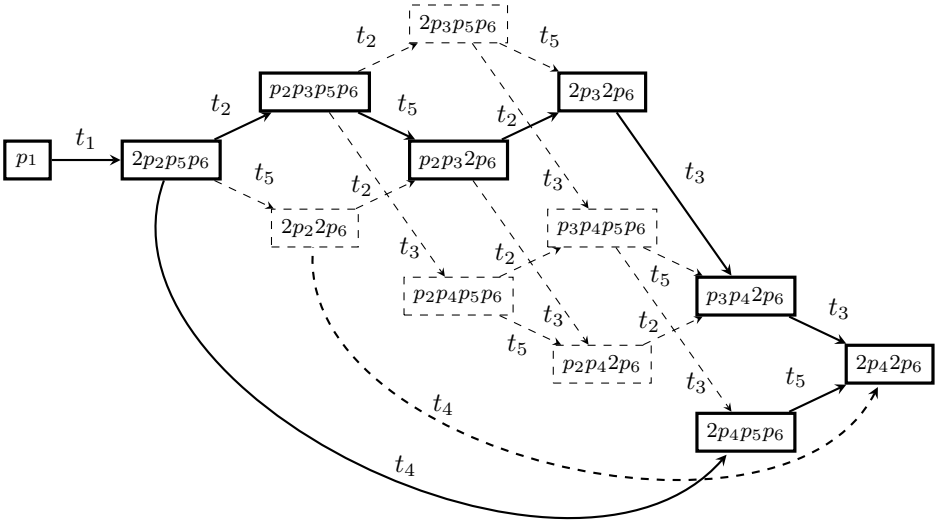


Figure 3: The pruned labelled transition system of Figure 2

in a stubborn set is sufficient to ensure verifiability of reachability formulae, it often leads to unnecessarily large stubborn sets. In Figure 1, if we include all visible transitions in the stubborn set, then no reduction is achieved at all. Instead, finer subsets such as *interesting sets* or *attractor sets* [80, 21, 22] are used, which are defined relative to a formula and a marking. For the formula φ and the marking seen in Figure 3, it is sufficient to include only the visible transitions of one side of the conjunction in φ, for example, t_1 and t_3.

Therefore, the formula limits the effectiveness of static partial order reductions. As the number of places in the formula increases, so does the number of initially chosen transitions. This enables an alternative way to improve state-space reductions by reducing the size and complexity of formulae, as shown in the following Subsection 1.5.

1.4 Structural Reductions

Next, we show how structural reductions [22, 69] introduced by Murata et. al [100, 102] can be used to reduce the size of Petri nets and enhance the possible state-space reduction in conjunction with stubborn sets.

Let $\mathit{places}(\varphi)$ denote the set of places that occurs in the formula

φ. For example, for the formula $\varphi := EF(p_4 = 2) \wedge (p_6 = 2)$ we have $\mathit{places}(\varphi) = \{p_4, p_6\}$. Structural reductions are enabled relative to a reachability formula φ s.t. if we apply the reduction rule and remove some places or transitions then we preserve the correctness of φ.

Definition 9 (Structural Reduction Correctness)**.** *Let $N = (P, T, W)$ be a Petri net, $M \in \mathcal{M}(N)$ be a marking, and φ a reachability formula. Let N' and M' be a modified N and M after applying some structural reduction rule. We say a structural reduction rule is* correct *with respect to φ if $M \models \varphi$ in N iff $M' \models \varphi$ in N'.*

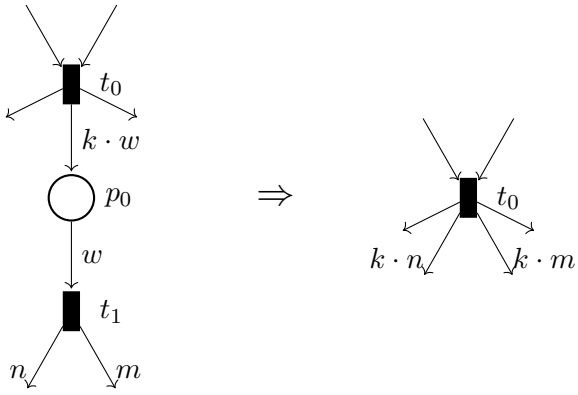
Figure 4a and 4b are examples of structural reduction rules from [22], which are refined to consider weights when compared to the original rules by Murata et al. [100, 102].

In Figure 4a we see the *sequential place removal* rule. The place p_0 and transition t_1 can be removed if t_0 is the only transition with an outgoing arc to p_0, the only outgoing arc from p_0 is to t_1, and the only incoming arc to t_1 is from p_0. Furthermore, in order to not change the correctness of φ we must have that $p_0 \notin \mathit{places}(\varphi)$ and either $k \cdot w = 1$ or $p' \notin \mathit{places}(\varphi)$ for all $p' \in \{p \in P \mid W(t_1, p) > 0\}$. If these conditions are met then we can remove p_0 and t_1. Lastly, we update the weight function for the outgoing arcs of t_0, as seen in Figure 4a. Intuitively, the rule removes temporarily storing tokens in p_0.

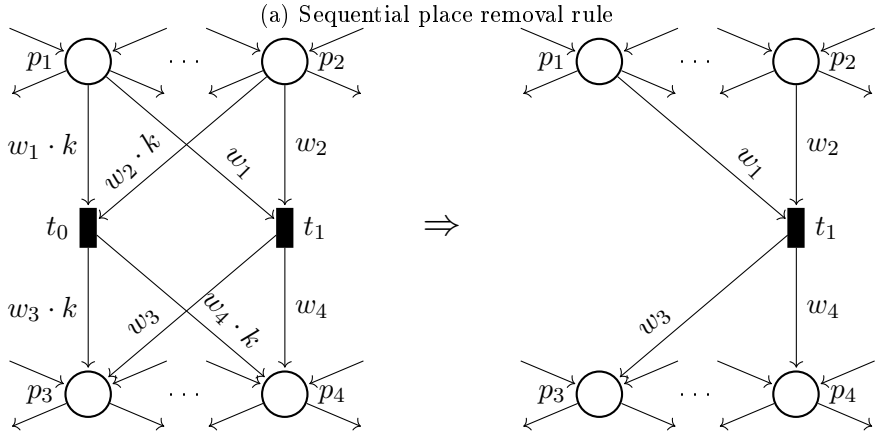
In Figure 4b we see the *parallel transition removal* rule. The transition t_0 can be removed if there exists $t_1 \in T$ and there exists $k \in \mathbb{N}$ s.t. $k > 0$ and for all $p \in P$ we have $W(p, t_0) = W(p, t_1) \cdot k$ and $W(t_0, p) = W(t_1, p) \cdot k$. Intuitively, if these conditions are met, we can achieve the effect of t_0 by firing t_1 exactly k times.

Theorem 2 (Correctness of Figure 4 [22])**.** *The structural reduction rules of Figure 4 are correct with respect to any reachability formula φ.*

Next, we apply the structural reduction rules in Figure 4a and 4b to our Petri net model in Figure 1. The place p_3 is redundant in the sense that together with the transitions t_2 and t_3 its only purpose is as an intermediate place before the token is transferred from p_2 arrives at p_4. Since there are no tokens in p_3 to begin with and the number of transferred tokens is one, we can collapse the place and two transitions into a single transition t'. This corresponds to the sequential place removal rule in Figure 4a. This transition t' consumes a single token from p_2 and produces a token at p_4. Next, we notice that the transition t_4 is now



(a) Sequential place removal rule



(b) Parallel transition removal rule

Figure 4: Structural reduction rules

redundant due to the newly created transition t'. Both transfers tokens from p_2 to p_4 where the only difference is the number of tokens. Since the number of tokens moved by t_4 is a multiple of the tokens moved by t', we can safely remove t_4. This corresponds to the parallel transition removal rule in Figure 4b. These reductions produce the Petri net seen in Figure 5, which is the structurally reduced Petri net of Figure 1. One of the more interesting features of structural reductions is that applying a reduction may allow for the detection of a previously undetectable redundancy, as was demonstrated in this example. This allows for iteratively applying the reduction rules until no more reductions are possible.

The state-space of Figure 5 can be seen in Figure 6. As can be observed, the effect of structural reductions has dramatically impacted

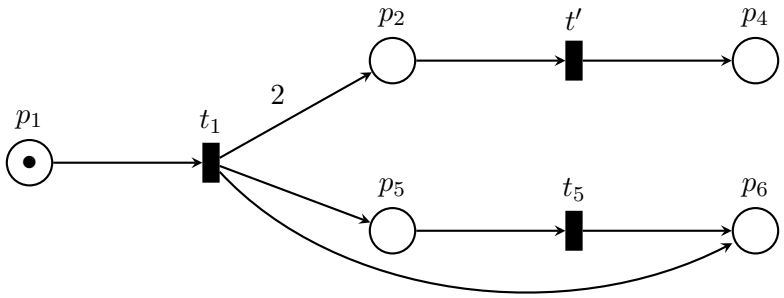


Figure 5: Structurally reduced Petri net of Figure 1

the size of the state-space. A total of six out of thirteen states have been pruned from the full state-space, resulting in a 46,2% reduction. Furthermore, the combination of structural reductions and static partial order reductions is complementary to each other. With static partial order reductions, we can prune two additional states. When compared to Figure 3, a total of eight out of thirteen states are pruned from the full state-space, resulting in a 61,5% reduction. This is a phenomenon we will see more of as we delve deeper into this thesis's contributions.

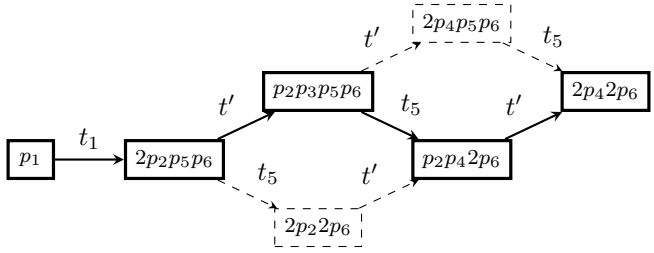


Figure 6: The state-space of Figure 5

Similarly to static partial order reductions, the formula limits the effectiveness of and structural reductions. As the number of places in the formula increases, the number of applicable structural reduction rules decreases.

1.5 State Equations

We now introduce the *state equations* [100, 101] approach for falsifying formulae. State equations is a Petri net overapproximation technique for checking the impossibility of reaching a given marking using integer

linear programming. We encode the reachability problem as an integer linear program with the implication that if the marking is reachable, then the integer linear program has a feasible solution.

Let us recall the basics of linear programming. Let $X = \{x_1, \ldots, x_n\}$ bet a set of variables and let $\overline{x} = (x_1 \cdots x_n)^T$ be a column vector. A linear equation has the form $\overline{c} \cdot \overline{x} \bowtie k$ where $\bowtie \in \{=, <, \leq, >, \geq\}$ is the set of possible equalities and inequalities, $k \in \mathbb{Z}$ an integer, and $\overline{c} = (c_1 \cdots c_n)$ is a row vector of integer constants s.t. $c_1, \ldots, c_n \in \mathbb{Z}$. A *linear program* LP is a finite set of linear equations. A *solution* to LP is a mapping $u : X \to \mathbb{R}^{\geq 0}$ from variables to nonnegative real numbers. Let $\overline{u} = (u(x_1) \cdots u(x_n))^T$ be a column vector. A solution u is *feasible* if for every linear equation $\overline{c} \cdot \overline{x} \bowtie k \in LP$ we have $\overline{c} \cdot \overline{u} \bowtie k$ is true, and we say that LP is feasible. Alternatively, if u is a mapping from X to $\mathbb{N}^0$ then u is a feasible integer solution. Feasibility of a linear program can be solved in polynomial time, while the existence of a feasible integer solution is an NP-complete problem [120].

As a concrete example, consider if it is possible to reach a marking of Figure 1 where the place p_6 has three tokens, and every other place has zero tokens. By inspection, it is clear that such a marking is not reachable from the initial marking. However, this can be proven using state equations.

Theorem 3 (State Equations [100])**.** *Let $N = (P, TW)$ be a Petri net and $M, M' \in \mathcal{M}(N)$ some initial and target markings of N. Let $X = \{x_t \mid t \in T\}$ be a set of variables. We then have that if there exists $w \in T^*$ s.t. $M \xrightarrow{w} M'$, then the following system of equations has a feasible solution over the variables X.*

$$M(p) + \sum_{t \in T} (W(t,p) - W(p,t))x_t = M'(p) \qquad \text{for all } p \in P$$

The variables correspond to the transitions of the Petri net in Figure 1. Each equation in the program corresponds to a place and constraints the number of times transitions can be fired. Assigning a number to one of the variables abstractly models the number of times a transition is fired. This is due to the $(W(t,p) - W(p,t))x_t$ part of the equations that multiplies x_t with the difference between the number of tokens produced and consumed at p.

Let M be the initial marking seen in Figure 1, and let M' be a marking s.t. $M'(p_6) = 3$ and $M'(p_1) = M'(p_2) = M'(p_3) = M'(p_4) = M'(p_5) = 0$. Writing the equations individually for M', we get the

following:

$$
\begin{aligned}
1 - x_{t_1} &= 0 && \text{corresponds to place } p_1 \\
2x_{t_1} - x_{t_2} - 2x_{t_4} &= 0 && \text{corresponds to place } p_2 \\
x_{t_2} - x_{t_3} &= 0 && \text{corresponds to place } p_3 \\
x_{t_3} + 2x_{t_4} &= 0 && \text{corresponds to place } p_4 \\
x_{t_1} - x_{t_5} &= 0 && \text{corresponds to place } p_5 \\
x_{t_1} + x_{t_5} &= 3 && \text{corresponds to place } p_6
\end{aligned}
$$

This program does not have a feasible solution. The sixth equation requires either x_{t_1} or x_{t_5} to be greater than one. At most, we can assign the variable x_{t_1} one; otherwise, the first equation is not satisfied. Therefore, we must have that x_{t_5} is assigned two. Due to the fifth equation, we have $x_{t_1} = x_{t_5}$, and the linear program is infeasible. We can infer that the marking M' is unreachable from the initial marking in Figure 1.

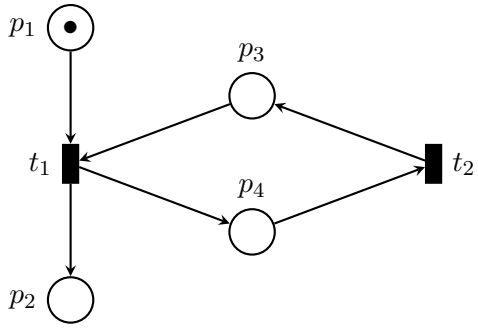


Figure 7: A Petri net state equations example

However, state equations are an overapproximation technique. A feasible solution is a necessary condition for marking reachability [100], but this does not imply that the marking is reachable. We can attempt to find a feasible integer solution for more precision, but that is not an exact approach either. Consider the Petri net show in Figure 7. We want to check if it is possible to reach a marking where there is a token in p_2 and zero tokens in every other place. This results in the following linear program:

$$
\begin{aligned}
1 - x_{t_1} &= 0 && \text{corresponds to place } p_1 \\
x_{t_1} &= 1 && \text{corresponds to place } p_2 \\
x_{t_2} - x_{t_1} &= 0 && \text{corresponds to place } p_3 \\
x_{t_1} - x_{t_2} &= 0 && \text{corresponds to place } p_4
\end{aligned}
$$

Clearly, the integer solution $x_{t_1} = x_{t_2} = 1$ is a feasible solution to this program but the place p_2 can never be marked because the marking in Figure 7 is a deadlock.

2 Contributions

With the notions of static partial order reductions, structural reductions, and state equations introduced in Sections 1.3, 1.4, and 1.5 we now present how we use and extend upon these techniques. Specifically, in this section, we present the papers and their contributions, which constitute the remainder of this thesis. The following introduces the papers and their details regarding authors and publication status:

A *Start Pruning When Time Gets Urgent: Partial order Reduction for Timed Systems.* Published in the proceedings of the *30th International Conference on Computer Aided Verification (CAV'2018)* [24].
Authors: Frederik M. Bønneland, Peter G. Jensen, Kim G. Larsen, Marco Muñiz, and Jiří Srba.

B *Stubborn Set Reduction for Two-Player Reachability Games.* Invited to a special issue of the journal of *Logical Methods in Computer Science* which is accepted with minor revisions [25], and published in the proceedings of the *30th International Conference on Concurrency Theory (CONCUR'2019)* [23].
Authors: Frederik M. Bønneland, Peter G. Jensen, Kim G. Larsen, Marco Muñiz, and Jiří Srba.

C *Stubborn Set Reduction for Timed Reachability and Safety Games.* Submitted for publication.
Authors: Frederik M. Bønneland, Peter G. Jensen, Kim G. Larsen, Marco Muñiz, and Jiří Srba.

D *Stubborn Versus Structural Reductions for Petri Nets.* Published in a special issue of the *Journal of Algebraic Methods in Programming, volume 102 (2019)* [22] and initially accepted for the *29th Nordic Workshop on Programming Theory (NWPT'2017).*
Authors: Frederik M. Bønneland, Jakob Dyhr, Peter G. Jensen, Mads Johannsen, and Jiří Srba.

E *Simplification of CTL Formulae for Efficient Model Checking of Petri nets.* Published in the proceedings of the *39th International Conference on Application and Theory of Petri Nets and Concurrency (Petri'2018)* [21].
Authors: Frederik M. Bønneland, Jakob Dyhr, Peter G. Jensen, Mads Johannsen, and Jiří Srba.

Henceforth, we use the respective letters to refer to each paper. Each subsection corresponds to one of the featured papers and presents challenges within the field of model checking and how we contribute towards these challenges.

2.1 Timed Systems

Static partial order reductions were a fundamental contribution when it was initially developed in the nineties. However, practical applicability has not been able to keep up as models become more complex and detailed, including time. Timed models render all actions dependent if not handled differently due to the existence of a clock as a global synchroniser [15, 26, 131, 122, 92]. The modelling of time is essential for formally verifying many classes of systems, such as reactive systems. In this section, we present a theoretical framework and implementation of static partial order reductions that have shown practical applicability in timed systems.

In paper **A** we develop a framework that supports partial order reductions for timed systems in the form of general timed labelled transition system. We focus on systems that exhibit urgent behaviour, which is a common feature of many timed systems. Lastly, we showcase its practical viability in model checking.

As an example, consider the Timed-Arc Petri net [65, 20] shown in Figure 8. It is an altered version of the Petri net shown in Figure 1 that we have equipped with syntactic elements relating to time. We can still represent its state-space as a timed labelled transition system by including the real numbers as possible delay actions. Tokens are now represented by their age. Delay actions progress the age of all tokens in the net evenly. Initially, and when the net produces tokens, we give tokens an age of 0. In the initial marking in Figure 8 we have a single token in p_1 of age 0. A place may have an invariant, such as p_2, limiting the age of tokens allowed in the place. Transitions may be urgent such as t_3 represented by a white inscribed circle within the rectangle.

Furthermore, transitions may only accept to consume tokens which age is within a certain interval, such as t_5. Urgency occurs when either an urgent transition is enabled or there exists a token in the net that is not allowed to age due to the invariant of the place it is located in.

The specific marking shown in Figure 9 is the result of firing t_1, delaying by 2, and then firing t_2 if the initial marking is as shown in Figure 8. This marking is urgent both because t_3 is enabled and there is a token of age 2 in p_2. In this case, our approach can reduce the state-space by, for example, producing a stubborn set where we include t_3, but not t_5.

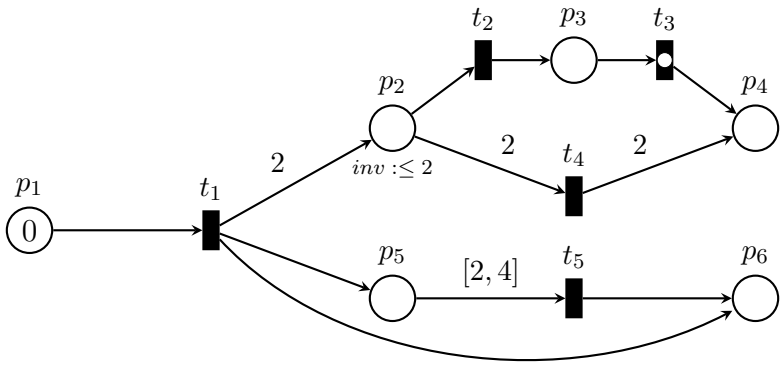


Figure 8: Example of a system modelled as a Timed-Arc Petri net

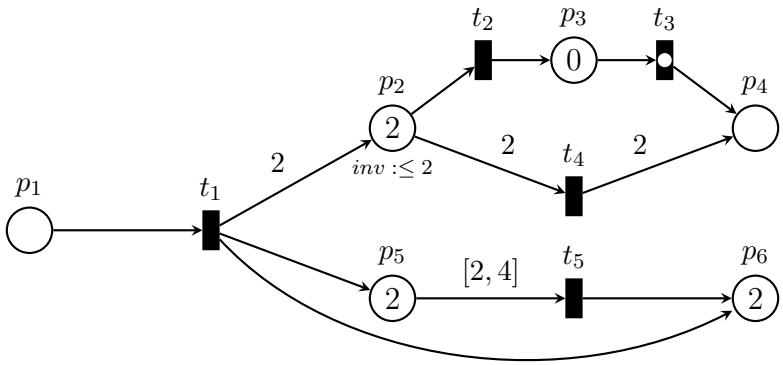


Figure 9: The derived marking if the sequence $t_1 2 t_2$ is executed from the initial marking in Figure 8

Next, we informally introduce our static partial order reduction ap-

proach to timed systems, such as the Timed-Arc Petri net shown in Figure 8. On the general class of timed labelled transition systems, we introduce partial order reductions conditions relating to time. We prove that if a reduction satisfies our conditions and the classical conditions of stubborn sets, then it preserves reachability for a given subset of states. Central to the approach is the observation that if a state is urgent, we can use classical partial order reduction rules for untimed systems freely. If a state is not urgent, i.e. time is allowed to progress, then we do not attempt to reduce the state-space from that state. This effectively results in a partial order reduction approach to timed systems that is not slowed down due to handling the impact time has on the interaction of independent actions.

Contribution 1

We introduce new static partial order reduction conditions on a general formalism of timed systems with either discrete or continuous semantics. The novelty is that the conditions exploit urgent behaviour, which is a common feature when modelling timed systems. We prove that the approach preserves the reachability of a given set of target states.

A theoretical result, while interesting, is insufficient in the area of partial order reductions since its purpose is to alleviate the practical limitations of model checking. We instantiate our approach to the formalism of Timed-Arc Petri nets and develop an algorithm that exploits the net's timing information to refine the reduction further. We show that the algorithm is terminating and correct, and we implement it in the model checking tool TAPAAL [43, 69]. Through several industrial case studies, we demonstrate the practical viability of the approach.

Contribution 2

We demonstrate our approach's practical applicability to partial order reductions for timed systems on the formalism of Timed-Arc Petri nets on an implementation in the model checking tool TAPAAL. Our evaluation on industrial case studies shows exponential speed up in the best case, and in the worst case, the slowdown is negligible.

Timed Systems Future Work. Potential future work is to relax the requirement that prevents the reduction from being applied exclusively in urgent states.

2.2 Games

Another example of a relevant modelling formalism is games. A game consists of a controllable part and a part of the system that is either uncontrollable or antagonistic. These are referred to respectively as the controller and the environment. We designate a subset of states as *goal states*. A *controller strategy* is a function that, given a state, returns an enabled controllable action if one exists, and otherwise nothing. An *environment strategy* is a function that either returns an enabled uncontrollable action or nothing. A controller strategy is a *winning strategy* relative to a given state if following the strategy ensures we eventually reach a goal state regardless of any action proposed by any possible environment strategy. If this is the case, then the state is said to be a *winning state* Trivially, all goal states are winning states. If the game's initial state is a winning state, then it is said to be a *winning game*. The winning strategy synthesis problem is important for verifying reactive systems with imperfect information or uncontrollable agents.

As an example of a game, consider the Petri net game shown in Figure 10. The only difference from Figure 1 is that we have partitioned the transitions into controller and environment transitions. Transitions represented as a solid rectangle are controller transitions, and transitions with a white filling are environment transitions. The two processes initiated by t_1 are now modelled as controllable and uncontrollable, respectively. A state is a controller state if only controller transitions are enabled, and similarly for environment transitions.

The literature on partial order reductions for game-theoretic formalisms is limited. Partial order reduction for the μ-calculus and the temporal logics LTL and CTL [127, 94, 91, 57, 48, 134, 114] makes it possible to encode the game semantics as a μ-calculus formula. Furthermore, partial order reductions for parity games have been developed [103]. However, similarly to timed systems, partial order reductions for games have had limited practical results or generality.

In paper **B** we develop a framework that supports partial order reductions for two-player reachability games in the form of general game labelled transition systems. The two players referred to hereafter as the *controller* and the *environment*, respectively, have opposing objectives

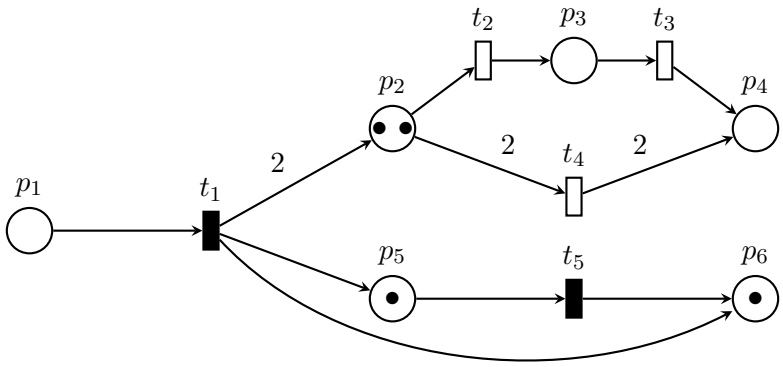


Figure 10: Example of a system modelled as a Petri net game

in this game. The controller seeks a strategy that ensures that a state from a subset of goal states is reached no matter the actions of the environment. In contrast, the environment wishes to find a strategy that counters the controller.

We introduce distinct conditions for both the game participants that are applied, together with the classical conditions of stubborn sets, whenever either player has exclusive control of the system. In other words, we do not attempt to reduce the state-space in states where both players have an enabled action. We call such states *mixed states*. If there is a winning strategy for either participant of the game, then our conditions also preserve the existence in the reduced game. This implies not only that winning strategies for the controller are preserved but also winning strategies for the environment.

Contribution 3

We introduce a partial order reductions approach to a general formalism of two-player games and present sufficient conditions that preserve winning strategies for both participants in the game. Furthermore, we show that the conditions can be relaxed in a non-mixed game setting.

We instantiate the framework to the formalism of Petri net games and implement it in the model checking tool TAPAAL in conjunction with its strategy synthesis algorithm [71, 72, 70]. Similarly to the work

on timed systems, we want to achieve practical viability of our framework. However, several of the introduced game-specific conditions can require an expensive analysis of the net if we do it precisely. Striking the right balance between analysis and approximations is fundamental for bringing static partial order reduction techniques from their theoretical frameworks to model checking's practical reality. To achieve practical viability of our framework for Petri net games, we develop and show the correctness of algorithms that overapproximate the aforementioned conditions. We refer to paper **B** at the end of this thesis for details. We also developed an overapproximation algorithm for preserving infinite paths of exclusive environment transitions [23]. However, this algorithm was indeed discovered as redundant for the purpose of preserving winning strategies [25].

Contribution 4

We develop new overapproximation algorithms for preserving infinite cycles and checking the feasibility of reaching a goal state. In both cases we prove the algorithms are terminating and correct.

We showcase the viability on several case studies that show significant to exponential speedups and memory savings. These case studies include, among others, a train scheduling problem, workflow models, and the producer-consumer system reformulated as a Petri net game.

Contribution 5

We implement the framework in the model checking tool TAPAAL in conjunction with its strategy synthesis algorithm. We demonstrate the practical applicability on several case studies that show significant time and memory savings in the strategy synthesis algorithm.

Games Future Work. The requirement of not attempting to reduce in mixed states can be too strict and should be relaxed. Consider the previously informally introduced uncontrollable train game where trains move around in a railway network while being directed by the controllable lights. Naturally, any light's action is independent of any train that

is currently not at the part of the railway where the light is located. However, our framework deems them dependent by default. To make the best use of our partial order reduction technique, it requires model creators to design their models with the technique in mind. This is not favourable since it results in less natural models and hinders the options available to modellers. Relaxing this condition will enable the reduction to be useful on more natural and a larger quantity of models and make the technique more seamless.

As a more concrete example of this problem, consider the Petri net game seen in Figure 10. Despite how the controller and environment process are always independent of each other, no reduction is possible at all. This is because any pair of enabled independent transitions are controller and environment transitions, respectively. Therefore, in any state where reduction is possible, we default to include every enabled transition in the stubborn set. This produces the entirety of the state-space seen in Figure 3. Altering the model to make state-space reduction possible involves removing the two processes' concurrency, practically linearising the model, which is neither natural nor intuitive for the modeller.

2.3 Timed Games

As we have seen, timed systems and games are relevant modelling formalisms for correctly modelling reactive systems and the presence of imperfect information and control. Naturally, a combination of timed systems and games in the form of *timed games* is also relevant.

As an example, consider the Timed-Arc Petri net game in Figure 10. A Timed-Arc Petri net game is a combination of the syntax for Timed-Arc Petri nets and Petri net games as seen in Figure 9 and 10. We partition all transitions into controller and environments stations, tokens have an associated age that progresses evenly, and we can prevent time from elapsing with place invariants and urgent transitions, etc.

Since the literature on static partial order reductions for timed systems and games are limited as discussed in Section 2.1 and 2.2, then this scarcity carries over into timed games as well. In paper **C** of this thesis we seek to change that. We combine the frameworks presented in paper **A** and **B** into a unified framework for timed games. On a high level, this combination reduces to taking the union of the conditions developed in the timed and game-theoretic setting, respectively. The main technical correctness lemmas require non-trivial changes due to the introduction of time into the game setting. In the untimed setting the correctness proof

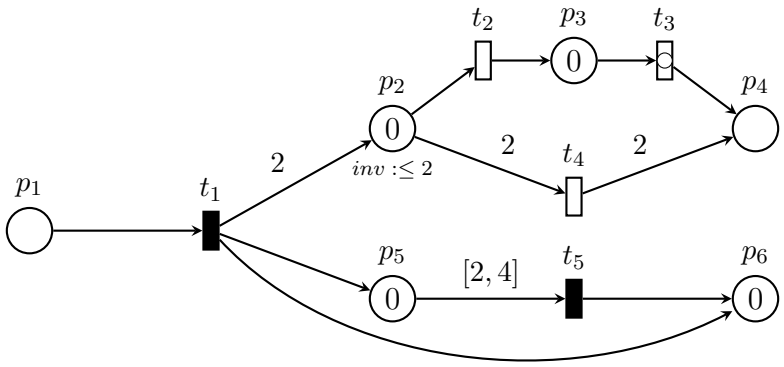


Figure 11: Example of a system modelled as a Timed-Arc Petri net game

relies on König's lemma to ensure a game subject to a winning strategy has a maximal depth before it reaches a goal state. The correctness proof then proceeds by induction on the depth of a winning strategy. However, with the addition of time, every non-urgent state can potentially have infinite branching due to time delays. This is the case with both discrete and continuous time semantics. Therefore the maximal depth cannot be ensured by König's lemma anymore. Furthermore, we prove that our framework for preserving winning reachability strategies also preserves winning safety strategies for both players, with minimal changes.

Contribution 6

We nontrivially extend the correctness proof in the game-theoretic setting to the combined time and game setting. We also further refine the correctness proof in the untimed game setting to not rely on the depth of winning strategies in its inductive argument. We prove that in addition to reachability strategies we also preserve winning safety strategies.

We can combine the two techniques by interleaving the various procedures used to ensure the combined set of conditions. We do this on the practical level of our chosen formalism of discrete-time Timed-Arc Petri net games. The sets produced by each procedure can then be unioned into a single set and iteratively made into the final stubborn set. Furthermore, the introduction of time allows us to refine the overapproximation introduced for the game setting. This is due to the introduction

of syntactic elements from the timed setting, such as guards, invariants, and clocks. Specifically, in the Timed-Arc Petri net formalism, if two transitions consume tokens from a shared place, but the interval of the accepted token ages is disjoint, then they can be determined to be independent. When transitions produce tokens, their initial clock value is set to zero. Therefore, an infinite sequence of exclusively environment transitions has to accept zero ages tokens or self-supply appropriately aged tokens. These are two possible refinements possible in the timed game setting. For more details we refer to paper **C** at the end of this thesis.

Contribution 7

We combine the frameworks we developed for the timed and game settings into a unified framework for timed games. We observe that the combination allows for further refinement of the stubborn set generation for Timed-Arc Petri net games.

We develop and prove the correctness of an algorithm for generating winning strategy preservation for timed games instantiated to Timed-Arc Petri net games. We implement the algorithm in the model checking tool TAPAAL [43, 69], and perform an experimental evaluation on a set of case studies. We observe increasing and potentially exponential time and memory savings when reduction is possible as the cases scale to larger instances.

Contribution 8

We implement the combined framework in the model checking tool TAPAAL. We show a promising experimental evaluation with a potential for exponential time and memory savings as the case studies scale to larger instances.

Timed Games Future Work. Since the framework for timed games is the product of combining two separate frameworks, then the stated future work for our partial order reduction techniques for timed systems and games also applies here. Going back to the problematic example Figure 10 from the game setting, consider the altered model in Figure 11 with added syntactic elements relating to time. The specific marking

shown in Figure 11 is an environment and urgent making. The marking is an environment marking because the controller transition t_5 is not enabled due to the token's age in p_5. The marking is urgent because the urgent transition t_3 is enabled. A reduction is possible since a stubborn set which includes t_3 but not t_2 is a proper stubborn set. However, this is one of the very few markings where reduction is possible in this model. In most other cases, the marking is either non-urgent or mixed, and we include every enabled transition in the stubborn set. Relaxing the conditions for non-urgent and mixed states remains essential.

2.4 Structural Reductions

While static partial order reductions show great potential, there may exist possible reductions that are not detected by partial order reductions but are detected and pruned by structural reductions [100, 102, 69, 126] in the context of Petri nets. This was shown in the introduction example of Figures 1, 3, 5, and 6. Furthermore, the combination of the two approaches are synergistic and allows for a greater reduction. In this section, we introduce our contribution to the technique of structural reductions.

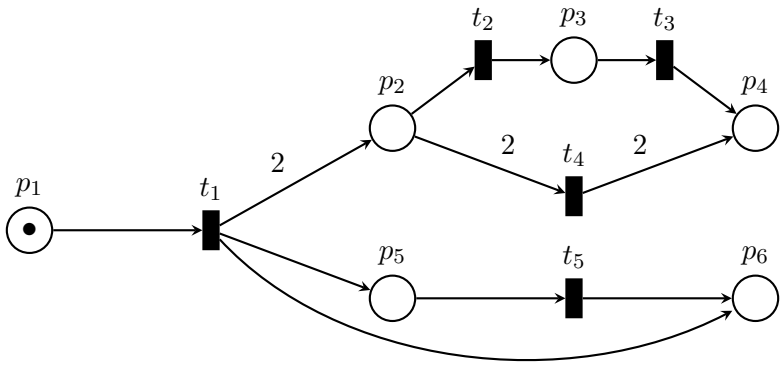


Figure 12: A Petri net

In paper **D** we present a comparison of the static partial order reductions and structural reductions for Petri net individually and the interaction when they are combined. We introduce new structural reduction rules and refine all rules to take arc weights and inhibitor arcs into account. In Section 1.4, we saw an example of structural reductions collapsing places and transition to produce a smaller net. As an exam-

ple of an introduced structural reduction rule, consider the Petri seen in Figure 12. Assume that the model checking formula we are verifying is the reachability formula $\varphi := EF(p_6 = 2)$, i.e., we are only interested in if one of the processes can terminate. In this case, we can use structural reductions to remove part of the Petri net irrelevant to verify the formula. We do this iteratively, starting from the set of transitions that can either increase or decrease the number of tokens in p_6. This corresponds to the set $\{t_5\}$. This set is then iterated upon, adding all the transitions that may enable any transition in the set. This iteration occurs once, producing the set $\{t_1, t_5\}$, and a fixed point has been reached. Lastly, all places that either occur in the formula or have an outgoing arc to any discovered transitions are found. This corresponds to the set $\{p_1, p_5, p_6\}$. The rule proceeds as follows: any transition or place not among the transitions and places found can safely be removed while preserving correctness of φ. In Figure 12, the removed places and transitions are p_2, p_3, p_4, t_2, t_3, and t_4. This results in the Petri net seen in Figure 13. For the specifics of this procedure of this other structural reduction rules, we refer to paper **D** at the end of this thesis

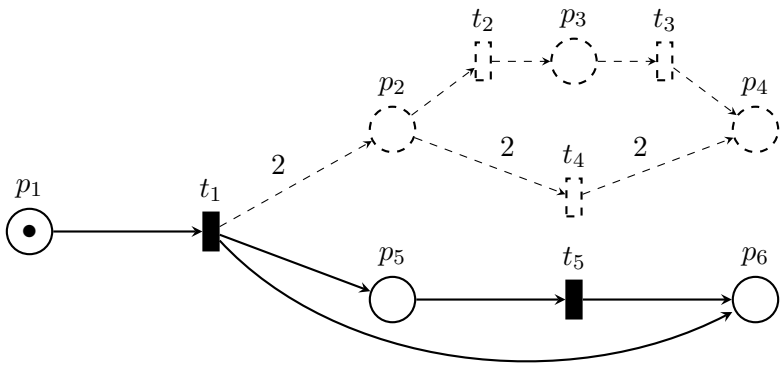


Figure 13: The Petri net from Figure 12 with all irrelevant places and transitions removed given the formula $EFp_6 = 2$

We formally prove the correctness of each structural reduction rule, including the rule shown in Figure 13. We define correctness with respect to the reachability logic used in the model checking contest (MCC) for the verification of Petri nets [79, 8]. Every rule preserves the validity of reachability formulae and a subset of the rules also preserves the validity of deadlock reachability.

Contribution 9

We introduce new structural reduction rules and refine all the rules to take arc weights and inhibitor arcs into account. We formally prove the correctness of the structural reductions rules for Petri net with respect to reachability and deadlock formulae.

Every structural reduction rule is implemented in the model checking tool TAPAAL [43, 69]. We compare the extended structural reductions against static partial order reductions and their combination. We perform the evaluation using the Petri net models and formulae database from the 2017 edition of the MCC [79]. Both techniques improve the number of formulae that we can solve for all categories of formulae. However, the combination of the techniques further increases the number of solved formulae. The evaluation indicates that while there is some overlap between the techniques, they still complement each other to form a stronger unified model checking procedure. Lastly, we compare on the same set of models and formulae the implementation in TAPAAL against the model checking tool LoLA [118, 136], the winner of the 2017 edition of the MCC. The results show that the combination, together with the pre-existing TAPAAL techniques, can solve more formulae than LoLA.

Contribution 10

We show that the combination of static partial order reductions and structural reductions are complementary of each other. We observe that the combined techniques implemented in TAPAAL result in solving more formulae than LoLA on the MCC'17 database of models and formulae.

Structural Reductions Future Work. Potential future work includes how structural reductions can be extended beyond standard Petri nets and applied to formalisms such as coloured Petri nets, Timed-Arc Petri nets, and Petri net games.

Furthermore, following the work of Thierry-Mieg [126], there are possibilities to develop more extensive structural reduction techniques by balancing processing time versus effectiveness.

2.5 Formula Simplification

As we discussed in Section 1.4, the effectiveness of static partial order reductions and structural reductions can be limited not only by how the model itself is constructed. Instead, the formula can be the limiting factor. The formula guides the model checking and the exploration of a state-space. We perform most static partial order reductions relative to this formula to ensure that we guarantee to the validity of the formula. This comes in the effect of defining which actions are respectively visible or invisible to the formula. In other words, an action is visible if its execution from an arbitrary state may affect the formula. If an action is determined to be visible, then we must eventually explore it. Another example is attractor sets [81, 80] or interesting sets [21, 22] which are defined relative to a given state. While attractor sets and interesting induces an additional overhead for each state during state-space exploration, they perform in general better than just visible actions for preserving reachability. In either case, the formula's size can result in a majority of actions branded as visible or interesting, all of which must be included. This hinders the effectiveness of static partial order reductions, as each action added to the stubborn set may result in the addition of dependent actions. For structural reductions, each place in the formula is exempt from being affected, limiting the rules' applicability. Reducing the formula size is then an orthogonal approach to improve both static partial order reductions and structural reductions. This section introduces a novel technique for reducing the complexity of model checking of Petri nets on a semantic level.

In paper **E** we present techniques for reducing the size of computation tree logic (CTL) formulae for Petri nets, which we call formula simplification. This consists of two smaller techniques where a formula is either verified or rewritten based on the initial marking, and a larger technique based on determining whether subformulae are satisfied or not using integer linear equations in the form of generalised state equations [100, 101]. The result is an equivalent formula in terms of validity that is smaller and simpler than the original formula. This improves the effectiveness of static partial order reductions and structural reductions and the model checking procedure itself. In some cases, we can simplify CTL formulae to a reachability formula that is significantly easier to verify. Formula simplifications have previously been studied as part of the verification tool LoLA [118, 136]. However, we take the simplification further by recursively analysing each subformulae and its negation of the

formula.

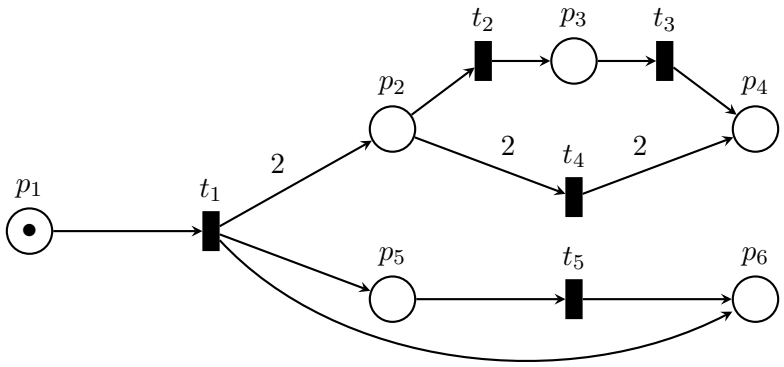


Figure 14: A Petri net

As an example of CTL formula simplification, consider the Petri net in Figure 14 with the CTL formula $\varphi := E((p_5 \leq 1)U(p_4 = 2))$. We read this formula as "there exists a run of the system where there are one or less tokens in p_5 until there are two tokens in p_4". For the initial marking seen in Figure 14 this formula is initially true since there are zero tokens in p_5. In Section 1.5 we saw how a reachability formula can be proven false without exploring the state-space by using state equations [100, 101]. Here our formula φ is a CTL formula, so we cannot use the same approach to verify φ. Instead, we show how the subformula $p_5 \leq 1$ can be reduced to *true* due to the impossibility of verifying the negation. Furthermore, we show how a CTL formula can be simplified to a reachability formula. The negation of $p_5 \leq 1$ corresponds to $p_5 > 1$ which results in the following (simplified) integer linear program:

$$1 - x_{t_1} \geq 0 \qquad \text{corresponds to place } p_1$$
$$x_{t_1} - x_{t_5} > 1 \qquad \text{corresponds to place } p_5$$

We can assign the variable x_{t_1} at most the value one; otherwise, the first equation is not satisfied. The second equation needs x_{t_1} to be at least two or greater to be satisfied. Therefore, the program is infeasible, and it is impossible to reach a marking where $p_5 > 1$ is satisfied. This implies that among the set of reachable markings from the initial marking the formula $p_5 \leq 1$ is universally true, and we have φ is equivalent to $E((true)U(p_4 = 2))$. This is equivalent to the reachability formula $EFp_4 = 2$, and we have simplified a CTL formula to a much easier to

verify reachability formula where static partial order reductions are also applicable. For more details, refer to paper **E** at the end of this thesis.

Contribution 11

We develop techniques for reducing the size of CTL formulae for Petri nets to improve the effectiveness of static partial order reductions, structural reductions, and model checking CTL properties.

Our formula simplification technique is implemented in the model checking tool TAPAAL [43, 69]. We extensively verify the effectiveness of formula simplification to reduce formula size and improving verification on the database of Petri net models and formulae from the 2017 edition of the MCC [79]. We reduce 22% of all CTL formulae to either *true* or *false*, respectively. Approximately half of all CTL formulae are reduced to simpler to verify reachability formulae. The size of formulae, defined as the number of nodes in the formula's parse tree, is reduced on average in half for CTL formulae.

Lastly, we compare our TAPAAL implementation against LoLA [118, 136] and the tool Sara [135] using integer linear programs to verify formulae without state-space exploration. In comparison to our formula simplification techniques, Sara is an exact technique for verifying formulae using CEGAR [35]. If an iteration is inconclusive, then the integer linear program is refined until the formulae can be determined to be satisfied or not. Sara is then executed in parallel with the verification engine of LoLA. We solve more formulae than Sara by reducing to either *true* or *false*. If we follow simplification with the verification engine of TAPAAL, then we also solve more formulae compared to both the verification engines of LoLA and Sara. For reachability formulae, Sara outperforms our formula simplification. However, followed by verification using TAPAAL, we marginally solve more formulae than LoLA and Sara.

Formula Simplification Future Work. Similarly to the future work of structural reductions in Section 2.4, future work includes how we can apply formula simplification to formalisms such as coloured Petri nets, Timed-Arc Petri nets, and Petri net games. Another possibility is to extend the technique to other temporal logics such as linear time logic. Furthermore, an interesting approach is to employ the exact reachability verification of Sara for formula simplification.

Contribution 12

We show the effectiveness of formula simplifications on the MCC'17 database of models and CTL and reachability formulae. The experiments show that we can simplify a significant amount of CTL formulae to either *true*, *false*, or a reachability formula. In comparison to LoLA+Sara, we solve more CTL formulae with and without state-space exploration. We solve more reachability formulae due to the benefit of simplification to static partial order reductions and structural reductions.

3 Conclusion

The work presented in this thesis improves the applicability of model checking by extending existing technique and developing new novel approaches. This includes extending static partial order reductions to time and game-theoretic formalisms, extending structural reductions for Petri nets and showing its benefit when combined with static partial order reductions, and developing simplification algorithms of CTL formulae.

We developed partial order reductions for timed systems and games by first determining a set of conditions for general labelled transition systems that preserves reachability and winning strategies for timed systems and games, respectively. We then instantiate these conditions to a concrete modelling formalism by efficiently overapproximating them, specifically Timed-Arc Petri nets, Petri net games, and Timed-Arc Petri net games. Notably, when combining the timed and game-theoretic setting, we notice that the inclusion allows us to refine the overapproximation algorithms to generate better static partial order reduction sets. Lastly, we showcase the practical viability of this approach on a set of case studies.

We explored orthogonal approaches like structural reductions, formula simplifications, and combinations thereof to complement static partial order reductions. For static partial order reductions and structural reductions, the two techniques synergistically improve each other, providing speedups and memory savings. Formula simplifications reduce the number of places in formulae for Petri nets, significantly improving both static partial order reductions and structural reductions. Combining these three techniques contributed to TAPAAL being compared

favourably to other competitive model checkers at the Model Checking Contest (MCC) [8].

Future work. There are several possible directions for future work besides the directions presented in Section 2. Dynamic partial order reductions [53, 2, 3, 9] are promising to employ either separately or complementary to static partial order reductions to solve subproblems. The application of partial order reductions to statistical model checking or systems with partial observability is also interesting. Lastly, it is relevant to explore more combinations of different formalisms, partial order reductions, and other state-space reduction techniques to improve verification effectiveness.

References

[1] P.A. Abdulla et al. "General Decidability Theorems for Infinite-State Systems". In: *Symposium on Logic in Computer Science*. LICS'96. IEEE, 1996, pp. 313–321. DOI: 10.1109/LICS.1996.561359.

[2] P.A. Abdulla et al. "Optimal Dynamic Partial Order Reduction". In: *Symposium on Principles of Programming Languages*. POPL '14. Association for Computing Machinery, 2014, pp. 373–384. DOI: 10.1145/2535838.2535845.

[3] P.A. Abdulla et al. "Source Sets: A Foundation for Optimal Dynamic Partial Order Reduction". In: *Journal of the ACM* 64.4 (2017). Association for Computing Machinery, pp. 1–49. DOI: 10.1145/3073408.

[4] R. Alur, C. Courcoubetis, and D. Dill. "Model-Checking for Probabilistic Real-Time Systems". In: *Automata, Languages and Programming*. Vol. 510. LNCS. Springer Berlin Heidelberg, 1991, pp. 115–126. DOI: 10.1007/3-540-54233-7_128.

[5] R. Alur and D. Dill. "The Theory of Timed Automata". In: *Real-Time: Theory in Practice*. Vol. 600. LNCS. Springer Berlin Heidelberg, 1992, pp. 45–73. DOI: 10.1007/BFb0031987.

[6] R. Alur, T.A. Henzinger, and M.Y. Vardi. "Parametric Real-Time Reasoning". In: *Symposium on Theory of Computing*. STOC'93. Association for Computing Machinery, 1993, pp. 592–601. DOI: 10.1145/167088.167242.

[7] Amazon. *Amazon DynamoDB: Developer Guide*. http://smtlib.cs.uiowa.edu/. 2012.

[8] E. Amparore et al. "Presentation of the 9th Edition of the Model Checking Contest". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 11429. LNCS. Springer International Publishing, 2019, pp. 50–68. DOI: 10.1007/978-3-030-17502-3_4.

[9] S. Aronis et al. "Optimal Dynamic Partial Order Reduction with Observers". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 10806. LNCS. Springer International Publishing, 2018, pp. 229–248. DOI: 10.1007/978-3-319-89963-3_14.

[10] C. Baier and J.P. Katoen. *Principles of Model Checking*. The MIT Press, 2009. ISBN: 026202649X, 9780262026499.

[11] C. Barrett and C. Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Model Checking*. Springer International Publishing, 2018. Chap. 11, pp. 305–343. DOI: 10.1007/978-3-319-10575-8_11.

[12] G. Behrmann et al. "Efficient Timed Reachability Analysis Using Clock Difference Diagrams". In: *Computer Aided Verification*. Vol. 1633. LNCS. Springer Berlin Heidelberg, 1999, pp. 341–353. DOI: 10.1007/3-540-48683-6_30.

[13] G. Behrmann et al. "UPPAAL Implementation Secrets". In: *Formal Techniques in Real-Time and Fault-Tolerant Systems*. Vol. 2469. LNCS. Springer Berlin Heidelberg, 2002, pp. 3–22. DOI: 10.1007/3-540-45739-9_1.

[14] J. Bengtsson. "Clocks, DBMs and States in Timed Systems". PhD thesis. Faculty of Science and Technology, Uppsala University, 2002.

[15] J. Bengtsson et al. "Partial Order Reductions for Timed Systems". In: *International Conference on Concurrency Theory*. Springer Berlin Heidelberg, 1998, pp. 485–500.

[16] A. Biere et al. "SAT-Based Model Checking". In: *Formal Models and Semantics*. Springer International Publishing, 2018. Chap. 10, pp. 277–303. DOI: 10.1007/978-3-319-10575-8_10.

[17] A. Biere et al. "Symbolic Model Checking without BDDs". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 1579. LNCS. Springer Berlin Heidelberg, 1999, pp. 193–2017. DOI: 10.1007/3-540-49059-0_14.

[18] R. Bloem, K. Chatterjee, and B. Jobstmann. "Graph Games and Reactive Synthesis". In: *Handbook of Model Checking*. Springer International Publishing, 2018. Chap. 17, pp. 921–962. DOI: 10.1007/978-3-319-10575-8_27.

[19] J. Bogdoll et al. "Partial Order Methods for Statistical Model Checking and Simulation". In: *Formal Techniques for Distributed Systems*. Vol. 6722. LNCS. Springer Berlin Heidelberg, 2011, pp. 59–74. DOI: 10.1007/978-3-642-21461-5_4.

[20] T. Bolognesi, F. Lucidi, and S. Trigila. "From Timed Petri Nets to Timed LOTOS". In: *Proceedings of the IFIP WG 6.1 Tenth International Symposium on Protocol Specification, Testing and Verification X*. North-Holland Publishing Co., 1990, pp. 395–408. DOI: 10.5555/645833.670383.

[21] F.M Bønneland et al. "Simplification of CTL Formulae for Efficient Model Checking of Petri Nets". In: *Application and Theory of Petri Nets and Concurrency*. Vol. 10877. LNCS. Springer International Publishing, 2018, pp. 176–185. DOI: 10.1007/978-3-319-91268-4_8.

[22] F.M Bønneland et al. "Stubborn Versus Structural Reductions for Petri nets". In: *Journal of Logical and Algebraic Methods in Programming* 102.1 (2019). Elsevier, pp. 46–63. DOI: 10.1016/j.jlamp.2018.09.002.

[23] F.M. Bønneland et al. "Partial Order Reduction for Reachability Games". In: *International Conference on Concurrency Theory*. Vol. 140. Leibniz International Proceedings in Informatics. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 23:1–23:15. DOI: 10.4230/LIPIcs.CONCUR.2019.23.

[24] F.M. Bønneland et al. "Start Pruning When Time Gets Urgent: Partial Order Reduction for Timed Systems". In: *Computer Aided Verification*. Vol. 10981. LNCS. Springer Berlin Heidelberg, 2018, pp. 527–546. DOI: 10.1007/978-3-319-96145-3_28.

[25] F.M. Bønneland et al. "Stubborn Set Reduction for Two-Player Reachability Games". In: *arXiv preprint arXiv:1912.09875* (2019).

[26] H. Boucheneb, K. Barkaoui, and K. Weslati. "Delay-Dependent Partial Order Reduction Technique for Time Petri Nets". In: *Formal Modeling and Analysis of Timed Systems*. Vol. 8711. LNCS. Springer International Publishing, 2014, pp. 53–68. DOI: 10.1007/978-3-319-10512-3_5.

[27] T. Brázdil et al. "Verification of Markov Decision Processes Using Learning Algorithms". In: *Automated Technology for Verification and Analysis*. Vol. 8837. LNCS. Springer International Publishing, 2014, pp. 98–114. DOI: 10.1007/978-3-319-11936-6_8.

[28] R.E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* C-35.8 (1986). IEEE, pp. 677–691. DOI: 10.1109/TC.1986.1676819.

[29] J.R. Burch et al. "Symbolic Model Checking: 10^{20} States and Beyond". In: *Information and Computation* C-35.8 (1992). Elsevier, pp. 142–170. DOI: 10.1016/0890-5401(92)90017-A.

[30] J.R. Burch et al. "Zero-Suppressed BDDs and Their Applications". In: *International Journal on Software Tools for Technology Transfer* 3.2 (2001). Springer, pp. 156–170. DOI: 10.1007/s100090100038.

[31] C. Calcagno et al. "Moving Fast with Software Verification". In: *NASA Formal Methods*. Vol. 9058. LNCS. Springer International Publishing, 2015, pp. 3–11. DOI: 10.1007/978-3-319-17524-9_1.

[32] J. Christ, J. Hoenicke, and A. Nutz. "SMTInterpol: An Interpolating SMT Solver". In: *Model Checking Software*. Vol. 7385. LNCS. Springer Berlin Heidelberg, 2012, pp. 248–254. DOI: 10.1007/978-3-642-31759-0_19.

[33] A. Cimatti et al. "NuSMV 2: An OpenSource Tool for Symbolic Model Checking". In: *Computer Aided Verification*. Vol. 2404. LNCS. Springer Berlin Heidelberg, 2002, pp. 359–364. DOI: 10.1007/3-540-45657-0_29.

[34] E.M. Clark et al. *Handbook of Model Checking*. Springer Cham, 2018. ISBN: N 978-3-319-10574-1,978-3-319-10575-8. DOI: 10.1007/978-3-319-10575-8.

[35] E. Clarke et al. "Counterexample-Guided Abstraction Refinement". In: *Computer Aided Verification*. Vol. 1855. LNCS. Springer Berlin Heidelberg, 2000, pp. 154–169. DOI: 10.1007/10722167_15.

[36] E.M. Clarke and E.A. Emerson. "Characterizing Correctness Properties of Parallel Programs Using Fixpoints". In: *Automata, Languages and Programming*. Vol. 85. LNCS. Springer Berlin Heidelberg, 1980, pp. 169–181. DOI: 10.1007/3-540-10003-2_69.

[37] E.M. Clarke and E.A. Emerson. "Design and Synthesis of Synchronization Skeletons Ssing Branching Time Temporal Logic". In: *Logics of Programs*. Vol. 131. LNCS. Springer Berlin Heidelberg, 1982, pp. 52–71. DOI: 10.1007/BFb0025774.

[38] E.M. Clarke, E.A. Emerson, and A.P. Sistia. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications". In: *ACM Trans. Program. Lang. Syst.* 8.2 (1986). Association for Computing Machinery, pp. 244–263. DOI: 10.1145/5397.5399.

[39] E.M. Clarke et al. "Exploiting Symmetry in Temporal Logic Model Checking". In: *Formal Methods in System Design* 9.1 (1996). Springer, pp. 77–104. DOI: 10.1007/BF00625969.

[40] H. Cohen et al. *BuDDy*. https://sourceforge.net/projects/buddy/.

[41] J.M. Couvreur. "On-the-fly Verification of Linear Temporal Logic". In: *Formal Methods*. Vol. 1708. LNCS. Springer Berlin Heidelberg, 1999, pp. 253–271. DOI: 10.1007/3-540-48119-2_16.

[42] D. Dams and O. Grumberg. "Abstraction and Abstraction Refinement". In: *Handbook of Model Checking*. Springer International Publishing, 2018. Chap. 13, pp. 385–419. DOI: 10.1007/978-3-319-10575-8_13.

[43] A. David et al. "TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 7214. LNCS. Springer Berlin Heidelberg, 2012, pp. 492–497. DOI: 10.1007/978-3-642-28756-5_36.

[44] A. David et al. "Uppaal SMC Tutorial". In: *International Journal on Software Tools for Technology Transfer* 17.4 (2015). Springer, pp. 397–415. DOI: 10.1007/s10009-014-0361-y.

[45] C. Daws et al. "The Tool KRONOS". In: *Hybrid Systems III*. Vol. 1066. LNCS. Springer Berlin Heidelberg, 1996, pp. 208–219. DOI: 10.1007/BFb0020947.

[46] E.W. Dijkstra. "The Humble Programmer". In: *Commun. ACM* 15.10 (1972). Association for Computing Machinery, pp. 859–866. DOI: 10.1145/355604.361591.

[47] D.L. Dill. "Timing Assumptions and Verification of Finite-State Concurrent Systems". In: *Automatic Verification Methods for Finite State Systems*. Vol. 407. LNCS. Springer Berlin Heidelberg, 1990, pp. 197–212. DOI: 10.1007/3-540-52148-8_17.

[48] E.A. Emerson. "Temporal and Modal Logic". In: *Formal Models and Semantics*. Elsevier, 1990. Chap. 16, pp. 995–1072. DOI: 10.1007/3-540-65306-6_20.

[49] E.A. Emerson, S. Jha, and D. Peled. "Combining Partial Order and Symmetry Reductions". In: *Transactions on Petri Nets and Other Models of Concurrency XI*. Vol. 1217. LNCS. Springer Berlin Heidelberg, 1997, pp. 19–34. DOI: 10.1007/BFb0035378.

[50] J. Esparza. "Decidability and Complexity of Petri Net Problems — An Introduction". In: *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*. Vol. 1491. LNCS. Springer Berlin Heidelberg, 1998, pp. 374–428. DOI: 10.1007/3-540-65306-6_20.

[51] P. Ezudheen et al. "Horn-ICE Learning for Synthesizing Invariants and Contracts". In: *Proc. ACM Program. Lang.* 2.OOPSLA (2016). Association for Computing Machinery, pp. 1–25. DOI: 10.1145/3276501.

[52] L. Feng et al. "Learning-Based Compositional Verification for Synchronous Probabilistic Systems". In: *Automated Technology for Verification and Analysis*. Vol. 6996. LNCS. Springer Berlin Heidelberg, 2011, pp. 511–521. DOI: 10.1007/978-3-642-24372-1_40.

[53] C. Flanagan and P. Godefroid. "Dynamic Partial-Order Reduction for Model Checking Software". In: *SIGPLAN Not.* 40.1 (2005). Association for Computing Machinery, pp. 110–121. DOI: 10.1145/1047659.1040315.

[54] P. Garg et al. "ICE: A Robust Framework for Learning Invariants". In: *Advances in Petri Nets 1990*. Vol. 8559. LNCS. Springer International Publishing, 2014, pp. 69–87. DOI: 10.1007/978-3-319-08867-9_5.

[55] P. Garg et al. "Learning Invariants Using Decision Trees and Implication Counterexamples". In: *SIGPLAN Not.* 51.1 (2016). Association for Computing Machinery, pp. 499–512. DOI: 10.1145/2914770.2837664.

[56] S.M. German and A.P. Sistla. "Reasoning about Systems with Many Processes". In: *Journal of the ACM* 39.3 (1992). Association for Computing Machinery, pp. 675–735. DOI: 10.1145/146637.146681.

[57] R. Gerth et al. "A Partial Order Approach to Branching Time Logic Model Checking". In: *Information and Computation* 150.2 (1999). Elsevier, pp. 132–152. DOI: 10.1006/inco.1998.2778.

[58] R. Gerth et al. "Simple On-the-fly Automatic Verification of Linear Temporal Logic". In: *Protocol Specification, Testing and Verification XV*. IFIPAICT. Springer Boston, 1995, pp. 3–18. DOI: 10.1007/978-0-387-34892-6_1.

[59] S. Ghemawat, H. Gobioff, and S.T. Leung. "The Google File System". In: *ACM SIGOPS Operating Systems Review* 37.5 (2003). Association for Computing Machinery, pp. 29–43.

[60] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*. Vol. 1032. LNCS. Springer Berlin Heidelberg, 1996. ISBN: 978-3-540-60761-8.

[61] P. Godefroid. "Refining Dependencies Improves Partial-Order Verification Methods (Extended Abstract)". In: *Computer Aided Verification*. Vol. 697. LNCS. Springer Berlin Heidelberg, 1993, pp. 438–449. DOI: 10.1007/3-540-56922-7_36.

[62] P. Godefroid. "Using Partial Orders to Improve Automatic Verification Methods". In: *Computer Aided Verification*. Vol. 531. LNCS. Springer Berlin Heidelberg, 1990, pp. 176–185. DOI: 10.1007/BFb0023731.

[63] P. Godefroid and P. Wolper. "Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties". In: *Formal Methods in System Design* 2.2 (1993). Springer, pp. 149–164. DOI: 10.1007/BF01383879.

[64] R. Grosu and S.A. Smolka. "Monte Carlo Model Checking". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 3440. LNCS. Springer Berlin Heidelberg, 2017, pp. 271–286. DOI: 10.1007/978-3-540-31980-1_18.

[65] H.M. Hanisch. "Analysis of Place/Transition Nets With Timed Arcs and Its Application to Batch Process Control". In: *Application and Theory of Petri Nets*. Vol. 691. LNCS. Springer Berlin Heidelberg, 1993, pp. 282–299. DOI: 10.1007/3-540-56863-8_52.

[66] H. Hansen et al. "Diamonds Are a Girl's Best Friend: Partial Order Reduction for Timed Automata with Abstractions". In: *Computer Aided Verification*. Vol. 8559. LNCS. Springer International Publishing, 2014, pp. 391–406. DOI: 10.1007/978-3-319-08867-9_26.

[67] T.A. Henzinger et al. "Abstractions From Proofs". In: *SIGPLAN Not.* 39.1 (2004). Association for Computing Machinery, pp. 232–244. DOI: 10.1145/982962.964021.

[68] D. Jackson and M. Vaziri. "Finding Bugs with a Constraint Solver". In: *SIGSOFT Softw. Eng. Notes* 25.5 (2000). Association for Computing Machinery, pp. 14–25. DOI: 10.1145/347636.383378.

[69] J.F. Jensen et al. "TAPAAL and Reachability Analysis of P/T Nets". In: *Transactions on Petri Nets and Other Models of Concurrency XI*. Vol. 9930. LNCS. Springer Berlin Heidelberg, 2016, pp. 307–318. DOI: 10.1007/978-3-662-53401-4_16.

[70] P.G. Jensen, K.G. Larsen, and J. Srba. "Discrete and Continuous Strategies for Timed-Arc Petri Net Games". In: *International Journal on Software Tools for Technology Transfer* 20.5 (2018). Springer, pp. 529–546. DOI: 10.1007/s10009-017-0473-2.

[71] P.G. Jensen, K.G. Larsen, and J. Srba. "PTrie: Data Structure for Compressing and Storing Sets via Prefix Sharing". In: *Theoretical Aspects of Computing*. Vol. 7214. 10580. Springer International Publishing, 2017, pp. 248–265. DOI: 10.1007/978-3-319-67729-3_15.

[72] P.G. Jensen, K.G. Larsen, and J. Srba. "Real-Time Strategy Synthesis for Timed-Arc Petri Net Games via Discretization". In: *Model Checking Software*. Vol. 9641. 10580. Springer International Publishing, 2016, pp. 129–146. DOI: 10.1007/978-3-319-32582-8_9.

[73] R. Jhala, A. Podelski, and A. Rybalchenko. "Predicate Abstraction for Program Verification". In: *Handbook of Model Checking*. Springer International Publishing, 2018. Chap. 15, pp. 447–491. DOI: 10.1007/978-3-319-10575-8_15.

[74] C.B. Jones. "Tentative Steps toward a Development Method for Interfering Programs". In: *ACM Trans. Program. Lang. Syst.* 5.4 (1983). Association for Computing Machinery, pp. 596–619. DOI: 10.1145/69575.69577.

[75] H. Kautz and B. Selman. "Planning as Satisfiability". In: *European Conference on Artificial Intelligence*. ECAI'92. IEEE, 1992, pp. 359–363. DOI: 10.5555/145448.146725.

[76] H. Kautz and B. Selman. "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search". In: *Conference on Artificial Intelligence*. Vol. 2. AAAI'96. AAAI Press, 1996, pp. 1194–1201. DOI: 10.5555/1864519.1864564.

[77] Y. Kesten et al. "Symbolic Model Checking With Rich Assertional Languages". In: *Theoretical Computer Science* 256.1-2 (2001). Elsevier, pp. 93–112. DOI: 10.1016/S0304-3975(00)00103-1.

[78] A. Komuravelli, C.S. Pasareanu, and E.M. Clarke. "Learning Probabilistic Systems from Tree Samples". In: *Symposium on Logic in Computer Science*. LICS'12. IEEE, 2012, pp. 511–521. DOI: 10.1109/LICS.2012.54.

[79] F. Kordon et al. "MCC'2017 – The Seventh Model Checking Contest". In: *Transactions on Petri Nets and Other Models of Concurrency XIII*. Vol. 11090. LNCS. Springer Berlin Heidelberg, 2018, pp. 181–209. DOI: 10.1007/978-3-662-58381-4_9.

[80] L.M. Kristensen, K. Schmidt, and A. Valmari. "Question-Guided Stubborn Set Methods for State Properties". In: *Formal Methods in System Design* 29.3 (2006). Springer, pp. 215–251. DOI: 10.1007/s10703-006-0006-1.

[81] L.M. Kristensen and A. Valmari. "Improved Question-Guided Stubborn Set Methods for State Properties". In: *Application and Theory of Petri Nets*. Vol. 1825. LNCS. Springer Berlin Heidelberg, 2000, pp. 282–302. DOI: 10.1007/3-540-44988-4_17.

[82] M. Kwiatkowska, G. Norman, and D. Parker. "Probabilistic Model Checking: Advances and Applications". In: *Formal System Verification: State-of the-Art and Future Trends*. Springer International Publishing, 2019, pp. 73–121. DOI: 10.1007/978-3-319-57685-5_3.

[83] M. Kwiatkowska et al. "2014 CAV Award Announcement". In: *Formal Methods in System Design* 48.3 (2016). Springer, pp. 149–152. DOI: 10.1007/s10703-016-0244-9.

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

[85] K.G. Larsen, D. Peled, and S. Sedwards. "Memory-Efficient Tactics for Randomized LTL Model Checking". In: *Verified Software. Theories, Tools, and Experiments*. Vol. 10712. LNCS. Springer International Publishing, 2017, pp. 152–169. DOI: 10.1007/978-3-319-72308-2_10.

[86] K.G. Larsen, P. Pettersson, and W. Yi. "Model-Checking for Real-Time Systems". In: *Fundamentals of Computation Theory*. Vol. 965. LNCS. Springer Berlin Heidelberg, 1995, pp. 62–88. DOI: 10.1007/3-540-60249-6_41.

[87] K.G. Larsen, P. Pettersson, and W. Yi. "Uppaal in a Nutshell". In: *International Journal on Software Tools for Technology Transfer* 1.1+2 (1997). Springer, pp. 134–152. DOI: 10.1007/s100090050010.

[88] K.G. Larsen et al. "Clock Difference Diagrams". In: *Nordic J. of Computing* 6.3 (1999). Publishing Association Nordic Journal of Computing, pp. 271–298. DOI: 10.5555/774455.774459.

[89] K.G. Larsen et al. "Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction". In: *Proceedings of Real-Time Systems Symposium*. Vol. 6605. LNCS. IEEE, 1997, pp. 14–24. DOI: 10.1109/REAL.1997.641265.

[90] A. Legay, B. Delahaye, and S. Bensalem. "Statistical Model Checking: An Overview". In: *Runtime Verification*. Vol. 6418. LNCS. Springer Berlin Heidelberg, 2010, pp. 122–135. DOI: 10.1007/978-3-642-16612-9_11.

[91] A. Lehmann, N. Lohmann, and K. Wolf. "Stubborn Sets for Simple Linear Time Properties". In: *Application and Theory of Petri Nets*. Vol. 7347. LNCS. Springer Berlin Heidelberg, 2012, pp. 228–247. DOI: 10.1007/978-3-642-31131-4_13.

[92] J. Lilius. "Efficient State Space Search for Time Petri Nets". In: *Electronic Notes in Theoretical Computer Science* 18.1 (1998). Elsevier, pp. 113–133. DOI: 10.1016/S1571-0661(05)80254-3.

[93] S.W. Lin et al. "Learning Assumptions for CompositionalVerification of Timed Systems". In: *IEEE Transactions on Software Engineering* 40.2 (2014). IEEE, pp. 137–153. DOI: 10.1109/TSE.2013.57.

[94] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag New York, 1992. ISBN: 978-0-387-97664-8,978-1-4612-6950-2. DOI: 10.1007/978-1-4612-0931-7.

[95] A. Mazurkiewicz. "Trace Theory". In: *Petri Nets: Applications and Relationships to Other Models of Concurrency*. Vol. 255. LNCS. Springer Berlin Heidelberg, 1987, pp. 278–324. DOI: 10.1007/3-540-17906-2_30.

[96] K.L. McMillan. "Interpolation and Model Checking". In: *Handbook of Model Checking*. Springer International Publishing, 2018. Chap. 14, pp. 447–491. DOI: 10.1007/978-3-319-10575-8_14.

[97] K.L. McMillan. "Interpolation and SAT-Based Model Checking". In: *Computer Aided Verification*. Vol. 2725. LNCS. Springer Berlin Heidelberg, 2003, pp. 1–13. DOI: 10.1007/978-3-540-45069-6_1.

[98] K.L. McMillan and N. Amla. "Automatic Abstraction without Counterexamples". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 2619. LNCS. Springer Berlin Heidelberg, 2003, pp. 2–17. DOI: 10.1007/3-540-36577-X_2.

[99] J. Misra. "Proofs of Networks of Processes". In: *IEEE Transactions on Software Engineering* SE-7.4 (1981). IEEE, pp. 417–426. DOI: 10.1109/TSE.1981.230844.

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

[101] T. Murata. "State Equation, Controllability, and Maximal Matchings of Petri Nets". In: *IEEE Transactions on Automatic Control* 22.3 (1977). IEEE, pp. 412–416. DOI: 10.1109/TAC.1977.1101509.

[102] T. Murata and J. Koh. "Reduction and Expansion of Live and Safe Marked Graphs". In: *IEEE Transactions on Circuits and Systems* 27.1 (1980). IEEE, pp. 68–71. DOI: 10.1109/TCS.1980.1084711.

[103] T. Neele, T.A.C. Willemse, and W. Wesselink. "Partial-Order Reduction for Parity Games with an Application on Parameterised Boolean Equation Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 12079. LNCS. Springer International Publishing, 2020, pp. 307–324. DOI: 10.1007/978-3-030-45237-7_19.

[104] C. Newcombe et al. "How Amazon Web Services Uses Formal Methods". In: *Commun. ACM* 58.4 (2015). Association for Computing Machinery, pp. 66–73. DOI: 10.1145/2699417.

[105] P.W. O'Hearn. "Continuous Reasoning: Scaling the Impact of Formal Methods". In: *Symposium on Logic in Computer Science*. LICS'18. Association for Computing Machinery, 2018, pp. 13–25. DOI: 10.1145/3209108.3209109.

[106] S. Owicki and D. Gries. "Verifying Properties of Parallel Programs: An Axiomatic Approach". In: *Commun. ACM* 19.1 (1969). Association for Computing Machinery, pp. 279–285. DOI: 10.1145/360051.360224.

[107] D. Peled. "All From One, One for All: On Model Checking Using Representatives". In: *Computer Aided Verification*. Vol. 697. LNCS. Springer Berlin Heidelberg, 1993, pp. 409–423. DOI: 10.1007/3-540-56922-7_34.

[108] D. Peled. "Combining Partial Order Reductions With On-The-Fly Model-Checking". In: *Formal Methods in System Design* 8.1 (1996). Springer, pp. 39–64. DOI: 10.1007/BF00121262.

[109] D. Peled. *Software Reliability Methods*. Springer-Verlag New York, 2001. ISBN: 978-1-4419-2876-4, 978-0-387-95106-5. DOI: 10.1007/978-1-4757-3540-6.

[110] D. Peled. "Ten Years of Partial Order Reduction". In: *Computer Aided Verification*. Vol. 1427. LNCS. Springer Berlin Heidelberg, 1998, pp. 17–28. DOI: 10.1007/BFb0028727.

[111] C.A Petri. "Kommunikation mit Automaten". In: *Bonn, Institut für Instrumentelle Mathematik* (1962).

[112] A. Pnueli. "In Transition From Global to Modular Temporal Reasoning about Programs". In: *Logics and Models of Concurrent Systems*. Vol. 13. NATO ASI. Springer Berlin Heidelberg, 1985, pp. 123–144. DOI: 10.1007/978-3-642-82453-1_5.

[113] J.P Queille and J. Sifakis. "Specification and Verification of Concurrent Systems in CESAR". In: *International Symposium on Programming*. Vol. 137. LNCS. Springer Berlin Heidelberg, 1982, pp. 337–351. DOI: 10.1007/BFb0025774.

[114] Y.S Ramakrishna and S.A. Smolka. "Partial-Order Reduction in the Weak Modal Mu-Calculus". In: *International Conference on Concurrency Theory*. Vol. 1243. LNCS. Springer Berlin Heidelberg, 1997, pp. 5–24. DOI: 10.1007/3-540-63141-0_2.

[115] W.P. Roever et al. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Vol. 54. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001. ISBN: 0-521-80608-9. DOI: 10.1007/978-3-319-10575-8.

[116] J. Rushby. "Formal Methods and Their Role in the Certification of Critical Systems". In: *Safety and Reliability of Software Based Systems*. Springer London, 1997, pp. 1–42. DOI: 10.1007/978-1-4471-0921-1_1.

[117] K. Schmidt. "Integrating Low Level Symmetries into Reachability Analysis". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 5404. LNCS. Springer Berlin Heidelberg, 2000, pp. 315–330. DOI: 10.1007/3-540-46419-0_22.

[118] K. Schmidt. "LoLA A Low Level Analyser". In: *Application and Theory of Petri Nets*. Vol. 1825. LNCS. Springer Berlin Heidelberg, 2000, pp. 465–474. DOI: 10.1007/3-540-44988-4_27.

[119] K. Schmidt. "Stubborn Sets for Standard Properties". In: *Application and Theory of Petri Nets*. Vol. 1639. LNCS. Springer Berlin Heidelberg, 1999, pp. 46–65. DOI: 10.1007/3-540-48745-X_4.

[120] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998. ISBN: 978-0-471-98232-6.

[121] S.F. Siegel. "Transparent Partial Order Reduction". In: *Formal Methods in System Design* 40.1 (2012). Springer, pp. 1–19. DOI: 10.1007/s10703-011-0126-0.

[122] R.H. Sloan and U. Buy. "Stubborn Sets for Real-Time Petri Nets". In: *Formal Methods in System Design* 11.1 (1997). Springer, pp. 23–40. DOI: 10.1023/A:1008629725384.

[123] F. Somenzi. *CUDD: CU Decision Diagram Package*. http://web.mit.edu/sage/export/tmp/y/usr/share/doc/polybori/cudd/cuddIntro.html.

[124] P. Stephan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. "Combinational Test Generation Using Satisfiability". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15.9 (1996). IEEE, pp. 1167–1176. DOI: 10.1109/43.536723.

[125] N.R. Storey. *Safety Critical Computer Systems*. Addison-Wesley, 1996. ISBN: 0201427877.

[126] Y. Thierry-Mieg. "Structural Reductions Revisited". In: *Application and Theory of Petri Nets and Concurrency*. Vol. 12152. LNCS. Springer International Publishing, 2020, pp. 303–323. DOI: 10.1007/978-3-030-51831-8_15.

[127] A. Valmari. "A Stubborn Attack on State Explosion". In: *Formal Methods in System Design* 1.4 (1992). Springer, pp. 297–322. DOI: 10.1007/BF00709154.

[128] A. Valmari. "On-The-Fly Verification With Stubborn Sets". In: *Computer Aided Verification*. Vol. 697. LNCS. Springer Berlin Heidelberg, 1993, pp. 397–408. DOI: 10.1007/3-540-56922-7_33.

[129] A. Valmari. "Stubborn Set Intuition Explained". In: *Transactions on Petri Nets and Other Models of Concurrency XII*. Vol. 10470. LNCS. Springer Berlin Heidelberg, 2017, pp. 140–165. DOI: 10.1007/978-3-662-55862-1_7.

[130] A. Valmari. "Stubborn Sets for Reduced State Space Generation". In: *Advances in Petri Nets 1990*. Vol. 483. LNCS. Springer Berlin Heidelberg, 1991, pp. 491–515. DOI: 10.1007/3-540-53863-1_36.

[131] I. Virbitskaite and E. Pokozy. "A Partial Order Method for the Verification of Time Petri Nets". In: *Fundamentals of Computation Theory*. Vol. 1684. LNCS. Springer Berlin Heidelberg, 1999, pp. 547–558. DOI: 10.1007/3-540-48321-7_46.

[132] Y. Vizel, G. Weissenbacher, and S. Malik. "Boolean Satisfiability Solvers and Their Applications in Model Checking". In: *Proceedings of the IEEE* 103.11 (2015). IEEE, pp. 2021–2035. DOI: 10.1109/JPROC.2015.2455034.

[133] M. Wan and G. Ciardo. "Symbolic Reachability Analysis of Integer Timed Petri Nets". In: *SOFSEM 2009: Theory and Practice of Computer Science*. Vol. 1785. LNCS. Springer Berlin Heidelberg, 2000, pp. 595–608. DOI: 10.1007/978-3-540-95891-8_53.

[134] B. Willems and P. Wolper. "Partial-Order Methods for Model Checking: From Linear Time to Branching Time". In: *Symposium on Logic in Computer Science*. LICS'96. IEEE, 1996, pp. 294–303. DOI: 10.1109/LICS.1996.561357.

[135] H. Wimmel and K. Wolf. "Applying CEGAR to the Petri Net State Equation". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 6605. LNCS. Springer Berlin Heidelberg, 2011, pp. 224–238. DOI: 10.1007/978-3-642-19835-9_19.

[136] K. Wolf. "Petri Net Model Checking With Lola 2". In: *Application and Theory of Petri Nets*. Vol. 10877. LNCS. Springer International Publishing, 2018, pp. 351–362. DOI: 10.1007/978-3-319-91268-4_18.

Part II

Papers

Paper A

Start Pruning When Time Gets Urgent: Partial Order Reduction for Timed Systems

Frederik M. Bønneland, Peter G. Jensen,
Kim G. Larsen, Marco Muñiz, and Jiří Srba

This paper has been published in:
Computer Aided Verification, LNCS Vol. 10981,
pp. 527-546, 2018.

Abstract

Partial order reduction for timed systems is a challenging topic due to the dependencies among events induced by time acting as a global synchronization mechanism. So far, there has only been a limited success in finding practically applicable solutions yielding significant state space reductions. We suggest a working and efficient method to facilitate stubborn set reduction for timed systems with urgent behaviour. We first describe the framework in the general setting of timed labelled transition systems and then instantiate it to the case of timed-arc Petri nets. The basic idea is that we can employ classical untimed partial order reduction techniques as long as urgent behaviour is enforced. Our solution is implemented in the model checker TAPAAL and the feature is now broadly available to the users of the tool in its latest release from January 2018. By a series of larger case studies, we document the benefits of our method and its applicability to real-world scenarios.

1 Introduction

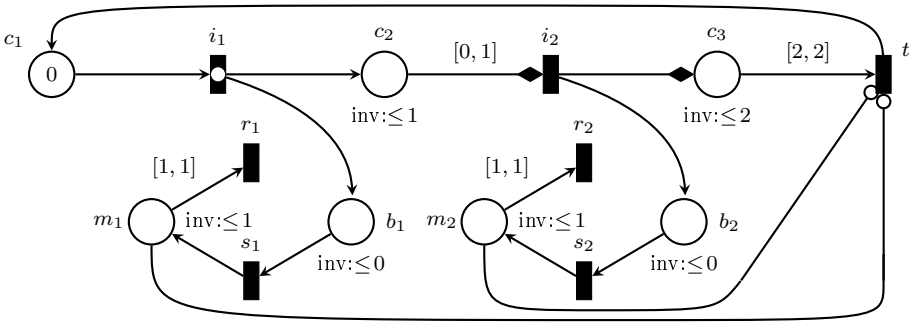
Partial order reduction techniques for untimed systems, introduced by Godefroid, Peled, and Valmari in the nineties (see e.g. [6]), have since long proved successful in combating the notorious state space explosion problem. For *timed* systems, the success of partial order reduction has been significantly challenged by the strong dependencies between events caused by time as a global synchronizer. Only recently—and moreover in combination with *approximate* abstraction techniques—stubborn set techniques have demonstrated a true reduction potential for systems modelled by timed automata [22].

We pursue an orthogonal solution to the current partial order approaches for timed systems and, based on a stubborn set reduction [26, 38], we target a general class of timed systems with *urgent behaviour*. In a modular modelling approach for timed systems, urgency is needed to realistically model behaviour in a component that should be unobservable to other components [35]. Examples of such instantaneously evolving behaviours include, among others, cases like behaviour detection in a part of a sensor (whose duration is assumed to be negligible) or handling of release and completion of periodic tasks in a real-time operating system. We observe that focusing on the urgent part of the behaviour of a timed system allows us to exploit the full range of partial

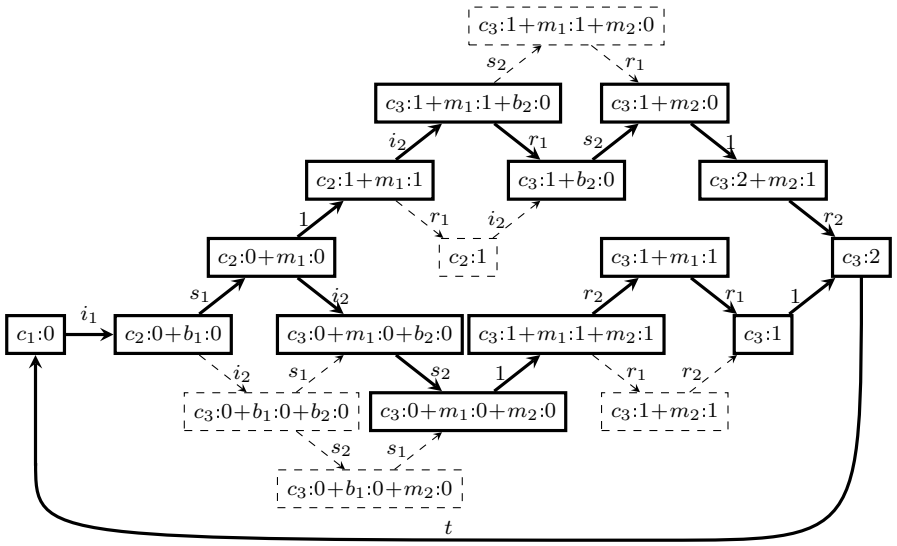
order reduction techniques already validated for untimed systems. This leads to an exact and broadly applicable reduction technique, which we shall demonstrate on a series of industrial case studies showing significant space and time reduction. In order to highlight the generality of the approach, we first describe our reduction technique in the setting of timed labelled transition systems. We shall then instantiate it to timed-arc Petri nets and implement and experimentally validate it in the model checker TAPAAL [18].

Let us now briefly introduce the model of timed-arc Peri nets and explain our reduction ideas. In timed-arc Petri nets, each token is associated with a nonnegative integer representing its age and input arcs to transitions contain intervals, restricting the ages of tokens available for transition firing (if an interval is missing, we assume the default interval $[0, \infty]$ that accepts all token ages). In Figure 1a we present a simple monitoring system modelled as a timed-arc Petri net. The system consists of two identical sensors where sensor i, $i \in \{1, 2\}$, is represented by the places b_i and m_i, and the transitions s_i and r_i. Once a token of age 0 is placed into the place b_i, the sensor gets started by executing the transition s_i and moving the token from place b_i to m_i where the monitoring process starts. As the place b_i has an associated age invariant ≤ 0, meaning that all tokens in b_i must be of age at most 0, no time delay is allowed and the firing of s_i becomes urgent. In the monitoring place m_i we have to delay one time unit before the transition r_i reporting the reading of the sensor becomes enabled. Due to the age invariant ≤ 1 in the place m_i, we cannot wait longer than one time unit, after which r_i becomes also urgent.

The places c_1, c_2 and c_3 together with the transitions i_1, i_2 and t are used to control the initialization of the sensors. At the execution start, only the transition i_1 is enabled and because it is an urgent transition (denoted by the white circle), no delay is initially possible and i_1 must be fired immediately while removing the token of age 0 from c_1 and placing a new token of age 0 into c_2. At the same time, the first sensor gets started as i_1 also places a fresh token of age 0 into b_1. Now the control part of the net can decide to fire without any delay the transition i_2 and start the second sensor, or it can delay one unit of time after which i_2 becomes urgent due to the age invariant ≤ 1 as the token in c_2 is now of age 1. If i_2 is fired now, it will place a fresh token of age 0 into b_2. However, the token that is moved from c_2 to c_3 by the pair of transport arcs with the diamond-shaped arrow tips preserves its age 1, so now we have to



(a) TAPN model of a simple monitoring system



(b) Reachable state space generated by the net in Figure 1a

Figure 1: Simple Monitoring System

wait precisely one more time unit before t becomes enabled. Moreover, before t can be fired, the places m_1 and m_2 must be empty as otherwise the firing of t is disabled due to inhibitor arcs with circle-shaped arrow tips.

In Figure 1b we represent the reachable state space of the simple monitoring system where markings are represented using the notation like $c_3 : 1 + b_2 : 2$ that stands for one token of age 1 in place c_3 and one token of age 2 in place b_2. The dashed boxes represent the markings that can be avoided during the state space exploration when we apply our partial order reduction method for checking if the termination transition

t can become enabled from the initial marking. We can see that the partial order reduction is applied such that it preserves at least one path to all configurations where our goal is reached (transition t is enabled) and where time is not urgent anymore (i.e. to the configurations that allow the delay of 1 time unit). The basic idea of our approach is to apply the stubborn set reduction on the commutative diamonds where time is not allowed to elapse.

Related Work. Our stubborn set reduction is based on the work of Valmari et al. [26, 38]. We formulate their stubborn set method in the abstract framework of labelled transition systems with time and add further axioms for time elapsing in order to guarantee preservation of the reachability properties.

For Petri nets, Yoneda and Schlingloff [41] apply a partial order reduction to one-safe time Petri nets, however, as claimed in [37], the method is mainly suitable for small to medium models due to a computational overhead, confirmed also in [27]. The experimental evaluation in [41] shows only one selected example. Sloan and Buy [37] try to improve on the efficiency of the method, at the expense of considering only a rather limited model of *simple time Petri nets* where each transition has a statically assigned duration. Lilius [27] suggests to instead use alternative semantics of timed Petri nets to remove the issues related to the global nature of time, allowing him to apply directly the untimed partial order approaches. However, the semantics is nonstandard and no experiments are reported. Another approach is by Virbitskaite and Pokozy [40], who apply a partial order method on the *region graph* of bounded time Petri nets. Region graphs are in general not an efficient method for state space representation and the method is demonstrated only on a small buffer example with no further experimental validation. Recently, partial order techniques were suggested by André, Chatain and Rodríguez for parametric time Petri nets [5], however, the approach is working only for safe and acyclic nets. Boucheneb and Barkaoui [13, 12, 11] discuss a partial order reduction technique for timed Petri nets based on *contracted state class graphs* and present a few examples on a prototype implementation (the authors do not refer to any publicly available tool). Their method is different from ours as it aims at adding timing constrains to the independence relation, but it does not exploit urgent behaviour. Moreover, the models of time Petri nets and timed-arc Petri nets are, even on the simplest nets, incomparable due to the different

way to modelling time.

The fact that we are still lacking a practically applicable method for the time Petri net model is documented by a missing implementation of the technique in leading tools for time Petri net model checking like TINA [39] and Romeo [21]. We are not aware of any work on partial order reduction technique for the class of timed-arc Petri nets that we consider in this paper. This is likely because this class of nets provides even more complex timing behaviour, as we consider unbounded nets where each token carries its timing information (and needs a separate clock to remember the timing), while in time Petri nets timing is associated only to a priory fixed number of transitions in the net.

In the setting of timed automata [3], early work on partial order reduction includes Bengtsson et al. [8] and Minea [31] where they introduce the notion of local as well as global clocks but provide no experimental evaluation. Dams et al. [17] introduce the notion of *covering* in order to generalize dependencies but also here no empirical evaluation is provided. Lugiez, Niebert et al. [28, 33] study the notion of *event zones* (capturing time-durations between events) and use it to implement Mazurkiewicz-trace reductions. Salah, Bozga and Maler [36] introduce and implement an exact method based on merging zones resulting from different interleavings. The method achieves performance comparable with the approximate convex-hull abstraction which is by now superseded by the exact LU-abstraction [7]. Most recently, Hansen et al. [22] introduce a variant of stubborn sets for reducing an *abstracted zone graph*, thus in general offering overapproximate analysis. Our technique is orthogonal to the other approaches mentioned above; not only is the model different but also the application of our reduction gives exact results and is based on new reduction ideas. Finally, the idea of applying partial order reduction for independent events that happen at the same time appeared also in [14] where the authors, however, use a static method that declares actions as independent only if they do not communicate, do not emit signals and do not access any shared variables. Our realization of the method to the case of timed-arc Petri nets applies a dynamic (on-the-fly) reduction, while executing a detailed timing analysis that allows us to declare more transitions as independent—sometimes even in the case when they share resources.

2 Partial Order Reduction for Timed Systems

We shall now describe the general idea of our partial order reduction technique (based on stubborn sets [26, 38]) in terms of timed transition systems. We consider real-time delays in the rest of this section, as these results are not specific only to discrete time semantics. Let A be a given set of actions such that $A \cap \mathbb{R}_{\geq 0} = \emptyset$ where $\mathbb{R}_{\geq 0}$ stands for the set of nonnegative real numbers.

Definition 1 (Timed Transition System)**.** *A* timed transition system *is a tuple* $(S, s_0, \rightarrow)$ *where* S *is a set of states,* $s_0 \in S$ *is the initial state, and* $\rightarrow \subseteq S \times (A \cup \mathbb{R}_{\geq 0}) \times S$ *is the transition relation.*

If $(s, \alpha, s') \in \rightarrow$ we write $s \xrightarrow{\alpha} s'$. We implicitly assume that if $s \xrightarrow{0} s'$ then $s = s'$, i.e. zero time delays do not change the current state. The set of *enabled actions* at a state $s \in S$ is defined as $\mathsf{En}(s) \stackrel{\mathsf{def}}{=} \{a \in A \mid \exists s' \in S.\ s \xrightarrow{a} s'\}$. Given a sequence of actions $w = \alpha_1\alpha_2\alpha_3 \ldots \alpha_n \in (A \cup \mathbb{R}_{\geq 0})^*$ we write $s \xrightarrow{w} s'$ iff $s \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s'$. If there is a sequence w of length n such that $s \xrightarrow{w} s'$, we also write $s \rightarrow^n s'$. Finally, let $\rightarrow^*$ be the reflexive and transitive closure of the relation $\rightarrow$ such that $s \rightarrow s'$ iff there is $\alpha \in \mathbb{R}_{\geq 0} \cup A$ and $s \xrightarrow{\alpha} s'$.

For the rest of this section, we assume a fixed transition system $(S, s_0, \rightarrow)$ and a set of goal states $G \subseteq S$. The *reachability problem*, given a timed transition system $(S, s_0, \rightarrow)$ and a set of goal states G, is to decide whether there is $s' \in G$ such that $s_0 \rightarrow^* s'$.

We now develop the theoretical foundations of stubborn sets for timed transition systems. A state $s \in S$ is *zero time* if time can not elapse at s. We denote the zero time property of a state s by the predicate $\mathsf{zt}(s)$ and define it as $\mathsf{zt}(s)$ iff for all $s' \in S$ and all $d \in \mathbb{R}_{\geq 0}$ if $s \xrightarrow{d} s'$ then $d = 0$. A *reduction* of a timed transition system is a function $\mathsf{St} : S \rightarrow 2^A$. A reduction defines a reduced transition relation $\underset{\mathsf{St}}{\longrightarrow} \subseteq \rightarrow$ such that $s \xrightarrow[\mathsf{St}]{\alpha} s'$ iff $s \xrightarrow{\alpha} s'$ and $\alpha \in \mathsf{St}(s) \cup \mathbb{R}_{\geq 0}$. For a given state $s \in S$ we define $\overline{\mathsf{St}(s)} \stackrel{\mathsf{def}}{=} A \setminus \mathsf{St}(s)$ as the set of all actions that are not in $\mathsf{St}(s)$.

Definition 2 (Reachability Conditions)**.** *A reduction* St *on a timed transition system* $(S, s_0, \rightarrow)$ *is* reachability preserving *if it satisfies the following four conditions.*

$(\mathcal{Z})$ $\quad \forall s \in S.\ \neg\mathsf{zt}(s) \implies \mathsf{En}(s) \subseteq \mathsf{St}(s)$

$(\mathcal{D})$ $\quad \forall s, s' \in S.\ \forall w \in \overline{\mathsf{St}(s)}^*.\ \mathsf{zt}(s) \wedge s \xrightarrow{w} s' \implies \mathsf{zt}(s')$

$(\mathcal{R})$ $\quad \forall s, s' \in S.\ \forall w \in \overline{\mathsf{St}(s)}^*.\ \mathsf{zt}(s) \wedge s \xrightarrow{w} s' \wedge s \notin G \implies s' \notin G$

$(\mathcal{W})$ $\quad \forall s, s' \in S.\ \forall w \in \overline{\mathsf{St}(s)}^*.\ \forall a \in \mathsf{St}(s).\ \mathsf{zt}(s) \wedge s \xrightarrow{wa} s' \implies s \xrightarrow{aw} s'$

Condition $\mathcal{Z}$ declares that in a state where a delay is possible, all enabled actions become stubborn actions. Condition $\mathcal{D}$ guarantees that in order to enable a time delay from a state where delaying is not allowed, a stubborn action must be executed, and similarly Condition $\mathcal{R}$ requires the same should a goal state be reachable from a non-goal state. Finally, Condition $\mathcal{W}$ allows us to commute stubborn actions with non-stubborn actions. The following theorem shows that reachability preserving reductions generate pruned transition systems where the reachability of goal states is preserved.

Theorem 1 (Minimum-Time Reachability Preservation)**.** *Let* St *be a reachability preserving reduction satisfying* $\mathcal{Z}$*,* $\mathcal{D}$*,* $\mathcal{R}$ *and* $\mathcal{W}$*. Let* $s \in S$*. If* $s \to^n s'$ *for some* $s' \in G$ *then also* $s \xrightarrow[\mathsf{St}]{}^m s''$ *for some* $s'' \in G$ *where* $m \leq n$*.*

Proof. We proceed by induction on n. *Base step.* If $n = 0$, then $s = s'$ and $m = n = 0$. *Inductive step.* Let $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s_{n+1}$ where $s_0 \notin G$ and $s_{n+1} \in G$. Without loss of generality we assume that for all i, $0 \leq i \leq n$, we have $\alpha_i \neq 0$ (otherwise we can simply skip these 0-delay actions and get a shorter sequence). We have two cases. Case $\neg\mathsf{zt}(s_0)$: by condition $\mathcal{Z}$ we have $\mathsf{En}(s_0) \subseteq \mathsf{St}(s_0)$ and by the definition of $\xrightarrow[\mathsf{St}]{}$ we have $s_0 \xrightarrow[\mathsf{St}]{\alpha_0} s_1$ since $\alpha_0 \in \mathsf{En}(s_0) \cup \mathbb{R}_{\geq 0}$. By the induction hypothesis we have $s_1 \xrightarrow[\mathsf{St}]{}^m s''$ with $s'' \in G$ and $m \leq n$ and $m + 1 \leq n + 1$. Case $\mathsf{zt}(s_0)$: let $w = \alpha_0\alpha_1 \ldots \alpha_n$ and α_i be such that $\alpha_i \in \mathsf{St}(s_0)$ and for all $k < i$ holds that $\alpha_k \notin \mathsf{St}(s_0)$, i.e. α_i is the first stubborn action in w. Such an α_i has to exist otherwise $s_{n+1} \notin G$ due to condition $\mathcal{R}$. Because of condition $\mathcal{D}$ we get $\mathsf{zt}(s_k)$ for all k, $0 \leq k < i$, otherwise α_i cannot be the first stubborn action in w. We can split w as $w = u\alpha_i v$ with $u \in \overline{\mathsf{St}(s_0)}^*$. Since all states in the path to s_i are zero time, by $\mathcal{W}$ we can swap α_i as $s_0 \xrightarrow{\alpha_i} s'_1 \xrightarrow{u} s_i \xrightarrow{v} s'$ with $|uv| = n$. Since $\alpha_i \in \mathsf{St}(s_0)$ we get $s_0 \xrightarrow[\mathsf{St}]{\alpha_i} s'_1$ and by the induction hypothesis we have $s'_1 \xrightarrow[\mathsf{St}]{}^m s''$ where $s'' \in G$, $m \leq n$, and $m + 1 \leq n + 1$. □

3 Timed-Arc Petri Nets

We shall now define the model of timed-arc Petri nets (as informally described in the introduction) together with a reachability logic and a few technical lemmas needed later on. Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ and $\mathbb{N}_0^\infty = \mathbb{N}_0 \cup \{\infty\}$. We define the set of *well-formed closed time intervals* as $\int \stackrel{\text{def}}{=} \{[a,b] \mid a \in \mathbb{N}_0, b \in \mathbb{N}_0^\infty, a \leq b\}$ and its subset $\int^{\text{inv}} \stackrel{\text{def}}{=} \{[0,b] \mid b \in \mathbb{N}_0^\infty\}$ used in age invariants.

Definition 3 (Timed-Arc Petri Net)**.** *A* timed-arc Petri net *(TAPN) is a 9-tuple* $N = (P, T_1, T_2, T_{urg}, IA, OA, g, w, Type, I)$ *where*

- P *is a finite set of* places,
- T *is a finite set of* transitions *such that* $P \cap T = \emptyset$,
- $T_{urg} \subseteq T$ *is the set of* urgent transitions,
- $IA \subseteq P \times T$ *is a finite set of* input arcs,
- $OA \subseteq T \times P$ *is a finite set of* output arcs,
- $g : IA \to \int$ *is a* time constraint function *assigning guards (time intervals) to input arcs s.t.*
  - *if* $(p,t) \in IA$ *and* $t \in T_{urg}$ *then* $g((p,t)) = [0, \infty]$,
- $w : IA \cup OA \to \mathbb{N}$ *is a function assigning* weights *to input and output arcs,*
- $Type : IA \cup OA \to \mathbf{Types}$ *is a* type function *assigning a type to all arcs where* $\mathbf{Types} = \{Normal, I\} \cup \{Transport_j \mid j \in \mathbb{N}\}$ *such that*
  - *if* $Type(z) = I$ *then* $z \in IA$ *and* $g(z) = [0, \infty]$,
  - *if* $Type((p,t)) = Transport_j$ *for some* $(p,t) \in IA$ *then there is exactly one* $(t,p') \in OA$ *such that* $Type((t,p')) = Transport_j$,
  - *if* $Type((t,p')) = Transport_j$ *for some* $(t,p') \in OA$ *then there is exactly one* $(p,t) \in IA$ *such that* $Type((p,t)) = Transport_j$,
  - *if* $Type((p,t)) = Transport_j = Type((t,p'))$ *then* $w((p,t)) = w((t,p'))$,
- $I : P \to \int^{inv}$ *is a function assigning* age invariants *to places.*

Note that for transport arcs we assume that they come in pairs (for each type $Transport_j$) and that their weights match. Also for inhibitor arcs and for input arcs to urgent transitions, we require that the guards are $[0, \infty]$.

Before we give the formal semantics of the model, let us fix some notation. Let $N = (P, T_1, T_2, T_{urg}, IA, OA, g, w, Type, I)$ be a TAPN. We denote by ${}^\bullet x \stackrel{\text{def}}{=} \{y \in P \cup T \mid (y, x) \in IA \cup OA,\ Type((y, x)) \neq I\}$ the preset of a transition or a place x. Similarly, the postset is defined as $x^\bullet \stackrel{\text{def}}{=} \{y \in P \cup T \mid (x, y) \in (IA \cup OA)\}$. We denote by ${}^\circ t \stackrel{\text{def}}{=} \{p \in P \mid (p, t) \in IA \wedge Type((p, t)) = I\}$ the inhibitor preset of a transition t. The inhibitor postset of a place p is defined as $p^\circ \stackrel{\text{def}}{=} \{t \in T \mid (p, t) \in IA \wedge Type((p, t)) = I\}$. Let $\mathcal{B}(\mathbb{R}^{\geq 0})$ be the set of all finite multisets over $\mathbb{R}^{\geq 0}$. A *marking* M on N is a function $M : P \longrightarrow \mathcal{B}(\mathbb{R}^{\geq 0})$ where for every place $p \in P$ and every token $x \in M(p)$ we have $x \in I(p)$, in other words all tokens have to satisfy the age invariants. The set of all markings in a net N is denoted by $\mathcal{M}(N)$.

We write (p, x) to denote a token at a place p with the age $x \in \mathbb{R}^{\geq 0}$. Then $M = \{(p_1, x_1), (p_2, x_2), \ldots, (p_n, x_n)\}$ is a multiset representing a marking M with n tokens of ages x_i in places p_i. We define the size of a marking as $|M| = \sum_{p \in P} |M(p)|$ where $|M(p)|$ is the number of tokens located in the place p. A marked TAPN (N, M_0) is a TAPN N together with an initial marking M_0 with all tokens of age 0.

Definition 4 (Enabledness). *Let $N = (P, T_1, T_2, T_{urg}, IA, OA, g, w,$ $Type, I)$ be a TAPN. We say that a transition $t \in T$ is* enabled *in a marking M by the multisets of tokens $In = \{(p, x_p^1), (p, x_p^2), \ldots, (p, x_p^{w((p,t))}) \mid p \in {}^\bullet t\} \subseteq M$ and $Out = \{(p', x_{p'}^1), (p', x_{p'}^2), \ldots, (p', x_{p'}^{w((t,p'))}) \mid p' \in t^\bullet\}$ if*

- *for all input arcs except the inhibitor arcs, the tokens from In satisfy the age guards of the arcs, i.e.*

$$\forall p \in {}^\bullet t.\ x_p^i \in g((p, t)) \text{ for } 1 \leq i \leq w((p, t))$$

- *for any inhibitor arc pointing from a place p to the transition t, the number of tokens in p is smaller than the weight of the arc, i.e.*

$$\forall (p, t) \in IA. Type((p, t)) = I \Rightarrow |M(p)| < w((p, t))$$

- *for all input arcs and output arcs which constitute a transport arc, the age of the input token must be equal to the age of the output*

token and satisfy the invariant of the output place, i.e.

$$\forall (p,t) \in IA. \forall (t,p') \in OA. Type((p,t)) = Type((t,p')) = Transport_j \\ \Rightarrow \left(x_p^i = x_{p'}^i \wedge x_{p'}^i \in I(p')\right) \text{ for } 1 \leq i \leq w((p,t))$$

- *for all normal output arcs, the age of the output token is* 0, *i.e.*

$$\forall (t,p') \in OA. Type((t,p')) = Normal \Rightarrow x_{p'}^i = 0 \text{ for } 1 \leq i \leq w((t,p'))$$

A given marked TAPN (N, M_0) defines a timed transition system $T(N) \stackrel{\text{def}}{=} (\mathcal{M}(N), M_0, \rightarrow)$ where the states are markings and the transitions are as follows.

- If $t \in T$ is enabled in a marking M by the multisets of tokens *In* and *Out* then t can *fire* and produce the marking $M' = (M \setminus In) \uplus Out$ where $\uplus$ is the multiset sum operator and $\setminus$ is the multiset difference operator; we write $M \xrightarrow{t} M'$ for this action transition.
- A time *delay* $d \in \mathbb{N}_0$ is allowed in M if
  - $(x+d) \in I(p)$ for all $p \in P$ and all $x \in M(p)$, i.e. by delaying d time units no token violates any of the age invariants, and
  - if $M \xrightarrow{t} M'$ for some $t \in T_{urg}$ then $d = 0$, i.e. enabled urgent transitions disallow time passing.

  By delaying d time units in M we reach the marking M' defined as $M'(p) = \{x + d \mid x \in M(p)\}$ for all $p \in P$; we write $M \xrightarrow{d} M'$ for this delay transition.

Note that the semantics above defines the discrete-time semantics as the delays are restricted to nonnegative integers. It is well known that for timed-arc Petri nets with nonstrict intervals, the marking reachability problem on discrete and continuous time nets coincide [30]. This is, however, not the case for more complex properties like liveness that can be expressed in the CTL logic (for counter examples that can be expressed in CTL see e.g. [24]).

Reachability Logic and Interesting Sets of Transitions

We now describe a logic for expressing the properties of markings based on the number of tokens in places and transition enabledness, inspired by

Formula φ	$A_M(\varphi)$	$A_M(\neg\varphi)$
deadlock	$(^\bullet t)^\bullet \cup {}^\bullet(^\circ t)$ for some $t \in \mathsf{En}(M)$	$\emptyset$
t	${}^\bullet p$ for some $p \in {}^\bullet t$ where $M(p) < w((p,t))$ or $p^\bullet$ for some $p \in {}^\circ t$ where $M(p) \geq w((p,t))$	$(^\bullet t)^\bullet \cup {}^\bullet(^\circ t)$
$e_1 < e_2$	$decr_M(e_1) \cup incr_M(e_2)$	$A_M(e_1 \geq e_2)$
$e_1 \leq e_2$	$decr_M(e_1) \cup incr_M(e_2)$	$A_M(e_1 > e_2)$
$e_1 > e_2$	$incr_M(e_1) \cup decr_M(e_2)$	$A_M(e_1 \leq e_2)$
$e_1 \geq e_2$	$incr_M(e_1) \cup decr_M(e_2)$	$A_M(e_1 < e_2)$
$e_1 = e_2$	$decr_M(e_1) \cup incr_M(e_2)$ if $eval_M(e_1) > eval_M(e_2)$ $incr_M(e_1) \cup decr_M(e_2)$ if $eval_M(e_1) < eval_M(e_2)$	$A_M(e_1 \neq e_2)$
$e_1 \neq e_2$	$incr_M(e_1) \cup decr_M(e_1) \cup incr_M(e_2) \cup decr_M(e_2)$	$A_M(e_1 = e_2)$
$\varphi_1 \wedge \varphi_2$	$A_M(\varphi_i)$ for some $i \in \{1,2\}$ where $M \not\models \varphi_i$	$A_M(\neg\varphi_1 \vee \neg\varphi_2)$
$\varphi_1 \vee \varphi_2$	$A_M(\varphi_1) \cup A_M(\varphi_2)$	$A_M(\neg\varphi_1 \wedge \neg\varphi_2)$

Table 1: Interesting transitions of φ (assuming $M \not\models \varphi$, otherwise $A_M(\varphi) = \emptyset$)

the logic used in the Model Checking Contest (MCC) Property Language [25]. Let $N = (P, T_1, T_2, T_{urg}, IA, OA, g, w, Type, I)$ be a TAPN. The formulae of the logic are given by the abstract syntax:

$$\varphi ::= \mathit{deadlock} \mid t \mid e_1 \bowtie e_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi$$

$$e ::= c \mid p \mid e_1 \oplus e_2$$

where $t \in T$, $\bowtie \in \{<, \leq, =, \neq, >, \geq\}$, $c \in \mathbb{Z}$, $p \in P$, and $\oplus \in \{+, -, *\}$. Let Φ be the set of all such formulae and let E_N be the set of arithmetic expressions over the net N. The semantics of φ in a marking $M \in \mathcal{M}(N)$ is given by

$$\begin{aligned} M &\models \mathit{deadlock} && \text{if } \mathsf{En}(M) = \emptyset \\ M &\models t && \text{if } t \in \mathsf{En}(M) \\ M &\models e_1 \bowtie e_2 && \text{if } eval_M(e_1) \bowtie eval_M(e_2) \end{aligned}$$

assuming a standard semantics for Boolean operators and where the semantics of arithmetic expressions in a marking M is as follows: $eval_M(c) = c$, $eval_M(p) = |M(p)|$, and $eval_M(e_1 \oplus e_2) = eval_M(e_1) \oplus eval_M(e_2)$.

Let φ be a formula. We are interested in the question, whether we can reach from the initial marking some of the goal markings from $G_\varphi = \{M \in \mathcal{M}(N) \mid M \models \varphi\}$. In order to guide the reduction such that transitions that lead to the goal markings are included in the generated stubborn set, we define the notion of *interesting transitions* for a marking

Expression e	$incr_M(e)$	$decr_M(e)$
c	$\emptyset$	$\emptyset$
p	${}^\bullet p$	$p^\bullet$
$e_1 + e_2$	$incr_M(e_1) \cup incr_M(e_2)$	$decr_M(e_1) \cup decr_M(e_2)$
$e_1 - e_2$	$incr_M(e_1) \cup decr_M(e_2)$	$decr_M(e_1) \cup incr_M(e_2)$
$e_1 * e_2$	$incr_M(e_1) \cup decr_M(e_1) \cup$ $incr_M(e_2) \cup decr_M(e_2)$	$incr_M(e_1) \cup decr_M(e_1) \cup$ $incr_M(e_2) \cup decr_M(e_2)$

Table 2: Increasing and decreasing transitions of expression e

M relative to φ, and we let $A_M(\varphi) \subseteq T$ denote the set of interesting transitions. Formally, we shall require that whenever $M \xrightarrow{w} M'$ via a sequence of transitions $w = t_1 t_2 \dots t_n \in T^*$ where $M \notin G_\varphi$ and $M' \in G_\varphi$, then there must exist i, $1 \leq i \leq n$, such that $t_i \in A_M(\varphi)$.

Table 1 gives a possible definition of $A_M(\varphi)$. Let us remark that the definition is at several places nondeterministic, allowing for a variety of sets of interesting transitions. Table 1 uses the functions $incr_M : E_N \to 2^T$ and $decr_M : E_N \to 2^T$ defined in Table 2. These functions take as input an expression e, and return all transitions that can possibly, when fired, increase resp. decrease the evaluation of e. The following lemma formally states the required property of the functions $incr_M$ and $decr_M$.

Lemma 1. *Let $N = (P, T_1, T_2, T_{urg}, IA, OA, g, w, Type, I)$ be a TAPN and $M \in \mathcal{M}(N)$ a marking. Let $e \in E_N$ and let $M \xrightarrow{w} M'$ where $w = t_1 t_2 \dots t_n \in T^*$.*

- *If $eval_M(e) < eval_{M'}(e)$ then there is i, $1 \leq i \leq n$, such that $t_i \in incr_M(e)$.*
- *If $eval_M(e) > eval_{M'}(e)$ then there is i, $1 \leq i \leq n$, such that $t_i \in decr_M(e)$.*

We finish this section with the main technical lemma, showing that at least one interesting transition must be fired before we can reach a marking satisfying a given reachability formula.

Lemma 2. *Let $N = (P, T_1, T_2, T_{urg}, IA, OA, g, w, Type, I)$ be a TAPN, let $M \in \mathcal{M}(N)$ be its marking and let $\varphi \in \Phi$ be a given formula. If $M \not\models \varphi$ and $M \xrightarrow{w} M'$ where $w \in \overline{A_M(\varphi)}^*$ then $M' \not\models \varphi$.*

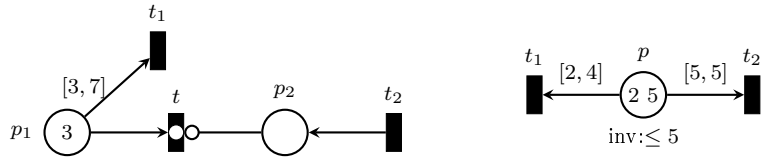
4 Partial Order Reductions for Timed-Arc Petri Nets

We are now ready to state the main theorem that provides sufficient syntax-driven conditions for a reduction in order to guarantee preservation of reachability. Let $N = (P, T_1, T_2, T_{urg}, IA, OA, g, w, Type, I)$ be a TAPN, let $M \in \mathcal{M}(N)$ be a marking of N, and let $\varphi \in \Phi$ be a formula. We recall that $A_M(\varphi)$ is the set of interesting transitions as defined earlier.

Theorem 2 (Reachability Preserving Closure)**.** *Let* St *be a reduction such that for all* $M \in \mathcal{M}(N)$ *it satisfies the following conditions.*

1. *If* $\neg\mathsf{zt}(M)$ *then* $\mathsf{En}(M) \subseteq \mathsf{St}(M)$.
2. *If* $\mathsf{zt}(M)$ *then* $A_M(\varphi) \subseteq \mathsf{St}(M)$.
3. *If* $\mathsf{zt}(M)$ *then either*
   (a) *there is* $t \in T_{urg} \cap \mathsf{En}(M) \cap \mathsf{St}(M)$ *where* ${}^\bullet({}^\circ t) \subseteq \mathsf{St}(M)$, *or*
   (b) *there is* $p \in P$ *where* $I(p) = [a, b]$ *and* $b \in M(p)$ *such that* $t \in \mathsf{St}(M)$ *for every* $t \in p^\bullet$ *where* $b \in g((p, t))$.
4. *For all* $t \in \mathsf{St}(M) \setminus \mathsf{En}(M)$ *either*
   (a) *there is* $p \in {}^\bullet t$ *such that* $|\{x \in M(p) \mid x \in g((p,t))\}| < w((p,t))$ *and*
      - $t' \in \mathsf{St}(M)$ *for all* $t' \in {}^\bullet p$ *where there is* $p' \in {}^\bullet t'$ *with* $Type((t', p)) = Type((p', t')) = Transport_j$ *and where* $g((p', t')) \cap g((p, t)) \neq \emptyset$, *and*
      - *if* $0 \in g((p, t))$ *then also* ${}^\bullet p \subseteq \mathsf{St}(M)$, *or*
   (b) *there is* $p \in {}^\circ t$ *where* $|M(p)| \geq w((p, t))$ *such that*
      - $t' \in \mathsf{St}(M)$ *for all* $t' \in p^\bullet$ *where* $M(p) \cap g((p, t')) \neq \emptyset$.
5. *For all* $t \in \mathsf{St}(M) \cap \mathsf{En}(M)$ *we have*
   (a) $t' \in \mathsf{St}(M)$ *for every* $t' \in p^\bullet$ *where* $p \in {}^\bullet t$ *and* $g((p, t)) \cap g((p, t')) \neq \emptyset$, *and*
   (b) $(t^\bullet)^\circ \subseteq \mathsf{St}(M)$.

Then St *satisfies* $\mathcal{Z}$, $\mathcal{D}$, $\mathcal{R}$, *and* $\mathcal{W}$.



(a) Transitions t_1 and t_2 can disable resp. inhibit the urgent transition t

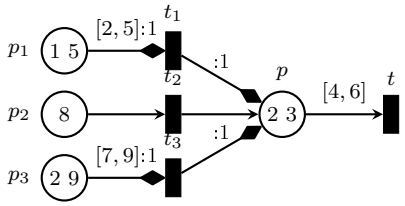
(b) Transition t_2 can remove the token of age 5 from p

Figure 2: Cases for Condition 3

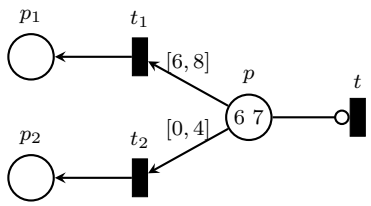
Let us now briefly discuss the conditions of Theorem 2. Clearly, Condition 1 ensures that if time can elapse, we include all enabled transitions into the stubborn set and Condition 2 guarantees that all interesting transitions (those that can potentially make the reachability proposition true) are included as well.

Condition 3 makes sure that that if time elapsing is disabled then any transition that can possibly enable time elapsing will be added to the stubborn set. There are two situations how time progress can be disabled. Either, there is an urgent enabled transition, like the transition t in Figure 2a. Since t_2 can add a token to p_2 and by that inhibit t, Condition 3a makes sure that t_2 is added into the stubborn set in order to satisfy $\mathcal{D}$. As t_1 can remove the token of age 3 from p_1 and hence disable t, we must add t_1 to the stubborn set too (guaranteed by Condition 5a). The other situation when time gets stopped is when a place with an age invariant contains a token that disallows time passing, like in Figure 2b where time is disabled because the place p has a token of age 5, which is the maximum possible age of tokens in p due to the age invariant. Since t_2 can remove the token of age 5 from p, we include it to the stubborn set due to Condition 3b. On the other hand t_1 does not have to be included in the stubborn set as its firing cannot remove the token of age 5 from p.

Condition 4 makes sure that an disabled stubborn transition can never be enabled by a non-stubborn transition. There are two reasons why a transition is disabled. Either, as in Figure 3a where t is disabled, there is an insufficient number of tokens of appropriate age to fire the transition. In this case, Condition 4a makes sure that transitions that can add tokens of a suitable age via transport arcs are included in the stubborn set. This is the case for the transition t_1 in our example, as $[2, 5]$ has a nonempty intersection with $[4, 6]$. On the other hand, t_3 does not have to be added. As the transition t_2 only adds fresh tokens of age

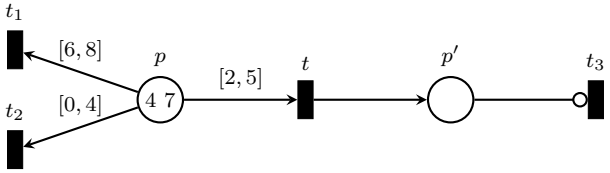


(a) Transition t_1 can transport well-aged tokens into p and enable t



(b) Transition t_1 can enable t by removing tokens from p

Figure 3: Cases for Condition 4



(a) Stubborn transition t can disable both t_2 and t_3

Figure 4: Cases for Condition 5

0 to p via normal arcs, there is no need to add t_2 into the stubborn set either. The other reason for a transition to be disabled is due to inhibitor arcs, as shown on the transition t in Figure 3b. Condition 4b makes sure that t_1 is added to the stubborn set, as it can enable t (the interval $[6, 8]$ has a nonempty intersection with the tokens of age 6 and 7 in the place p). As this is not the case for t_2, this transition can be left out from the stubborn set.

Finally, Condition 5 guarantees that enabled stubborn transitions can never disable any non-stubborn transitions. For an illustration, take a look at Figure 4a and assume that t is an enabled stubborn transition. Firing of t can remove the token of age 4 from p and disable t_2, hence t_2 must become stubborn by Condition 5a in order to satisfy $\mathcal{W}$. On the other hand, the intervals $[6, 8]$ and $[2, 5]$ have empty intersection, so there is no need to declare t_1 as a stubborn transition. Moreover, firing of t can also disable the transition t_3 due to the inhibitor arc, so we must add t_3 to the stubborn set by Condition 5b.

The conditions of Theorem 2 can be turned into an iterative saturation algorithm for the construction of stubborn sets as shown in Algorithm 1 and 2. When running this algorithm for the net in our running example, we can reduce the state space exploration for fireability of the transition t as depicted in Figure 1b. Our last theorem states that the

Algorithm 1: Construction of a reachability preserving stubborn set

input : $N = (P, T_1, T_2, T_{urg}, IA, OA, g, w, Type, I)$, $M \in \mathcal{M}(N)$, $\varphi \in \Phi$
output : $\mathsf{St}(M) \cap \mathsf{En}(M)$

if $\neg\mathsf{zt}(M)$ **then**
  return $\mathsf{En}(M)$;
$Y := A_M(\varphi)$;
if $T_{urg} \cap \mathsf{En}(M) \neq \emptyset$ **then**
  pick any $t \in T_{urg} \cap \mathsf{En}(M)$;
  if $t \notin Y$ **then**
    $Y := Y \cup \{t\}$;
  $Y := Y \cup {}^\bullet({}^\circ t)$;
else
  pick any $p \in P$ where $I(p) = [a, b]$ and $b \in M(p)$
  forall $t \in p^\bullet$ **do**
    if $b \in g((p, t))$ **then**
      $Y := Y \cup \{t\}$;
$X := \mathit{Saturate}(Y)$;
return $X \cap \mathsf{En}(M)$;

algorithm returns stubborn subsets of enabled transitions that satisfy the four conditions of Theorem 1 and hence we preserve the reachability property as well as the minimum path to some reachable goal.

Theorem 3. *Algorithm 1 terminates and returns* $\mathsf{St}(M) \cap \mathsf{En}(M)$ *for some reduction* St *that satisfies* $\mathcal{Z}$, $\mathcal{D}$, $\mathcal{R}$, *and* $\mathcal{W}$.

5 Implementation and Experiments

We implemented our partial order method in C++ and integrated it within the model checker TAPAAL [18] and its discrete time engine `verifydtapn` [4, 10]. We evaluate our partial order reduction on a wide range of case studies.

PatientMonitoring. The patient monitoring system [16] models a medical system that through sensors periodically scans patient's vital functions, making sure that abnormal situations are detected and re-

Algorithm 2: *Saturate*(Y)

```
1  X := ∅;
2  while Y ≠ ∅ do
3      pick any t ∈ Y;
4      if t ∉ En(M) then
5          if ∃p ∈ •t. |{x ∈ M(p) | x ∈ g((p,t))}| < w((p,t)) then
6              pick any such p;
7              forall t' ∈ •p \ X do
8                  forall p' ∈ •t' do
9                      if Type((t',p)) = Type((p',t')) =
                         Transport_j ∧ g((p',t')) ∩ g((p,t)) ≠ ∅ then
10                         Y := Y ∪ {t'};
11             if 0 ∈ g((p,t)) then
12                 Y := Y ∪ (•p \ X);
13         else
14             pick any p ∈ °t s.t. |M(p)| ≥ w((p,t));
15             forall t' ∈ p• \ X do
16                 if M(p) ∩ g((p,t')) ≠ ∅ then
17                     Y := Y ∪ {t'};
18     else
19         forall p ∈ •t do
20             Y := Y ∪ ({t' ∈ p•|g((p,t)) ∩ g((p,t')) ≠ ∅} \ X);
21         Y := Y ∪ ((t•)° \ X);
22     Y := Y \ {t};
23     X := X ∪ {t};
24 return X;
```

ported within given deadlines. The timed-arc Petri net model was described in [16] for two sensors monitoring patient's pulse rate and oxygen saturation level. We scale the case study by adding additional sensors.

BloodTransfusion. This case study models a larger blood transfusion workflow [15], the benchmarking case study of the little-JIL language. The timed-arc Petri net model was described in [9] and we verify that the workflow is free of deadlocks (unless all sub-workflows correctly terminate). The problem is scaled by the number of patients receiving a

	Time (seconds)		Markings ×1000		Reduction	
Model	NORMAL	POR	NORMAL	POR	%Time	%Markings
PatientMonitoring 3	5.88	0.35	333	28	94	92
PatientMonitoring 4	22.06	0.48	1001	36	98	96
PatientMonitoring 5	80.76	0.65	3031	44	99	99
PatientMonitoring 6	305.72	0.85	9248	54	100	99
PatientMonitoring 7	5516.93	5.75	130172	318	100	100
BloodTransfusion 2	0.32	0.41	48	43	-28	11
BloodTransfusion 3	7.88	6.45	792	546	18	31
BloodTransfusion 4	225.18	109.30	14904	7564	51	49
BloodTransfusion 5	5256.01	1611.14	248312	94395	69	62
FireAlarm 10	28.95	14.17	796	498	51	37
FireAlarm 12	116.97	17.51	1726	526	85	70
FireAlarm 14	598.89	21.65	5367	554	96	90
FireAlarm 16	5029.25	29.48	19845	582	99	97
FireAlarm 18	27981.90	34.55	77675	610	100	99
FireAlarm 20	154495.29	41.47	308914	638	100	100
FireAlarm 80	> 2 days	602.71	-	1522	-	-
FireAlarm 125	> 2 days	1957.00	-	2260	-	-
BAwPC 2	0.21	0.41	19	16	-95	15
BAwPC 4	3.45	4.04	193	125	-17	35
BAwPC 6	23.01	17.08	900	452	26	50
BAwPC 8	73.73	39.29	2294	952	47	58
BAwPC 10	135.62	60.66	3819	1412	55	63
BAwPC 12	173.09	73.53	4736	1665	58	65
Fischer-9	3.24	2.37	281	233	27	17
Fischer-11	12.68	8.73	923	738	31	20
Fischer-13	42.52	28.53	2628	2041	33	22
Fischer-15	121.31	77.50	6700	5066	36	24
Fischer-17	313.69	198.36	15622	11536	37	26
Fischer-19	748.52	456.30	33843	24469	39	28
Fischer-21	1622.69	985.07	68934	48904	39	29
LynchShavit 9	3.98	3.31	282	234	17	17
LynchShavit 11	15.73	12.19	925	740	23	20
LynchShavit 13	51.08	37.97	2631	2043	26	22
LynchShavit 15	146.63	103.63	6703	5069	29	24
LynchShavit 17	384.52	258.09	15626	11540	33	26
LynchShavit 19	907.60	597.68	33848	24474	34	28
LynchShavit 21	2011.58	1307.72	68940	48910	35	29
MPEG2 3	13.17	15.43	2188	2187	-17	0
MPEG2 4	109.62	125.45	15190	15180	-14	0
MPEG2 5	755.54	840.84	87568	87478	-11	0
MPEG2 6	4463.19	5092.58	435023	434354	-14	0
AlternatingBit 20	9.17	9.51	617	617	-4	0
AlternatingBit 30	48.20	49.13	2804	2804	-2	0
AlternatingBit 40	161.18	162.94	8382	8382	-1	0
AlternatingBit 50	408.34	408.86	19781	19781	0	0

Table 3: Experiments with and without partial order reduction (POR)

blood transfusion. *FireAlarm.* This case study uses a modified (due to trade secrets) fire alarm system owned by a German company [20, 19]. It models a four-channel round-robin frequency-hopping transmission scheduling in order to ensure a reliable communication between a number of wireless sensors (by which the case study is scaled) and a central control unit. The protocol is based on time-division multiple access (TDMA) channel access and we verify that for a given frequency-jammer, it takes never more than three cycles before a fire alarm is communicated to the central unit. *BAwPC.* Business Activity with Participant Completion (BAwPC) is a web-service coordination protocol from WS-BA specification [32] that ensures a consistent agreement on the outcome of long-running distributed applications. In [29] it was shown that the protocol is flawed and a correct, enhanced variant was suggested. We model check this enhanced protocol and scale it by the capacity of the communication buffer. *Fischer.* Here we consider a classical Fischer's protocol for ensuring mutual exclusion for a number of timed processes. The timed-arc Petri net model is taken from [1] and it is scaled by the number of processes. *LynchShavit.* This is another timed-based mutual exclusion algorithm by Lynch and Shavit, with the timed-arc Petri net model taken from [2] and scaled by the number of processes. *MPEG2.* This case study describes the workflow of the MPEG-2 video encoding algorithm run on a multicore processor (the timed-arc Petri net model was published in [34]) and we verify the maximum duration of the workflow. The model is scaled by the number of B frames in the IB^nP frame sequence. *AlternatingBit.* This is a classical case study of alternating bit protocol, based on the timed-arc Petri net model given in [23]. The purpose of the protocol is to ensure a safe communication between a sender and a receiver over an unreliable medium. Messages are time-stamped in order to compensate (via retransmission) for the possibility of losing messages. The case study is scaled by the maximum number of messages in transfer.

All experiments were run on AMD Opteron 6376 Processors with 500 GB memory. In Table 3 we compare the time to verify a model without (NORMAL) and with (POR) partial order reduction, the number of explored markings (in thousands) and the percentage of time and memory reduction. We can observe clear benefits of our technique on PatientMonitoring, BloodTransfusion and FireAlarm where we are both exponentially faster and explore only a fraction of all reachable markings. For example in FireAlarm, we are able to verify its correctness for all 125

sensors, as it is required by the German company [19]. This would be clearly unfeasible without the use of partial order reduction.

In BAwPC, we can notice that for the smallest instances, there is some computation overhead from computing the stubborn sets, however, it clearly pays off for the larger instances where the percentages of reduced state space are closely followed by the percentages of the verification times and in fact improve with the larger instances. Fischer and LynchShavit case studies demonstrate that even moderate reductions of the state space imply considerable reduction in the running time and computing the stubborn sets is well worth the extra effort.

MPEG2 is an example of a model that allows only negligible reduction of the state space size, and where we observe an actual slowdown in the running time due to the computation of the stubborn sets. Nevertheless, the overhead stays constant in the range of about 15%, even for increasing instance sizes. Finally, AlternatingBit protocol does not allow for any reduction of the state space (even though it contains age invariants) but the overhead in the running time is negligible.

We observed similar performance of our technique also for the cases where the reachability property does not hold and a counter example can be generated.

6 Conclusion

We suggested a simple, yet powerful and application-ready partial order reduction for timed systems. The reduction comes into effect as soon as the timed system enters an urgent configuration where time cannot elapse until a nonempty sequence of transitions gets executed. The method is implemented and fully integrated, including GUI support, into the open-source tool TAPAAL. We demonstrated its practical applicability on several case studies and conclude that computing the stubborn sets causes only a minimal overhead while providing large benefits for reducing the state space in numerous models. The method is not specific to stubborn reduction technique only and it preserves the shortest execution sequences. Moreover, once the time gets urgent, other classical (untimed) partial order approaches should be applicable too. Our method was instantiated to (unbounded) timed-arc Petri nets with discrete time semantics, however, we claim that the technique allows for general application to other modelling formalisms like timed automata and timed Petri nets, as well as an extension to continuous time. We are

currently working on adapting the theory and providing an efficient implementation for UPPAAL-style timed automata with continuous time semantics.

Acknowledgements. We thank Mads Johannsen for his help with the GUI support for partial order reduction. The work was funded by the center IDEA4CPS, Innovation Fund Denmark center DiCyPS and ERC Advanced Grant LASSO. The last author is partially affiliated with FI MU in Brno.

References

[1] P.A. Abdulla and A. Nylén. "Timed Petri Nets and BQOs". In: *Applications and Theory of Petri Nets*. Vol. 2075. LNCS. Springer Berlin Heidelberg, 2001, pp. 53–70. DOI: 10.1007/3-540-45740-2_5.

[2] P.A. Abdulla et al. "Using Forward Reachability Analysis for Verification of Timed Petri Nets". In: *Nordic J. of Computing* 14.1 (2007). Publishing Association Nordic Journal of Computing, pp. 1–42. DOI: 10.5555/1515784.1515785.

[3] R. Alur and D. Dill. "The Theory of Timed Automata". In: *Real-Time: Theory in Practice*. Vol. 600. LNCS. Springer Berlin Heidelberg, 1992, pp. 45–73. DOI: 10.1007/BFb0031987.

[4] M. Andersen et al. "Verification of Liveness Properties on Closed Timed-Arc Petri Nets". In: *Mathematical and Engineering Methods in Computer Science*. Vol. 7721. LNCS. Springer Berlin Heidelberg, 2013, pp. 69–81. DOI: 10.1007/978-3-642-36046-6_8.

[5] E. André, T. Chatain, and C. Rodríguez. "Preserving Partial-Order Runs in Parametric Time Petri Nets". In: *ACM Trans. Embed. Comput. Syst.* 16.2 (2016). Association for Computing Machinery, pp. 1–26. DOI: 10.1145/3012283.

[6] C. Baier and J.P. Katoen. *Principles of Model Checking*. The MIT Press, 2009. ISBN: 026202649X, 9780262026499.

[7] G. Behrmann et al. "Lower and upper bounds in zone-based abstractions of timed automata". In: *International Journal on Software Tools for Technology Transfer* 8.3 (2006). Springer, pp. 204–215. DOI: 10.1007/s10009-005-0190-0.

[8] J. Bengtsson et al. "Partial Order Reductions for Timed Systems". In: *International Conference on Concurrency Theory*. Springer Berlin Heidelberg, 1998, pp. 485–500.

[9] C. Bertolini, Z. Liu, and J. Srba. "Verification of Timed Healthcare Workflows Using Component Timed-Arc Petri Nets". In: *Foundations of Health Information Engineering and Systems*. Vol. 7789. LNCS. Springer Berlin Heidelberg, 2013, pp. 19–36. DOI: 10.1007/978-3-642-39088-3_2.

[10] S.V. Birch et al. "Interval Abstraction Refinement for Model Checking of Timed-Arc Petri Nets". In: *Formal Modeling and Analysis of Timed Systems*. Vol. 8711. LNCS. Springer International Publishing, 2013, pp. 237–251. DOI: 10.1007/978-3-319-10512-3_17.

[11] H. Boucheneb and K. Barkaoui. "Reducing Interleaving Semantics Redundancy in Reachability Analysis of Time Petri Nets". In: *ACM Trans. Embed. Comput. Syst.* 12.1 (2013). Association for Computing Machinery, pp. 1–24. DOI: 10.1145/2406336.2406343.

[12] H. Boucheneb and K. Barkaoui. "Stubborn Sets for Time Petri Nets". In: *ACM Trans. Embed. Comput. Syst.* 14.1 (2015). Association for Computing Machinery, pp. 1–25. DOI: 10.1145/2680541.

[13] H. Boucheneb, K. Barkaoui, and K. Weslati. "Delay-Dependent Partial Order Reduction Technique for Time Petri Nets". In: *Formal Modeling and Analysis of Timed Systems*. Vol. 8711. LNCS. Springer International Publishing, 2014, pp. 53–68. DOI: 10.1007/978-3-319-10512-3_5.

[14] M. Bozga et al. "The IF Toolset". In: *Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design of Computer, Communication, and Software Systems*. Vol. 3185. LNCS. Springer Berlin Heidelberg, 2004, pp. 237–267. DOI: 10.1007/978-3-540-30080-9_8.

[15] S.C. Christov et al. "A Benchmark for Evaluating Software Engineering Techniques for Improving Medical Processes". In: *Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care*. SEHC'10. Association for Computing Machinery, 2010, pp. 50–56. DOI: 10.1145/1809085.1809092.

[16] F. Cicirelli, A. Furfaro, and L. Negro. "Model checking time-dependent system specifications using Time Stream Petri Nets and UPPAAL". In: *Applied Mathematics and Computation* 218.16 (2012). Elsevier, pp. 8160–8186. DOI: 10.1016/j.amc.2012.02.018.

[17] D. Dams et al. "Partial-order Reduction Techniques for Real-time Model Checking". In: *Formal Aspects of Computing* 10.5 (1998). Springer, pp. 469–482. DOI: 10.1007/s001650050028.

[18] A. David et al. "TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 7214. LNCS. Springer Berlin Heidelberg, 2012, pp. 492–497. DOI: 10.1007/978-3-642-28756-5_36.

[19] S. Feo-Arenis et al. "Ready for Testing: Ensuring Conformance to Industrial Standards Through Formal Verification". In: *Formal Aspects of Computing* 28.3 (2016). Springer, pp. 499–527. DOI: 10.1007/s00165-016-0365-3.

[20] S. Feo-Arenis et al. "The Wireless Fire Alarm System: Ensuring Conformance to Industrial Standards through Formal Verification". In: *Formal Methods*. Vol. 8442. LNCS. Springer Berlin Heidelberg, 2014, pp. 307–318. DOI: 10.1007/978-3-319-06410-9_44.

[21] G. Gardey et al. "Romeo: A Tool for Analyzing Time Petri Nets". In: *Computer Aided Verification*. Vol. 3576. LNCS. Springer Berlin Heidelberg, 2005, pp. 418–423. DOI: 10.1007/11513988_41.

[22] H. Hansen et al. "Diamonds Are a Girl's Best Friend: Partial Order Reduction for Timed Automata with Abstractions". In: *Computer Aided Verification*. Vol. 8559. LNCS. Springer International Publishing, 2014, pp. 391–406. DOI: 10.1007/978-3-319-08867-9_26.

[23] L. Jacobsen et al. "Verification of Timed-Arc Petri Nets". In: *SOFSEM 2011: Theory and Practice of Computer Science*. Vol. 6543. LNCS. Springer Berlin Heidelberg, 2011, pp. 46–72. DOI: 10.1007/978-3-642-18381-2_4.

[24] P.G. Jensen, K.G. Larsen, and J. Srba. "Discrete and Continuous Strategies for Timed-Arc Petri Net Games". In: *International Journal on Software Tools for Technology Transfer* 20.5 (2018). Springer, pp. 529–546. DOI: 10.1007/s10009-017-0473-2.

[25] F. Kordon et al. *Complete Results for the 2016 Edition of the Model Checking Contest.* http://mcc.lip6.fr/2016/results.php.

[26] L.M. Kristensen, K. Schmidt, and A. Valmari. "Question-Guided Stubborn Set Methods for State Properties". In: *Formal Methods in System Design* 29.3 (2006). Springer, pp. 215–251. DOI: 10.1007/s10703-006-0006-1.

[27] J. Lilius. "Efficient State Space Search for Time Petri Nets". In: *Electronic Notes in Theoretical Computer Science* 18.1 (1998). Elsevier, pp. 113–133. DOI: 10.1016/S1571-0661(05)80254-3.

[28] D. Lugiez, P. Niebert, and S. Zennou. "A Partial Order Semantics Approach to the Clock Explosion Problem of Timed Automata". In: *Tools and Algorithms for the Construction and Analysis of Systems.* Vol. 2988. LNCS. Springer Berlin Heidelberg, 2004, pp. 296–311. DOI: 10.1007/978-3-540-24730-2_24.

[29] A.P. Marques et al. "Model-Checking Web Services Business Activity Protocols". In: *International Journal on Software Tools for Technology Transfer* 15.2 (2013). Springer, pp. 125–147. DOI: 10.1007/s10009-012-0231-4.

[30] J.A. Mateo, J. Srba, and M.G. Sørensen. "Soundness of Timed-Arc Workflow Nets in Discrete and Continuous-Time Semantics". In: *Fundamenta Informaticae* 140.1 (2015). IOS Press, pp. 89–121. DOI: 10.3233/FI-2015-1246.

[31] M. Minea. "Partial Order Reduction for Model Checking of Timed Automata". In: *International Conference on Concurrency Theory.* Vol. 1664. LNCS. Springer Berlin Heidelberg, 1999, pp. 431–446. DOI: 10.1007/3-540-48320-9_30.

[32] E. Newcomer and I. Robinson. *Web Services Business Activity (WS-BusinessActivity) Version 1.* http://docs.oasis-open.org/ws-tx/wstx-wsba-1.2-spec-os/wstx-wsba-1.2-spec-os.html. 2009.

[33] P. Niebert and H. Qu. "Adding Invariants to Event Zone Automata". In: *Formal Modeling and Analysis of Timed Systems*. Vol. 4202. LNCS. Springer Berlin Heidelberg, 2006, pp. 290–305. DOI: 10.1007/11867340_21.

[34] F.L. Pelayo et al. "Applying Timed-Arc Petri Nets to Improve the Performance of the Mpeg-2 Encoding Algorithm". In: *International Multimedia Modelling Conference*. MMM'04. IEEE, 2004, pp. 49–56. DOI: 10.1109/MULMM.2004.1264966.

[35] M. Perin and J. Faure. "Coupling Timed Plant and Controller Models With Urgent Transitions Without Introducing Deadlocks". In: *17th International Conference on Emerging Technologies & Factory Automation*. ETFA'12. IEEE, 2012, pp. 1–9. DOI: 10.1109/ETFA.2012.6489682.

[36] R.B. Salah, M. Bozga, and O. Maler. "On Interleaving in Timed Automata". In: *International Conference on Concurrency Theory*. Vol. 4137. LNCS. Springer Berlin Heidelberg, 2006, pp. 465–476. DOI: 10.1007/11817949_31.

[37] R.H. Sloan and U. Buy. "Stubborn Sets for Real-Time Petri Nets". In: *Formal Methods in System Design* 11.1 (1997). Springer, pp. 23–40. DOI: 10.1023/A:1008629725384.

[38] A. Valmari. "Stubborn Set Intuition Explained". In: *Transactions on Petri Nets and Other Models of Concurrency XII*. Vol. 10470. LNCS. Springer Berlin Heidelberg, 2017, pp. 140–165. DOI: 10.1007/978-3-662-55862-1_7.

[39] F. Vernadat and B. Berthomeiu. "Time Petri Nets Analysis with TINA". In: *Third International Conference on the Quantitative Evaluation of Systems*. QEST'06. IEEE, 2006, pp. 123–124. DOI: 10.1109/QEST.2006.56.

[40] I. Virbitskaite and E. Pokozy. "A Partial Order Method for the Verification of Time Petri Nets". In: *Fundamentals of Computation Theory*. Vol. 1684. LNCS. Springer Berlin Heidelberg, 1999, pp. 547–558. DOI: 10.1007/3-540-48321-7_46.

[41] T. Yoneda and B. Schlingloff. "Efficient Verification of Parallel Real–Time Systems". In: *Formal Methods in System Design* 11.2 (1997). Springer, pp. 187–215. DOI: 10.1023/A:1008682131325.

A Proof of Lemma 1

Proof. Let $N = (P, T_1, T_2, T_{urg}, IA, OA, g, w, Type, I)$ be a TAPN and $M \in \mathcal{M}(N)$ a marking. Let $e \in E_N$ and let $M \xrightarrow{w} M'$ where $w = t_1 t_2 \dots t_n \in T^*$. The proof proceeds by structural induction on e. We only show the *incr* cases as it is analogous to *decr*.

$e = c$: Trivial

$e = p$: We here prove that if for all $i \in [1, n]$ we have $t_i \notin incr_M(p)$ then we have that $eval_M(p) \geq eval_{M'}(p)$, which implies Lemma 1 for this case. We know ${}^\bullet p \subseteq incr_M(p)$ by Table 2. Therefore for all $i \in [1, n]$ we have $t_i \notin {}^\bullet p$ and we must have that $|M(p)| \geq |M'(p)|$ and $eval_M(p) \geq eval_{M'}(p)$, since t_i cannot increase the number of tokens in p as it would otherwise contradict the definition of ${}^\bullet p$.

$e = e_1 + e_2$: In Table 2 we see that $incr_M(e) = incr_M(e_1) \cup incr_M(e_2)$. By the induction hypothesis we know $eval_M(e_1) < eval_{M'}(e_1)$ and $eval_M(e_2) < eval_{M'}(e_2)$, and therefore also $eval_M(e) < eval_{M'}(e)$ since we have $eval_M(e) = eval_M(e_1) + eval_M(e_2) < eval_{M'}(e_1) + eval_{M'}(e_2) = eval_{M'}(e)$.

$e = e_1 - e_2$: In Table 2 we see that $incr_M(e) = incr_M(e_1) \cup decr_M(e_2)$. By the induction hypothesis we know $eval_M(e_1) < eval_{M'}(e_1)$ and $eval_M(e_2) > eval_{M'}(e_2)$, and therefore also $eval_M(e) < eval_{M'}(e)$ since we have $eval_M(e) = eval_M(e_1) - eval_M(e_2) < eval_{M'}(e_1) - eval_{M'}(e_2) = eval_{M'}(e)$.

$e = e_1 * e_2$: We know $eval_M(e) < eval_{M'}(e)$ and $incr_M(e) = incr_M(e_1) \cup decr_M(e_1) \cup incr_M(e_2) \cup decr_M(e_2)$ due to Table 2. If for all $i \in [1, n]$ we have $t_i \notin incr_M(e)$ then $eval_M e_1 = eval_{M'}(e_1)$ and $eval_M(e_2) = eval_{M'}(e_2)$ and therefore $eval_M(e) = eval_{M'}(e)$. Since $eval_M(e) < eval_{M'}(e)$ we must have that there exists i, $1 \leq i \leq n$, such that $t_i \in incr_M(e)$.

□

B Proof of Lemma 2

Proof. Let $N = (P, T_1, T_2, T_{urg}, IA, OA, g, w, Type, I)$ be a TAPN, $M \in \mathcal{M}(N)$ be a marking, and $\varphi \in \Phi$ a given formula. Assume that $M \not\models \varphi$. The proof proceeds by structural induction on φ.

$\varphi =$ *deadlock*: If $M \not\models \varphi$ then there exists $t \in \mathsf{En}(M)$ s.t. $({}^\bullet t)^\bullet \cup {}^\bullet({}^\circ t) \subseteq A_M(\varphi)$. To disable t we have to fire a transition $t' \in ({}^\bullet t)^\bullet \cup {}^\bullet({}^\circ t)$ to either remove or add tokens that will disable or inhib t, respectively. Since $({}^\bullet t)^\bullet \cup {}^\bullet({}^\circ t) \subseteq A_M(\varphi)$ we have $M' \not\models \varphi$.

$\varphi = t$: If $M \not\models \varphi$ then $t \notin \mathsf{En}(M)$. Either there exists $p \in {}^\bullet t$ s.t. $|M(p)| < w((p,t))$ and we have to fire some $t' \in {}^\bullet p$ to enable t, or there exists $p \in {}^\circ t$ s.t. $|M(p)| \geq w((p,t))$ and we have to fire some $t'' \in p^\bullet$ to enabled t. Since ${}^\bullet p \subseteq A_M(\varphi)$ or $p^\bullet \subseteq A_M(\varphi)$, respectively, we have $M' \not\models \varphi$.

$\varphi = e_1 < e_2$: If $M \not\models \varphi$ then $eval_M(e_1) \geq eval_M(e_2)$. Since $decr_M(e_1) \cup incr_M(e_2) \subseteq A_M(\varphi)$ we know $eval_M(e_1) \geq eval_{M'}(e_1)$ and $eval_M(e_2) \leq eval_{M'}(e_2)$ because of Lemma 1, and therefore $M' \not\models \varphi$.

$\varphi = e_1 > e_2$: If $M \not\models \varphi$ then $eval_M(e_1) \leq eval_M(e_2)$. Since $incr_M(e_1) \cup decr_M(e_2) \subseteq A_M(\varphi)$ we know $eval_M(e_1) \leq eval_{M'}(e_1)$ and $eval_M(e_2) \geq eval_{M'}(e_2)$ because of Lemma 1, and therefore $M' \not\models \varphi$.

$\varphi = \varphi_1 \wedge \varphi_2$: If $M \not\models \varphi$ then there exists $i \in \{1,2\}$ s.t. $M \not\models \varphi_i$. We know $A_M(\varphi_i) \subseteq A_M(\varphi)$ which by the induction hypothesis implies $M' \not\models \varphi_i$. By the semantics of φ we also have that $M' \not\models \varphi$.

$\varphi = \varphi_1 \vee \varphi_2$: If $M \not\models \varphi$ then $M \not\models \varphi_1$ and $M \not\models \varphi_2$. We know $A_M(\varphi_1) \cup A_M(\varphi_2) \subseteq A_M(\varphi)$ which by the induction hypothesis implies $M' \not\models \varphi_1$ and $M' \not\models \varphi_2$. By the semantics of φ we also have that $M' \not\models \varphi$.

$\varphi = \neg t$: If $M \not\models \varphi$ then $t \in \mathsf{En}(M)$. To disable t we have to fire at least one transition $t' \in ({}^\bullet t)^\bullet \cup {}^\bullet({}^\circ t)$ to either remove or add tokens that will disable or inhib t, respectively. Since $({}^\bullet t)^\bullet \cup {}^\bullet({}^\circ t) \subseteq A_M(\varphi)$ we have $M' \not\models \varphi$.

The remaining cases and negation cases are analogous. □

C Proof of Theorem 2

Proof. We shall argue that any reduction St satisfying the conditions of the theorem has the properties $\mathcal{Z}$, $\mathcal{D}$, $\mathcal{R}$, and $\mathcal{W}$.

- ($\mathcal{Z}$): Follows from Condition 1.

- $(\mathcal{R})$: Follows from Lemma 2 and Condition 2.
- $(\mathcal{D})$: Let $M \in \mathcal{M}(N)$ be a marking and $w \in \overline{\mathsf{St}(M)}^*$ s.t. $M \xrightarrow{w} M'$. We will show that if $\mathsf{zt}(M)$ then $\mathsf{zt}(M')$.

  Assume that $\mathsf{zt}(M)$. Since $\mathsf{zt}(M)$ there are two cases: $T_{urg} \cap \mathsf{En}(M) \neq \emptyset$ or there is $p \in P$ where $I(p) = [a, b]$ and $b \in M(p)$.

  In the first case (Figure 2a), there is $t \in T_{urg} \cap \mathsf{En}(M) \cap \mathsf{St}(M)$ and ${}^\bullet({}^\circ t) \subseteq \mathsf{St}(M)$ due to Condition 3a. For any $p \in {}^\bullet t$ and $p' \in {}^\circ t$ we have that $|\{x \in M(p) \mid x \in g((p,t))\}| \geq w((p,t))$ and $|M(p')| < w((p',t))$. Due to Condition 5a we know for all $t' \in p^\bullet$ that $t' \in \mathsf{St}(M)$ if $g((p,t) \cap g((p,t')) \neq \emptyset$. Therefore we have $|\{x \in M'(p) \mid x \in g((p,t))\}| \geq w((p,t))$ since $w \in \overline{\mathsf{St}(M)}^*$. Due to ${}^\bullet({}^\circ t) \subseteq \mathsf{St}(M)$ in Condition 3a w cannot add any tokens to p' since $w \in \overline{\mathsf{St}(M)}^*$ and we have that $|M'(p')| < w((p',t))$. This implies $\mathsf{zt}(M')$.

  In the second case (Figure 2b), there is $p \in P$ where $I(p) = [a, b]$ and $b \in M(p)$. Due to Condition 3b for all $t \in p^\bullet$ where $b \in g((p,t))$ we have that $t \in \mathsf{St}(M)$. Therefore t can never occur in w since $w \in \overline{\mathsf{St}(M)}^*$ and we must have that $b \in M'(p)$, implying that $\mathsf{zt}(M')$.
- $(\mathcal{W})$: Let $M, M' \in \mathcal{M}(N)$ be markings, $t \in \mathsf{St}(M)$, and $w \in \overline{\mathsf{St}(M)}^*$. We will show that if $M \xrightarrow{wt} M'$ then $M \xrightarrow{tw} M'$.

  Let $M_w \in \mathcal{M}(N)$ be a marking s.t. $M \xrightarrow{w} M_w$. By contradiction assume that $t \notin \mathsf{En}(M)$. Then t is disabled in M because there is $p \in {}^\bullet t$ such that $|\{x \in M(p) \mid x \in g((p,t))\}| < w((p,t))$ or there is $p \in {}^\circ t$ such that $|M(p)| \geq w((p,t))$. In the first case (Figure 3a), due to Condition 4a all the transitions that can add tokens that are in the guard $g((p,t))$ to p are included in $\mathsf{St}(M)$. Since $w \in \overline{\mathsf{St}(M)}^*$ this implies that $|\{x \in M_w(p) \mid x \in g((p,t))\}| < w((p,t))$ and $t \notin \mathsf{En}(M_w)$ contradicting our assumption that $M_w \xrightarrow{t} M'$. In the second case (Figure 3b), due to Condition 4b all the transitions that can remove at least one token from p are included in $\mathsf{St}(M)$. Since $w \in \overline{\mathsf{St}(M)}^*$ this implies that $|M_w(p)| \geq w((p,t))$ and $t \notin \mathsf{En}(M_w)$, again contradicting our assumption that $M_w \xrightarrow{t} M'$. Therefore we must have that $t \in \mathsf{En}(M)$.

  Since $t \in \mathsf{En}(M)$ there is $M_t \in \mathcal{M}(N)$ s.t. $M \xrightarrow{t} M_t$. We have to show that $M_t \xrightarrow{w} M'$ is a possible execution sequence. For the sake

of contradition, assume that this is not the case. Then there must exist a transition t' that occurs in w that became disabled because t was fired. There are two cases: t removed one or more tokens from a shared pre-place $p \in {}^\bullet t \cap {}^\bullet t'$ where $g((p,t)) \cap g((p,t')) \neq \emptyset$ or t added one or more tokens to a place $p \in t^\bullet \cap {}^\circ t'$ (both in Figure 4a). In the first case, due to Condition 5a all the transitions that can remove tokens that are in the guard $g((p,t))$ from p are included in $\mathsf{St}(M)$, implying that $t' \in \mathsf{St}(M)$. Since $w \in \overline{\mathsf{St}(M)}^*$ such a t' cannot exist. In the second case, due to Condition 5b we know that $(t^\bullet)^\circ \subseteq \mathsf{St}(M)$, implying that $t' \in \mathsf{St}(M)$. Since $w \in \overline{\mathsf{St}(M)}^*$ such a t' cannot exist. Therefore we must have that $M_t \xrightarrow{w} M'$ and can conlude with $M \xrightarrow{tw} M'$ as requested. We note here that the execution sequences wt and tw must lead to the same marking as the order of transition firings does not impact the reachable marking.

This completes the proof of the theorem. □

D Proof of Theorem 3

Proof. Termination: If $\neg\mathsf{zt}(M)$ then we terminate in line 2. If $A_M(\varphi) = \emptyset$ or no transitions are added to Y in line 5-8 or in line 11-13 then $Y = \emptyset$, we never enter the while-loop in Algorithm 2, and we terminate. Otherwise $Y \neq \emptyset$ and we enter the while-loop in Algorithm 2. Notice that $X \cap Y = \emptyset$ is always the case in the execution of Algorithm 2. We never remove transitions from X after they have been added. Therefore, since in line 23 a new transition is added to X at the end of each loop iteration, the loop can iterate at most once for each transition. Since T is finite by the TAPN definition, the loop iterates a finite number of times, and we terminate.

Correctness: It was shown that the construction in Theorem 2 satisfies $\mathcal{Z}$, $\mathcal{D}$, $\mathcal{R}$, and $\mathcal{W}$. It is therefore sufficient to show that Algorithm 1 replicates the construction. Notice that every transition that is added to Y is eventually added to X in line 23 in Algorithm 2 and returned in line 15. Let $t \in Y$.

Condition 1: If $\neg\mathsf{zt}(M)$ then we return $\mathsf{En}(M)$ in line 2.

Condition 2: We have $A_M(\varphi) \subseteq Y$ in line 3.

Condition 3a: In line 5 we pick any $t \in \mathsf{En}(M) \cap T_{urg}$, and in line 7 we have $t \in Y$ and in line 8 we have ${}^\bullet({}^\circ t) \subseteq Y$.

Condition 3b: In line 10 we pick a p where there exist $b \in M(p)$ s.t. $I(p) = [a, b]$, and in line line 11-13 we add all the $t \in p^\bullet$ where $b \in g((p,t))$ to Y.

Condition 4a: In line 5 and 6 in Algorithm 2 we pick a $p \in {}^\bullet t$ s.t. $|\{x \in M(p) \mid x \in g((p,t))\}| < w((p,t))$. In line 7 through 10 in Algorithm 2 we iterate through the relevant transport arcs and add them to Y. We check for $0 \in g((p,t))$ in line 11 in Algorithm 2 and add ${}^\bullet p$ to Y in line 12.

Condition 4b: In line 14 in Algorithm 2 we pick any $p \in {}^\circ$ s.t. $|M(p)| \geq w((p,t))$. In line 15 through 17 in Algorithm 2 we iterate over all $t' \in p^\bullet$ not already in X, and if t' is able to remove a token from p it is added to Y.

Condition 5a: In line 19 and 20 in Algorithm 2 for all $p \in {}^\bullet t$ and all $t' \in p^\bullet$ we add t' to Y if $t' \notin X$ and $g((p,t)) \cap g((p,t')) \neq \emptyset$.

Condition 5b: In line 21 in Algorithm 2 we add the transitions of $(t^\bullet)^\circ$ that are not already in X to Y. □

E User Manual for Partial Order Reduction in TAPAAL

The newest release 3.4.0 of the model checker TAPAAL includes the implementation of discrete-time partial order reduction for timed-arc Petri nets. It is available as precompiled binaries for Windows, Mac, and Linux operating systems at: `http://www.tapaal.net/download`. It requires Java run-time environment installed on the machine. At the bottom of the download page, one can obtain also the zip file `partial-order-experiments.zip` with all .xml model files for the case studies described in Section 5.

In Figure 5 we display a screenshot of TAPAAL 3.4.0 graphical user interface. To open a file, such as the FireAlarm-12 case study as seen in Figure 5, select the File dropdown menu, followed by the Open option, and then navigate to the appropriate directory and open the .xml model file. In the main work space in Figure 5, we see the sensor1_4ch compoment of the FireAlarm-12 model. The net can be modified using the toolbox menu at the top of the screen below the window options. Manual simulation mode can be enabled by selecting the green flag in the toolbox menu. Verification can be done by adding, or editing an existing query, in the Queries section of the interface, where the edit option is highlighted by the cursor in Figure 5. Selecting edit will open the query configuration window for the query "No More Than 3 Misses" as seen in

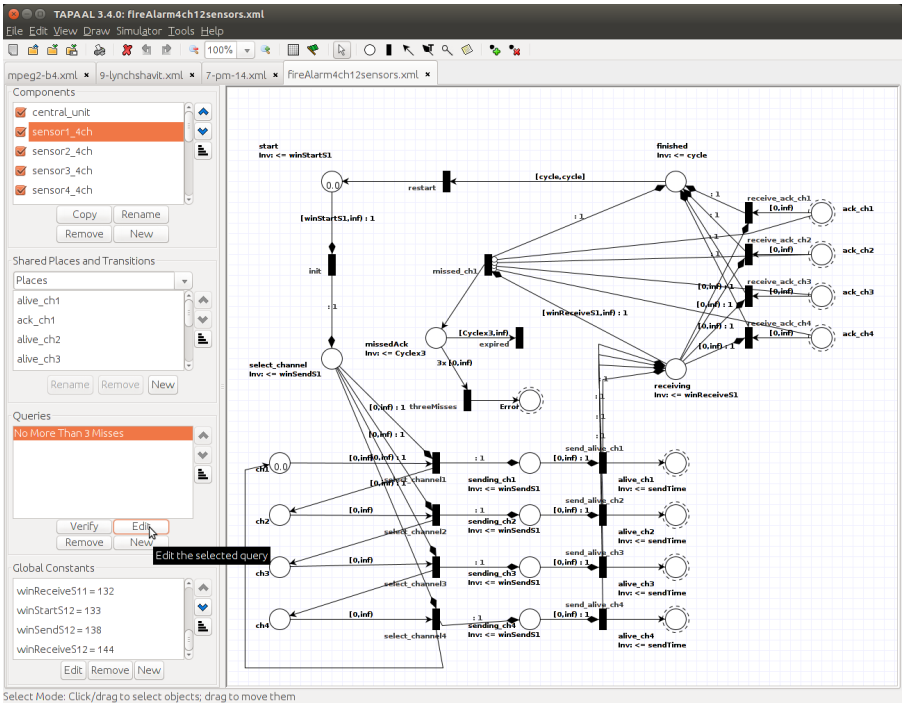


Figure 5: The TAPAAL GUI.

Figure 6.

In this window it is possible to edit queries, such as choosing or changing a quantification, adding logical connectives such as conjunction, and predicates seen in the upper half of Figure 6, and selecting various options to customize the verification of the query seen in the lower half of Figure 6, such as search strategy, traces, etc. Make sure to select the "Advanced View" in order to see the verification options. Now various verification optimization techniques can be utilized to assist the verification engine, such as symmetry reductions and PTries. Among these, there is the discrete partial order reduction implementation which in Figure 6 corresponds to the "Use stubborn reduction" checkbox option highlighted with an ellipse. It is currently enabled in the displayed query dialog. Selecting the "Save and Verify" option saves any changes made to the custimization of the query and begins the verification of the query using the verification engine. The output of the verifcation can be seen in Figure 7.

Figure 7 shows the output of verifying the query both with and with-

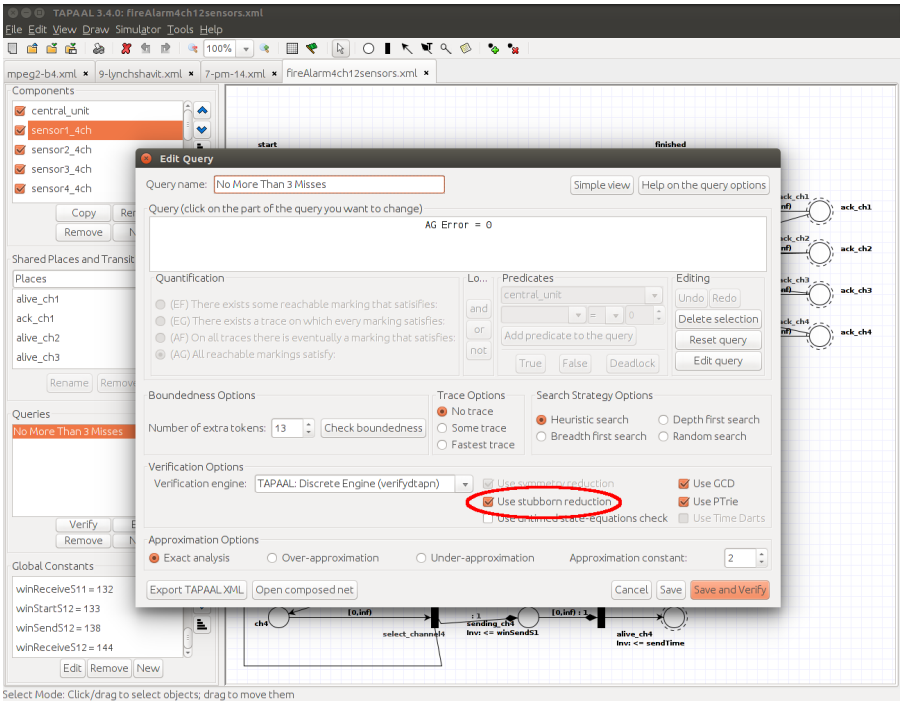
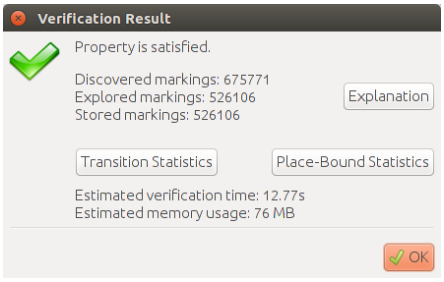
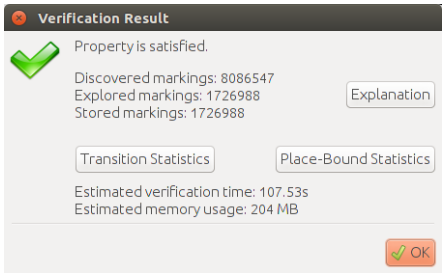


Figure 6: Enabling and disabling partial order reduction

out enabling the partial order reduction, seen in Figure 7a and Figure 7b, respectively. Both cases agree that the property described by the query is satisfied, and show some statistics of the verification. A total of 526106 markings was explored to verify the property with partial order reduction enabled, and a total of 1726988 markings was explored to verify it with partial order reduction disabled. Both correspond to the numbers seen in Table 3 for the FireAlarm-12 case study. The verification time is not the same due to being run on different machines.



(a) Partial order reduction enabled



(b) Partial order reduction disabled

Figure 7: Verification result dialog

Paper B

Stubborn Set Reduction for Two-Player Reachability Games

Frederik M. Bønneland, Peter G. Jensen,
Kim G. Larsen, Marco Muñiz, and Jiří Srba

This paper has been accepted for:
Special issue of the journal *Logical Methods in Computer Science*.
This paper has also previously been published in
International Conference on Concurrency Theory, Leibniz International Proceedings in Informatics Vol. 140, pp. 23:1-23:15, 2019.

Paper C

Stubborn Set Reduction for Timed Reachability and Safety Games

Frederik M. Bønneland, Peter G. Jensen,
Kim G. Larsen, Marco Muñiz, and Jiří Srba

This paper has been submitted for publication.

Paper D

Stubborn Versus Structural Reductions for Petri Nets

Frederik M. Bønneland, Jakob Dyhr, Peter G. Jensen,
Mads Johannsen, and Jiří Srba

This paper has been published in:
Journal of Logical and Algebraic Methods in Programming, Vol. 102(1), pp. 46-64, 2019.

Abstract

Partial order and structural reduction techniques are some of the most beneficial methods for state space reduction in reachability analysis of Petri nets. This is among others documented by the fact that these techniques are used by the leading tools in the annual Model Checking Contest (MCC) of Petri net tools. We suggest improved versions of a partial order reduction based on stubborn sets and of a structural reduction with additional new reduction rules, and we extend both methods for the application on Petri nets with weighted arcs and weighted inhibitor arcs. All algorithms are implemented in the open-source verification tool TAPAAL and evaluated on a large benchmark of Petri net models from MCC'17, including a comparison with the tool LoLA (the last year winner of the competition). The experiments document that both methods provide significant state space reductions and, even more importantly, that their combination is indeed beneficial as a further nontrivial state space reduction can be achieved.

1 Introduction

Model checking of large distributed and concurrent systems is often limited in its applicability due to the state space explosion problem. Components in concurrent systems may independently perform actions without being in conflict with other components, forcing an explicit state space analysis to explore every possible interleaving of the actions and hence creating an explosion in the number of executable action sequences. Petri nets are a popular formalism for modelling of concurrent systems [12], however, due to the state space explosion problem, essentially all interesting questions about their behaviour, including the reachability and coverability problems, are EXPSPACE-hard (see e.g. [4]).

Despite the discouraging complexity results, numerous techniques have been developed to improve the feasibility of reachability analysis, including methods based on reducing the state space by eliminating the interleaving in independent components (see e.g. [5, 1]). The focus of our work is on two such techniques: structural reductions [11] and stubborn set reductions [17], both applied to and evaluated on the model of weighted Petri nets with inhibitor arcs. *Structural reductions* preprocess the Petri net model by collapsing redundant places and transitions, while preserving the validity of the model checking question. The idea

is that a smaller number of places and transitions in a net can help to reduce the degree of concurrency and eliminate some unnecessary interleavings. In partial order reductions, like e.g. *stubborn set reduction*, we identify transitions that are independent of each other and the order of their execution does not influence the model checking property in question. This can be considered as another method that can, in an on-the-fly manner, reduce the number of possible interleavings of independent actions. Both structural and stubborn reductions can be in a straightforward way combined, however, to the best of our knowledge, the effect of this combination has not previously been studied in detail.

We perform a comparative study of the effects of the two types of state space reduction techniques and their combination. For our experiments, we use the database of nets and reachability queries from the annual Model Checking Contest (MCC) [8] and conclude that while both techniques are clearly beneficial for the performance of the reachability analysis, the combination of the two methods demonstrates yet another degree of performance improvements. Apart from this experimental evaluation, we make several technical contributions to stubborn and structural reductions applied to the model of Petri nets. Both techniques are extended to work for the reachability logic used in MCC, while allowing us to use weighted arcs as well as weighted inhibitor arcs. In particular, the stubborn set reduction as well as the structural reduction were refined to take weighted arcs into account, in order to minimize the size of the state space that is necessary to explore for a given reachability query. In stubborn reduction, we refine the computation of dependencies between transitions so that instead of the traditional comparison of presets and postsets of places, we utilize a more detailed analysis of the increasing/decreasing effect of a transition on a given place. All techniques are proved correct and implemented in the model checker TAPAAL [3]. The experiments are encouraging as the improved techniques in their combination allow us to solve more reachability queries from MCC'17 [8] than the model checker LoLA [21], the last year winner in the reachability category.

Related work The stubborn reduction technique is related to and based upon the seminal work on stubborn sets by Valmari et al. [16, 14, 9, 17, 15]. This includes write up/down sets [17], the closure procedure [9], and attractor sets [14]. We contribute by adding support for inhibitor arcs, extending the technique to a reachability logic used in MCC and

presenting a different formulation of stubborn sets for reachability in the general setting of labelled transition systems. Further analysis during the generation of stubborn sets can help to generate more optimal (smaller) stubborn sets, which can be done e.g. by extracting terminal strongly connected components from the derived transition dependency graph [16]. We choose to use instead heuristic methods for the generation of stubborn sets as they have smaller computational overhead and achieve better performance in our experiments.

Structural reductions of Petri nets were studied by Murata et al. [11, 10] with the main focus on preserving liveness, safety, and boundedness. The reduction rules were recently extended to include weighted nets with inhibitor arcs while preserving the reachability of cardinality queries [6]. We contribute by increasing the applicability of the four rules presented in [6] and refining them for the use with weighted arcs so that a more significant net reduction can be achieved compared to [6]. Moreover, we introduce five new reduction rules, allowing us to reduce the size of the input net even further.

Stubborn sets are also an important state space reduction technique used in the tool LoLA [21] that we compare against to in our experiments. Their stubborn set implementation have several approaches to reachability analysis, utilising up/down sets and terminal strongly connected components [9] to mention some. Approaches using terminal strongly connected components can present some performance problems due to concurrent cycles of invisible (or non-interesting) transitions, forcing the method to sometimes explore the full parallel composition [18, 19]. Remedies to this have been explored in the form of frozen actions [18], removing transitions from consideration if they are tagged as frozen. Besides LoLA's take on stubborn sets [14], their tool includes several other reduction and verification improvements such as symmetry reduction [13] and Counter Example Guided Abstraction Refinement (CEGAR) [20], however, LoLA does not employ structural reductions. Our experiments document that the refined and combined application of our stubborn and structural reduction techniques becomes competitive in performance compared with the tool LoLA.

2 Preliminaries

A *labelled transition system* (LTS) is a tuple $G = (\mathcal{S}, A, \rightarrow)$ where $\mathcal{S}$ is a set of states, A is a set of actions (or labels), and $\rightarrow \subseteq \mathcal{S} \times A \times \mathcal{S}$ is a

transition relation. We write $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \rightarrow$ and say that a is *enabled* in s. The set of all enabled actions in a state s is denoted $en(s)$. A state s is a *deadlock* if $en(s) = \emptyset$. We write $s \rightarrow s'$ whenever there is an action a such that $s \xrightarrow{a} s'$. We inductively extend the relation $\xrightarrow{a}$ to sequences of transitions $w \in A^*$ such that $s \xrightarrow{\epsilon} s$ and $s \xrightarrow{wa} s'$ if $s \xrightarrow{w} s''$ and $s'' \xrightarrow{a} s'$. We write $s \rightarrow^n s'$ if there is $w \in T^*$ of length n such that $s \xrightarrow{w} s'$, and we write $s \rightarrow^* s'$ if $s \rightarrow^n s'$ for some $n \geq 0$.

The *reachability problem* is, given an LTS $G = (\mathcal{S}, A, \rightarrow)$, an initial state $s \in \mathcal{S}$, and a set of goal states $G \subseteq \mathcal{S}$, to decide whether there is $s' \in G$ s.t. $s \rightarrow^* s'$.

Petri Nets

Let $\mathbb{N}^0 = \mathbb{N} \cup \{0\}$ be the set of natural numbers including 0. Let $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$ be the set of natural numbers including infinity.

Definition 1 (Petri Net with Inhibitor Arcs)**.** *A Petri net is a tuple $N = (P, T, W, I)$ where P and T are finite and disjoint sets of places and transitions, W: $(P \times T) \cup (T \times P) \rightarrow \mathbb{N}^0$ is a weight function for regular arcs, and $I : (P \times T) \rightarrow \mathbb{N}^\infty$ is a weight function for inhibitor arcs.*

A *marking* M on N is a function $M : P \rightarrow \mathbb{N}^0$, where $M(p)$ denotes the number of tokens in place p. The set of all markings of a Petri net N is written as $\mathcal{M}(N)$. Let $M_0 \in \mathcal{M}(N)$ be a given *initial marking* of N.

A Petri net $N = (P, T, W, I)$ defines an LTS $G(N) = (\mathcal{S}, A, \rightarrow)$ where $\mathcal{S} = \mathcal{M}(N)$ is the set of all markings, $A = T$ is the set of labels, and $M \xrightarrow{t} M'$ whenever for all $p \in P$ we have $M(p) < I((p, t))$ and $M(p) \geq W((p, t))$ such that $M'(p) = M(p) - W((p, t)) + W((t, p))$.

Example 1. An example of a Petri net is given in Figure 1a. We use the standard notation and denote places by circles, transitions by squares and the dots represent tokens in the initial marking. The weights of all arcs are implicitly fixed to 1, the only exception being the arc from p_3 to t_3 that requires two tokens in order to fire t_3. The arrow from p_5 to t_3 with cirle head-tip is an inhibitor arc (again with the default weight 1) and it inhibits the enabledness of t_3 as soon as p_5 contains at least one token. The labelled transition system (containing only markings reachable from the initial one) is depicted in Figure 1b. Here the notation e.g. $2p_1p_4$ represents the initial marking with two tokens in p_1 and one token in p_4.

The net contains lots of interleavings and the markings in dashed boxes are those that can be disregarded once we apply our stubborn set reduction for verifying the reachability of a marking with at least two tokens in the place p_3. In Figure 1c we display a reduced net where the place p_1 and transition t_1 are removed. This structural reduction preserves the reachability of the goal marking and we can see that the reachable state space gets significantly reduced as demonstrated in Figure 1d. An application of a stubborn set reduction on top of the structural reduction allows for an even greater state space reduction, as showed by the dashed boxes which can be removed by stubborn set reduction during the state space search of the reduced net. The details of these methods are explained in the remainder of this paper.

Let us first fix some useful notation. For a place or transition x, we denote the *preset* of x as ${}^\bullet x = \{y \mid W((y,x)) > 0\}$, and the *postset* of x as $x^\bullet = \{y \mid W((x,y)) > 0\}$. For a place p we define the *increasing preset* of p, containing all transitions that increase the number of tokens in p, as ${}^+p = \{t \in {}^\bullet p \mid W((t,p)) > W((p,t))\}$, and similarly the *decreasing postset* of p as $p^- = \{t \in p^\bullet \mid W((t,p)) < W((p,t))\}$. For a transition t, we denote the *inhibitor preset* of t as ${}^\circ t = \{p \mid I((p,t)) < \infty\}$, and for a place p, we denote the *inhibitor postset* of p as $p^\circ = \{t \mid I((p,t)) < \infty\}$. For a set X of either places or transitions, we extend the notation as ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$, and similarly for the other operators.

In order to syntactically define the set of goal states G for the reachability problem on Petri nets, we use the *reachability logic* from the Model Checking Contest [8]. The syntax of the logic is as follows:

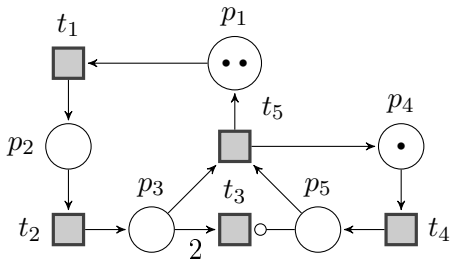
$$\varphi ::= \mathit{true} \mid \mathit{false} \mid \alpha \mid \mathit{deadlock} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi$$

$$\alpha ::= t \mid e_1 \bowtie e_2$$

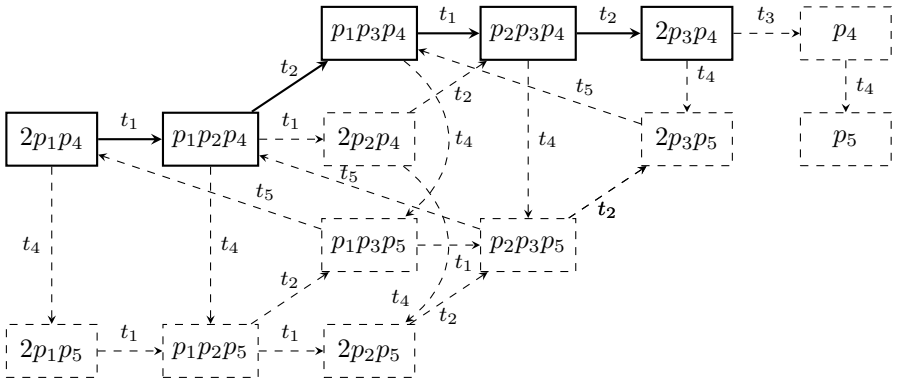
$$e ::= c \mid p \mid e_1 \oplus e_2$$

where $t \in T$, $c \in \mathbb{N}^0$, $\bowtie \in \{<, \leq, =, \neq, >, \geq\}$, $p \in P$, and $\oplus \in \{+, -, \cdot\}$. The evaluation of an arithmetical expression e in a marking M is defined as $\mathit{eval}_M(c) = c$, $\mathit{eval}_M(p) = M(p)$ and $\mathit{eval}_M(e_1 \oplus e_2) = \mathit{eval}_M(e_1) \oplus \mathit{eval}_M(e_2)$. The semantics of a reachability formula φ in a marking M is given in Figure 2.

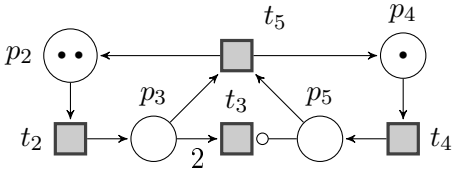
Formulae that do not use any atomic predicate t for transition firing and no predicate *deadlock* are called *cardinality formulae* and formulae that avoid the use of $e_1 \bowtie e_2$ and *deadlock* are called *fireability formulae*. The formula *deadlock* is called the *deadlock formula*.



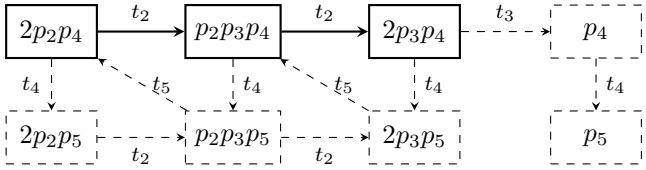
(a) A Petri net (the arc from p_5 to t_3 is the inhibitor arc and the weight of the arc from p_3 to t_3 is 2; all other arcs have the default weight 1)



(b) Reachable state space for the net from Figure 1a (the dashed boxes are removed by a possible stubborn set reduction preserving the reachability of $\varphi = p_3 \geq 2$)



(c) Petri Net from Figure 1a after the application of structural reductions which removed the place p_1 and transition t_1 while preserving the property $\varphi = p_3 \geq 2$



(d) Reachable state space for the net from Figure 1c (the dashed boxes are removed by a possible stubborn set reduction preserving the reachability of $\varphi = p_3 \geq 2$)

Figure 1: Combination of stubborn and structural reductions

$$
\begin{array}{ll}
M \models \mathit{true} & \\
M \not\models \mathit{false} & \\
M \models t & \text{iff } t \in en(M) \\
M \models e_1 \bowtie e_2 & \text{iff } eval_M(e_1) \bowtie eval_M(e_2) \\
M \models \mathit{deadlock} & \text{iff } en(M) = \emptyset \\
M \models \varphi_1 \wedge \varphi_2 & \text{iff } M \models \varphi_1 \text{ and } M \models \varphi_2 \\
M \models \varphi_1 \vee \varphi_2 & \text{iff } M \models \varphi_1 \text{ or } M \models \varphi_2 \\
M \models \neg\varphi & \text{iff } M \not\models \varphi
\end{array}
$$

Figure 2: Semantics of formulae

3 Stubborn Reduction for Weighted Petri Nets with Inhibitor Arcs

We shall now introduce a general idea of a reduction on an LTS that reduces the size of the state space and later instantiate it to the case of Petri nets with inhibitor arcs. A reduction can be seen as a filter that, for each state, specifies a subset of actions that are sufficient to explore in order to reach a state satisfying a given reachability formula.

Definition 2 (Reduced Transition Relation)**.** *Let $G = (\mathcal{S}, A, \rightarrow)$ be an LTS. A* reduction *of the transition system G is a function $St : \mathcal{S} \rightarrow 2^A$. A* reduced transition relation *is a relation $\xrightarrow[St]{} \subseteq \rightarrow$ such that $s \xrightarrow[St]{a} s'$ iff $s \xrightarrow{a} s'$ and $a \in St(s)$.*

Let $G = (\mathcal{S}, A, \rightarrow)$ be an LTS, $s \in \mathcal{S}$ a state, and St a reduction of G. Let $\overline{St(s)} = A \setminus St(s)$ be the set of all actions not in $St(s)$. We define the following property **W** of a reduction that guarantees that any action in $St(s)$ commutes with respect to actions from $\overline{St(s)}$.

W For all $s \in \mathcal{S}$, all $a \in St(s)$, and all $w \in \overline{St(s)}^*$, if $s \xrightarrow{wa} s'$ then $s \xrightarrow{aw} s'$.

Reductions that satisfy **W** are called *(weak) semistubborn* reductions [17]. In the rest of the paper, we say that $St(s)$ is the *stubborn set* of s and that an action $a \in St(s)$ is a *stubborn action* in s.

Let $G = (\mathcal{S}, A, \rightarrow)$ be an LTS and $G \subseteq \mathcal{S}$ a set of goal states. For a reduction St to preserve the reachability of a goal state, we define the following sufficient condition on St.

R For all $s \in \mathcal{S}$, if $s \notin G$ and $s \xrightarrow{w} s'$ where $w \in \overline{St(s)}^*$ then $s' \notin G$.

Condition **R** states that if we start in a non-goal state, the execution of non-stubborn transitions cannot reach any goal state in G. Hence it ensures that at least one stubborn action must be executed in order to reach a goal state. Any reduction that satisfies Conditions **W** and **R** also guarantees the preservation of reachability as stated by the following theorem.

Theorem 1 (Reachability Preservation)**.** *Let $G = (\mathcal{S}, A, \rightarrow)$ be an LTS and let $G \subseteq S$ be a set of goal states. Let St be a reduction of G satisfying **W** and **R** and let $s \in \mathcal{S}$. If $s \rightarrow^n s'$ for some $s' \in G$ then $s \xrightarrow[St]{}^m s''$ for some $s'' \in G$ where $m \leq n$.*

Proof. Let $w \in A^*$ be a transition sequence such that $s \xrightarrow{w} s'$ for some $s' \in G$. The proof proceeds by induction on the length of w. *Base case:* If $|w| = 0$ then $s = s' \in G$ and the claim trivially holds.
Inductive case: Let $|w| > 0$. If $s \in G$ then the claim immediately holds as in the base case. If $s \notin G$ then by **R** we then get that at least one transition in w must belong to $St(s)$, otherwise it is impossible that $s' \in G$. Hence we can divide w into vau where $v \in \overline{St(s)}^*$ and $a \in St(s)$. Condition **W** now implies the existence of s_a such that $s \xrightarrow{a} s_a \xrightarrow{vu} s'$. If $s_a \in G$, the length of the path from s_0 to s_a is less than or equal to $|w|$ and we are done. Otherwise, by the induction hypothesis we get that $s_a \xrightarrow[St]{}^m s''$ for some $s'' \in G$ where $m \leq |vu|$. This together with $s \xrightarrow[St]{a} s_a$ gives that $s \xrightarrow[St]{}^{m+1} s''$ where $s'' \in G$ such that $m + 1 \leq |w|$ as requested. □

In the following subsection, we shall instantiate this general framework to the case of Petri nets, taking a particular care to account for weights on arcs in order to construct the smallest possible stubborn sets.

Stubborn Set Reduction of Petri Nets

Let $N = (P, T, W, I)$ be a fixed Petri net and φ a reachability formula. We are interested in the question, whether we can reach from the initial

φ	$A_M(\varphi)$	$A_M(\neg\varphi)$
deadlock	$t \cup ({}^\bullet t)^- \cup {}^+({}^\circ t)$ for some $t \in en(M)$	$\emptyset$
t	${}^+p$ for some $p \in {}^\bullet t$ where $M(p) < w((p,t))$ or p^- for some $p \in {}^\circ t$ where $M(p) \geq I((p,t))$	$({}^\bullet t)^- \cup {}^+({}^\circ t)$
$e_1 < e_2$	$decr_M(e_1) \cup incr_M(e_2)$	$A_M(e_1 \geq e_2)$
$e_1 \leq e_2$	$decr_M(e_1) \cup incr_M(e_2)$	$A_M(e_1 > e_2)$
$e_1 > e_2$	$incr_M(e_1) \cup decr_M(e_2)$	$A_M(e_1 \leq e_2)$
$e_1 \geq e_2$	$incr_M(e_1) \cup decr_M(e_2)$	$A_M(e_1 < e_2)$
$e_1 = e_2$	$decr_M(e_1) \cup incr_M(e_2)$ if $eval_M(e_1) > eval_M(e_2)$ $incr_M(e_1) \cup decr_M(e_2)$ if $eval_M(e_1) < eval_M(e_2)$	$A_M(e_1 \neq e_2)$
$e_1 \neq e_2$	$incr_M(e_1) \cup decr_M(e_1) \cup incr_M(e_2) \cup decr_M(e_2)$	$A_M(e_1 = e_2)$
$\varphi_1 \wedge \varphi_2$	$A_M(\varphi_i)$ for some $i \in \{1,2\}$ where $M \not\models \varphi_i$	$A_M(\neg\varphi_1 \vee \neg\varphi_2)$
$\varphi_1 \vee \varphi_2$	$A_M(\varphi_1) \cup A_M(\varphi_2)$	$A_M(\neg\varphi_1 \wedge \neg\varphi_2)$

Table 1: Interesting transitions of φ (assuming $M \not\models \varphi$, otherwise $A_M(\varphi) = \emptyset$)

Expression e	$incr_M(e)$	$decr_M(e)$
c	$\emptyset$	$\emptyset$
p	${}^+p$	p^-
$e_1 + e_2$	$incr_M(e_1) \cup incr_M(e_2)$	$decr_M(e_1) \cup decr_M(e_2)$
$e_1 - e_2$	$incr_M(e_1) \cup decr_M(e_2)$	$decr_M(e_1) \cup incr_M(e_2)$
$e_1 \cdot e_2$	$incr_M(e_1) \cup decr_M(e_1) \cup$ $incr_M(e_2) \cup decr_M(e_2)$	$incr_M(e_1) \cup decr_M(e_1) \cup$ $incr_M(e_2) \cup decr_M(e_2)$

Table 2: Increasing and decreasing transitions of expression e

marking some of the goal markings from $G_\varphi = \{M \in \mathcal{M}(N) \mid M \models \varphi\}$. We first define the notion of *interesting transitions* $A_M(\varphi) \subseteq T$ for a marking M relative to φ such that whenever $M \xrightarrow{w} M'$ via the sequence of transitions $w = t_1 t_2 \ldots t_n \in T^*$ where $M \notin G_\varphi$ and $M' \in G_\varphi$, then there must exist i, $1 \leq i \leq n$, such that $t_i \in A_M(\varphi)$.

Table 1 gives the definition of $A_M(\varphi)$. The definition is at several places nondeterministic, allowing for a variety of sets of interesting transitions. Table 1 uses the functions $incr_M : E_N \to 2^T$ and $decr_M : E_N \to 2^T$ defined in Table 2, where E_N is the set of all arithmetic expressions that can be constructed for the net N. These functions, given an expression e, return all transitions that can possibly increase resp. decrease the evaluation of e. Formally, the required properties of the functions $incr_M(e)$ and $decr_M(e)$ are summarized in the next lemma.

Lemma 1. *Let e be an arithmetic expression, let $M, M' \in \mathcal{M}(N)$, and let $w = t_1 t_2 \ldots t_n \in T^*$ be such that $M \xrightarrow{w} M'$.*

- *If $eval_M(e) < eval_{M'}(e)$ then there is i, $1 \leq i \leq n$, s.t. $t_i \in$*

$incr_M(e)$.

- *If* $eval_M(e) > eval_{M'}(e)$ *then there is* i, $1 \leq i \leq n$, *s.t.* $t_i \in decr_M(e)$.

Proof. The proof follows from the definition of the functions by a straightforward structural induction on e. □

Let us by $\overline{A_M(\varphi)}$ denote the set $T \setminus A_M(\varphi)$ of non-interesting transitions. We can now formulate a lemma stating that at least one interesting transition must be executed before we can reach a goal marking.

Lemma 2. *Let* $N = (P, T, W, I)$ *be a Petri net,* $M \in \mathcal{M}(N)$ *a marking,* φ *a reachability formula, and* $w \in \overline{A_M(\varphi)}^*$ *a sequence of non-interesting transitions. If* $M \not\models \varphi$ *and* $M \xrightarrow{w} M'$ *then* $M' \not\models \varphi$.

Proof. Let $N = (P, T, W, I)$ be a Petri net, $M \in \mathcal{M}(N)$ be a marking, and φ a given formula. Assume that $M \not\models \varphi$. The proof proceeds by structural induction on φ.

$\varphi = deadlock$: If $M \not\models \varphi$ then there exists a $t \in en(M)$ s.t. $t \cup ({}^\bullet t)^- \cup {}^+({}^\circ t) \subseteq A_M(\varphi)$. To disable t we have to fire a transition $t' \in ({}^\bullet t)^- \cup {}^+({}^\circ t)$ to either remove or add tokens that will disable or inhibit t, respectively. Since $({}^\bullet t)^- \cup {}^+({}^\circ t) \subseteq A_M(\varphi)$ we have $M' \not\models \varphi$ as the selected transition t is still enabled.

$\varphi = t$: If $M \not\models \varphi$ then $t \notin en(M)$. Either there exists $p \in {}^\bullet t$ such that $M(p) < w(p,t)$ and we have to fire some transition from ${}^+p$ to enable t, or there exists $p \in {}^\circ t$ such that $M(p) \geq I(p,t)$ and we have to fire some transition from p^- to enable t. Since ${}^+p \subseteq A_M(\varphi)$ or $p^- \subseteq A_M(\varphi)$, we get that $M' \not\models \varphi$.

$\varphi = e_1 < e_2$: If $M \not\models \varphi$ then $eval_M(e_1) \geq eval_M(e_2)$. Since $decr_M(e_1) \cup incr_M(e_2) \subseteq A_M(\varphi)$ we know that $eval_M(e_1) \leq eval_{M'}(e_1)$ as well as $eval_M(e_2) \geq eval_{M'}(e_2)$ because of Lemma 1, and therefore $M' \not\models \varphi$.

$\varphi = e_1 > e_2$: If $M \not\models \varphi$ then $eval_M(e_1) \leq eval_M(e_2)$. Since $incr_M(e_1) \cup decr_M(e_2) \subseteq A_M(\varphi)$ we know that $eval_M(e_1) \geq eval_{M'}(e_1)$ as well as $eval_M(e_2) \leq eval_{M'}(e_2)$ because of Lemma 1, and therefore $M' \not\models \varphi$.

$\varphi = \varphi_1 \wedge \varphi_2$: If $M \not\models \varphi$ then there exists $i \in \{1, 2\}$ s.t. $M \not\models \varphi_i$. We know that $A_M(\varphi_i) \subseteq A_M(\varphi)$ which by the induction hypothesis implies $M' \not\models \varphi_i$. By the semantics of conjunction we also have that $M' \not\models \varphi$.

$\varphi = \varphi_1 \vee \varphi_2$: If $M \not\models \varphi$ then $M \not\models \varphi_1$ and $M \not\models \varphi_2$. We know that $A_M(\varphi_1) \cup A_M(\varphi_2) \subseteq A_M(\varphi)$ which by the induction hypothesis implies $M' \not\models \varphi_1$ and $M' \not\models \varphi_2$. By the semantics of disjunction we also have that $M' \not\models \varphi$.

$\varphi = \neg t$: If $M \not\models \varphi$ then $t \in en(M)$. To disable t we have to fire at least one transition from $({}^\bullet t)^- \cup {}^+({}^\circ t)$ to either remove or add tokens that will disable or inhibit t, respectively. Since $({}^\bullet t)^- \cup {}^+({}^\circ t) \subseteq A_M(\varphi)$ we have $M' \not\models \varphi$.

The remaining cases and negation cases are analogous. □

Lemma 2 allows us to satisfy Property **R** of Theorem 1 by including all interesting transitions in the stubborn set. Ensuring Property **W** is achieved by including further transitions according to the following theorem.

Theorem 2 (Reachability Preserving Closure)**.** *Let $N = (P, T, W, I)$ be a Petri net, φ a formula, and St a reduction of $G(N)$ such that for all $M \in \mathcal{M}(N)$ the following conditions hold.*

1 *We have $A_M(\varphi) \subseteq St(M)$.*

2 *For all $t \in St(M)$, if $t \notin en(M)$ then*

- *there is $p \in {}^\bullet t$ s.t. $M(p) < w(p, t)$ and ${}^+p \subseteq St(M)$, or*
- *there is $p \in {}^\circ t$ s.t. $M(p) \geq I(p, t)$ and $p^- \subseteq St(M)$.*

3 *For all $t \in St(M)$, if $t \in en(M)$ then*

- *for all $p \in {}^\bullet t$ where $t \in p^-$ we have $p^\bullet \subseteq St(M)$, and*
- *for all $p \in t^\bullet$ where $t \in {}^+p$ we have $p^\circ \subseteq St(M)$.*

*Then St satisfies **W** and **R**.*

Proof. From Condition 1 we know that $A_M(\varphi) \subseteq St(M)$ and hence Property **R** holds for St by Lemma 2. We will now argue that St satisfies Property **W**. Let $M \in \mathcal{M}(N)$ be a marking, $t \in T$ a transition such that

$t \in St(M)$, and $w \in \overline{St(M)}^*$ a transition sequence of non-stubborn transitions. We want to show that if $M \xrightarrow{wt} M'$ then also $M \xrightarrow{tw} M'$.

Let $M_w \in \mathcal{M}(N)$ be a marking such that $M \xrightarrow{w} M_w$. Let us assume for the sake of contradiction that $t \notin en(M)$. Then either (i) there exists $p \in {}^\bullet t$ such that $M(p) < w(p,t)$ or (ii) there is $p \in t^\circ$ where $M(p) \geq I(p,t)$. In case (i) we get by Condition 2 that all transitions that can increase the number of tokens in p are stubborn. Since $w \in \overline{St(M)}^*$ this implies that $M_w(p) < w(p,t)$ and $t \notin en(M_w)$, contradicting our assumption that $M_w \xrightarrow{t} M'$. In case (ii) we get by Condition 2 that all transitions that can decrease the number of tokens in p are stubborn. Since $w \in \overline{St(M)}^*$ this implies that also $M_w(p) \geq I(p,t)$ and $t \notin en(M_w)$, again contradicting our assumption that $M_w \xrightarrow{t} M'$. Therefore we can conclude that $t \in en(M)$.

Since $t \in en(M)$ there is $M_t \in \mathcal{M}(N)$ such that $M \xrightarrow{t} M_t$. We also have to show that $M_t \xrightarrow{w} M'$. For the sake of contradiction, assume that this is not the case. Then there must exist a transition t' that occurs in w and that became disabled because t was fired before the sequence w. There are two cases how this can happen: (i) either t decreased the number of tokens in a shared pre-place $p \in {}^\bullet t \cap {}^\bullet t'$ (ii) or t increased the number of tokens in a place $p \in t^\bullet \cap {}^\circ t'$. In case (i), due to Condition 3, we know that for all $p \in {}^\bullet t$ if $t \in p^-$ then $p^\bullet \subseteq St(M)$, implying that $t' \in St(M)$. Since $w \in \overline{St(M)}^*$ such a t' cannot exist in w. In case (ii), due to Condition 3, we know that for all $p \in t^\bullet$ if $t \in {}^+p$ then $p^\circ \subseteq St(M)$, implying that $t' \in St(M)$. Since $w \in \overline{St(M)}^*$ such a t' cannot exist in w either. Hence the sequence w is executable from M_t and because the firings of wt and tw from M both reach a unique marking, we conclude with $M \xrightarrow{tw} M'$ as requested. □

In Algorithm 3 we now provide, based on Theorem 2, a pseudocode for a construction of a reachability preserving stubborn set that satisfies **W** and **R** for a given marking M and a reachability formula φ.

Theorem 3. *Algorithm 3 terminates and computes a reduction St satisfying* ***W*** *and* ***R****.*

Proof. For the proof of termination, we first notice the while-loop invariant $X \cap Y = \emptyset$. At the end of each iteration of the while-loop, we move one transition from Y into X and this can happen only finitely many times (there are finitely many transitions) before the set Y becomes empty and the algorithm terminates. For the correctness argument, we notice

Algorithm 3: Construction of a reachability preserving stubborn set

input : Net $N = (P, T, W, I)$, $M \in \mathcal{M}(N)$ and a formula φ
output : Stubborn set $St(M)$ such that St satisfies **W** and **R**

```
1  X := ∅; Y := A_M(φ);
2  while Y ≠ ∅ do
3      pick any t ∈ Y;
4      if t ∉ en(M) then
5          if ∃p ∈ •t such that M(p) < w(p,t) then
6              pick any p ∈ •t such that M(p) < w(p,t);
7              Y := Y ∪ (⁺p \ X);
8          else
9              pick any p ∈ °t such that M(p) ≥ I(p,t);
10             Y := Y ∪ (p⁻ \ X);
11     else
12         foreach p ∈ •t do
13             if t ∈ p⁻ then
14                 Y := Y ∪ (p• \ X);
15         foreach p ∈ t• do
16             if t ∈ ⁺p then
17                 Y := Y ∪ (p° \ X);
18     Y := Y \ {t};
19     X := X ∪ {t};
20 return X;
```

that Algorithm 3 replicates exactly the closure operations described in Theorem 2 and guarantees that all conditions of Theorem 2 are met at the termination of the algorithm. □

Finally, we want to point out that there is nondeterminism in both generating the interesting set of transitions and applying the stubborn set closure. For the interesting set of transitions, this is whenever we resolve the deadlock or the fireability predicate, and in case of conjunction where none of the conjuncts hold in the given marking. For the stubborn set closure, there is a nondeterministic choice whenever we have to select either a disabling or inhibiting place for a disabled transition. As documented by practical examples, we often prefer to construct the smallest possible stubborn set in order to reduce the reachable state space (though this does not in general guarantee the smallest state space as demonstrated in [16]). In our implementation of Algorithm 3, we perform the nondeterministic choices such that they minimize the number of newly added transitions to the set Y. In particular, for the nondeterminism in lines 6 and 9 of Algorithm 3, we experienced that selecting places such that their preset is already included in the closure greatly improves performance. In general, we only apply a heuristic approach to resolve the nondeterministic choices as our experiments showed that the computational overhead of finding a minimal stubborn set can be significant.

4 Structural Reductions for Weighted Petri Nets with Inhibitor Arcs

After refining the stubborn set reduction for the case of weighted Petri nets with inhibitor arcs, we shall now focus also on adapting the structural reduction techniques for this class of nets. In what follows, we extend the standard structural reduction rules as presented e.g. in [10] so that they allow to more efficiently reduce weighted nets (with inhibitor arcs). We also add five additional reduction rules in order to further reduce the size of the input net.

Let $N = (P, T, W, I)$ be a fixed Petri net. We shall first notice that all formulae involving the fireability predicate $t \in T$ can be rewritten into an equivalent cardinality formula

$$p_0 \geq w(p_0, t) \wedge \cdots \wedge p_n \geq w(p_n, t) \wedge p'_0 < I(p'_0, t) \wedge \cdots \wedge p'_m < I(p'_m, t)$$

where ${}^\bullet t = \{p_0, \ldots, p_n\}$ and $t^\circ = \{p'_0, \ldots, p'_m\}$. It is thus sufficient in the remainder of this section to consider only cardinality formulae. We will, at the end of this section, also discuss the correctness of the presented rules in relation to the deadlock formulae.

The rules presented in this section assume that $p(\varphi)$ denotes the set of all places that occur in the cardinality formula φ such that

$$\begin{aligned} p(\mathit{true}) = p(\mathit{false}) = p(c) &= \emptyset \\ p(p) &= \{p\} \\ p(\neg\varphi) &= p(\varphi) \\ p(\varphi_1 \vee \varphi_2) = p(\varphi_1 \wedge \varphi_2) &= p(\varphi_1) \cup p(\varphi_2) \\ p(e_1 \bowtie e_2) = p(e_1 \oplus e_2) &= p(e_1) \cup p(e_2)\ . \end{aligned}$$

In each structural reduction rule, we fix some places and transitions that satisfy given preconditions and perform updates that can change the weight function and remove places and/or transitions (including all their connected arcs). Let us now give a general definition of correctness of a given rule X (where X is one of the rules A to I), stating that the reachability of a given cardinality formula φ is preserved.

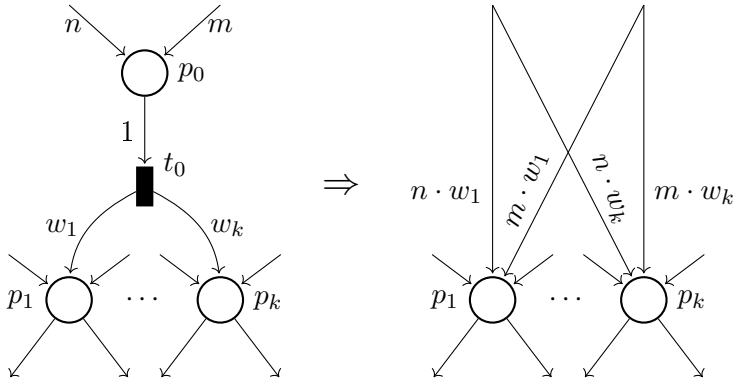
Definition 3 (Correctness of Rule X)**.** *Let $N = (P, T, W, I)$ be a Petri net and $M_0 \in \mathcal{M}(N)$ its initial marking. Let $N' = (P', T', w', I')$ and $M'_0 \in \mathcal{M}(N')$ be the modified N and M_0 after applying once Rule X for a cardinality formula φ. We say that Rule X is* correct *for a cardinality formula φ if there exists $M \in \mathcal{M}(N)$ s.t. $M_0 \to^* M$ and $M \models \varphi$ if and only if there exists $M' \in \mathcal{M}(N')$ s.t. $M'_0 \to^* M'$ and $M' \models \varphi$.*

Lemma 3. *Rule A in Figure 3 is correct for any cardinality formula φ.*

Proof. Assume a given net N, a marking M_0, and a cardinality formula φ. Let N' and M'_0 be the net and the initial marking after one application of Rule A. We shall argue that Rule A is correct. First, we define an equivalence relation $\equiv_A \subseteq \mathcal{M}(N) \times \mathcal{M}(N')$ such that $M \equiv_A M'$ if and only if

- $M'(p) = M(p)$ for all $p \in P \setminus \{p_0, p_1, \ldots, p_k\}$, and
- $M'(p) = M(p) + M(p_0) \cdot w(t_0, p)$ for all $p \in \{p_1, \ldots, p_k\}$.

Let us first realize that $M \models \varphi$ iff $M' \models \varphi$ whenever $M \equiv_A M'$. This follows from Precondition A4 and the definition of $\equiv_A$. Moreover, due to Update UA1 we also have $M_0 \equiv_A M'_0$. Our lemma then follows from the next two properties. Let $M \equiv_A M'$ then



Precondition	Update
Fix p_0, t_0 where $t_0^\bullet = \{p_1, \ldots, p_k\}$ s.t.	UA1) For all $p \in \{p_1, \ldots, p_k\}$ change the initial marking s.t. $M_0'(p) := M_0(p) + M_0(p_0) \cdot W(t_0, p)$
A1) ${}^\bullet t_0 = \{p_0\}$ and $w(p_0, t_0) = 1$.	
A2) $p_0^\bullet = \{t_0\}$ and $p_0 \notin \{p_1, \ldots, p_k\}$	UA2) For all $t \in T \setminus \{t_0\}$ and all $p \in \{p_1, \ldots, p_k\}$ set $W'(t, p) := W(t, p_0) \cdot W(t_0, p) + W(t, p)$
A3) $p_0^\circ = p_1^\circ = \ldots = p_k^\circ = {}^\circ t_0 = \emptyset$	
A4) $\{p_0, p_1, \ldots, p_k\} \cap p(\varphi) = \emptyset$	UA3) Remove p_0 and t_0.

Figure 3: Rule A: Sequential Transition Removal

P1) if $M \xrightarrow{t} M_1$ then either $M_1 \equiv_A M'$ or $M' \xrightarrow{t} M_1'$ s.t. $M_1 \equiv_A M_1'$, and

P2) if $M' \xrightarrow{t} M_1'$ then $M \xrightarrow{t_0^n t} M_1$ for some $n \in \mathbb{N}^0$ s.t. $M_1 \equiv_A M_1'$.

Let us first argue for Property P1. There are two cases.

- Case $t = t_0$: We want to show that $M_1 \equiv_A M'$. For all $p \in P \setminus \{p_0, p_1, \ldots, p_k\}$ we clearly have $M'(p) = M_1(p)$ as firing t_0 only changes the number of tokens in $p_0, p_1, \ldots, p_k$. By Precondition A1 we observe that firing of t_0 removes one token from p_0 and adds $w(t_0, p)$ tokens to p, for all $p \in \{p_1, \ldots, p_k\}$. Hence by our assumption that $M \equiv_A M'$ and the definition of $\equiv_A$ we have $M'(p) = M(p) + M(p_0) \cdot w(t_0, p) = M(p) + w(t_0, p) + (M(p_0) - 1) \cdot w(t_0, p) = M_1(p) + M_1(p_0) \cdot w(t_0, p)$ for all $p \in \{p_1, \ldots, p_k\}$. This means that

$M'(p) = M_1(p) + M_1(p_0) \cdot w(t_0, p)$ for all $p \in \{p_1, \ldots, p_k\}$, implying that $M_1 \equiv_A M'$ as required.

- Case $t \neq t_0$: We want to show that $M' \xrightarrow{t} M'_1$ such that $M_1 \equiv_A M'_1$. As $M \equiv_A M'$ implies that $M(p) \leq M'(p)$ for all $p \in \{p_1, \ldots, p_k\}$ then together with A2 and A3 we get that $t \in en(M')$ and we can fire it such that $M' \xrightarrow{t} M'_1$. For all $p \in P \setminus \{p_0, p_1, \ldots, p_k\}$ we can easily notice that $M_1(p) = M'_1(p)$. Once t is fired from M', we get by UA2 that for all $p \in \{p_1, \ldots, p_k\}$ we have $M'_1(p) = M'(p) + w'(t, p) = M'(p) + w(t, p_0) \cdot w(t_0, p) + w(t, p)$. Because $M \equiv_A M'$ we know that for all $p \in \{p_1, \ldots, p_k\}$ also $M'(p) = M(p) + M(p_0) \cdot W(t_0, p)$. By substituting this to the equation above, we get $M'_1(p) = M(p) + M(p_0) \cdot w(t_0, p) + w(t, p_0) \cdot w(t_0, p) + w(t, p) = M(p) + w(t, p) + (M(p_0) + w(t, p_0)) \cdot w(t_0, p) = M_1(p) + M_1(p_0) \cdot w(t_0, p)$ which implies that $M_1 \equiv_A M'_1$ as required.

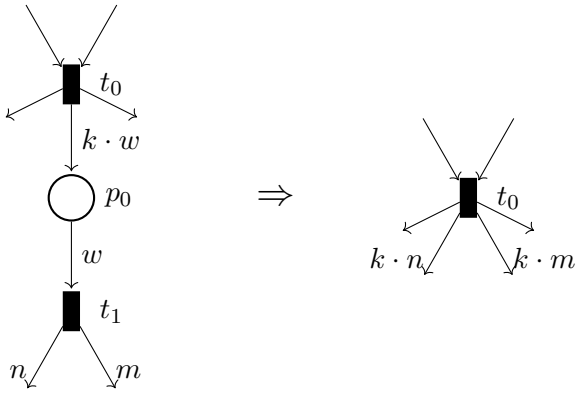
Finally, let us finish the proof by arguing for Property P2. Let $M \equiv_A M'$ and we want to show that if $M' \xrightarrow{t} M'_1$ then we can fire the transition t_0 from M several times followed by the transition t and reach a marking M_1 such that $M_1 \equiv_A M'_1$. Clearly $t \neq t_0$ as the transition t_0 was removed in N'. Due to Precondition A1 we can fire t_0 in the marking M exactly $M(p_0)$ times so that we reach a marking with no tokens in p_0 and $M(p_0) \cdot w(t_0, p)$ additional tokens in each place $p \in \{p_1, \ldots, p_k\}$, so that they now contains exactly $M'(p)$ tokens. Now t must be enabled as we assume that it is enabled in M' and after firing t, we reach a marking M_1 such that $M_1 \equiv_A M'_1$. □

Lemma 4. *Rule B in Figure 4 is correct for any cardinality formula φ.*

Proof. Assume a given net N, a marking M_0, and a cardinality formula φ. Let N' and M'_0 be the net and the initial marking after one application of Rule B. We shall argue that Rule B is correct. First, we define an equivalence relation $\equiv_B \subseteq \mathcal{M}(N) \times \mathcal{M}(N')$ such that $M \equiv_B M'$ if and only if $M'(p) = M(p) + \lfloor M(p_0)/w(p_0, t_1) \rfloor \cdot w(t_1, p)$ for all $p \in P \setminus \{p_0\}$.

Let us first realize that $M \models \varphi$ iff $M' \models \varphi$ whenever $M \equiv_B M'$. This follows from Precondition B4, B5, and the definition of $\equiv_B$. Moreover, due to Update UB1 we also have $M_0 \equiv_B M'_0$. Our lemma then follows from the next two properties. Let $M \equiv_B M'$ then

P1) if $M \xrightarrow{t} M_1$ then either $M_1 \equiv_B M'$ or $M' \xrightarrow{t} M'_1$ s.t. $M_1 \equiv_B M'_1$, and



Precondition	Update
Fix p_0 and t_0, t_1 where $t_0 \neq t_1$ s.t. B1) ${}^\bullet p_0 = \{t_0\}$, $p_0^\bullet = \{t_1\}$, ${}^\bullet t_1 = \{p_0\}$ B2) $w(t_0, p_0) = k \cdot w(p_0, t_1)$ for $k \geq 1$ B3) $p_0^\circ = {}^\circ t_0 = {}^\circ t_1 = \emptyset$ B4) $p_0 \notin p(\varphi)$ B5) $p^\circ = \emptyset$ and $p \notin p(\varphi)$ for all $p \in t_1^\bullet$	UB1) For all $p \in P \setminus \{p_0\}$ set $M_0'(p) = M_0(p) + \lfloor M_0(p_0)/w(p_0, t_1) \rfloor \cdot w(t_1, p)$ UB2) For all $p \in P \setminus \{p_0\}$ set $w'(t_0, p) = w(t_0, p) + k \cdot w(t_1, p)$ UB3) Remove p_0 and t_1.

Figure 4: Rule B: Sequential place removal

P2) if $M' \xrightarrow{t} M_1'$ then $M \xrightarrow{t_1^n t} M_1$ for some $n \in \mathbb{N}^0$ s.t. $M_1 \equiv_B M_1'$.

Let us first argue for Property P1. There are three cases.

- Case $t = t_1$: We want to show that $M_1 \equiv_B M'$. For all $p \in P \setminus (\{p_0\} \cup t_1^\bullet)$ we clearly have $M'(p) = M_1(p)$ as firing of t_1 only changes the number of tokens in p_0 and places in $t_1^\bullet$. By Preconditions B1 and B2 we notice that firing of t_1 removes $w(p_0, t_1)$ tokens from p_0 and adds $w(t_1, p)$ tokens to p, for all $p \in t_1^\bullet$. Is it now easy to observe that $M_1 \equiv_B M'$.

- Case $t = t_0$: We want to show that $M_1 \equiv_B M_1'$ where $M' \xrightarrow{t_0} M_1'$. Observe that in the reduced net we have that $w'(t, p) = w(t, p) +$

$k \cdot w(t_1, p)$ by Update UB2 for all $p \in t_1^\bullet$, which corresponds to also firing t_1 a total of k times. As before, we can see that $M_1 \equiv_B M_1'$.

- Case $t \in T \setminus \{t_1, t_0\}$: As $M \equiv_B M'$ we know that $M(p) \leq M'(p)$ for all $p \in t_1^\bullet$ and thanks to B5, we notice that whenever $M \xrightarrow{t} M_1$ then also $M' \xrightarrow{t} M_1'$ and clearly $M_1 \equiv_B M_1'$.

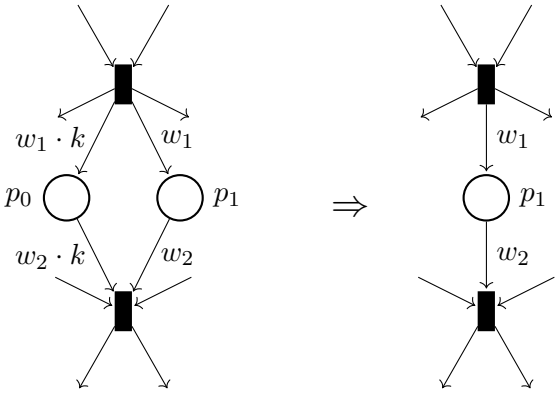
Let us finish the proof by arguing for Property P2. Let $M \equiv_B M'$ and we want to show that if $M' \xrightarrow{t} M_1'$ then we can fire from M the transition t_1 several times, followed by the transition t and reach a marking M_1 such that $M_1 \equiv_B M_1'$. Indeed, once we fire t_1 exactly $\lfloor M(p_0)/w(p_0, t_1) \rfloor$ times, the number of tokens in all places $p \in t_1^\bullet$ becomes equal to $M'(p)$ and thanks to B3 we can now fire t and reach a marking M_1 such that $M_1 \equiv_B M_1'$. □

Lemma 5. *Rule C in Figure 5 is correct for any cardinality formula φ.*

Proof. Assume a given net N, a marking M_0, and a cardinality formula φ. Let N' and M_0' be the net and the initial marking after one application of Rule C. We shall argue that Rule C is correct. Due to Precondition C3 we can see that $M(p_0) \geq M(p_1) \cdot k$ for every M such that $M_0 \to^* M$. Now any transition $t \in p_0^\bullet$ enabled from the marking M in the net with the place p_0 removed is also enabled in the original net and vice versa. Hence any marking M reachable from M_0 has a corresponding marking M' reachable from M_0' such that $M(p) = M'(p)$ for all $p \in P \setminus \{p_0\}$ and vice versa. By C1 and C2 we can now conclude that $M \models \varphi$ iff $M' \models \varphi$. □

Lemma 6. *Rule D in Figure 6 is correct for any cardinality formula φ.*

Proof. Assume a given net N, a marking M_0, and a cardinality formula φ. Let N' and M_0' be the net and the initial marking after one application of Rule D. We shall argue that Rule D is correct. Obviously, $M_0 = M_0'$ since Rule D does not change token counts in places. Moreover, from Precondition D2 it follows that the behaviour of t_0 can be replicated by firing t_1 exactly k times. Let $M_0 \xrightarrow{w} M$ such that $M \models \varphi$ and let w' be w with all occurrences of t_0 replaced by t_1^k. From the observation above and by D1 we get that in the net N' we have $M_0' \xrightarrow{w'} M'$ such that $M = M'$ and hence $M' \models \varphi$. The other direction is trivial as any behaviour of N' can be directly mimicked by N. □



Precondition	Update
Fix p_0, p_1 where $p_0 \neq p_1$ s.t. C1) $p_0^\circ = p_1^\circ = \emptyset$ C2) $p_0 \notin p(\varphi)$ C3) there is $k \geq 1$ such that (a) $M_0(p_0) \geq M_0(p_1) \cdot k$ (b) $W(t, p_0) \geq W(t, p_1) \cdot k$ for all $t \in {}^\bullet p_0$ (c) $W(p_0, t) \leq W(p_1, t) \cdot k$ for all $t \in p_0^\bullet$	UC1) Remove p_0.

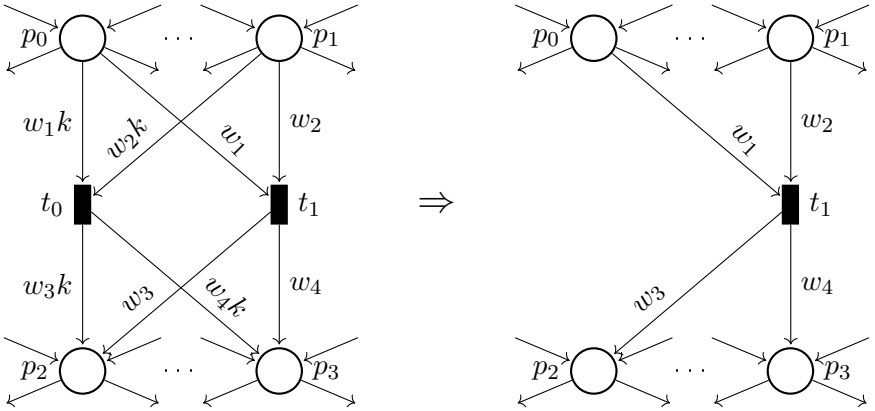
Figure 5: Rule C: Parallel place removal

Lemma 7. *Rule E in Figure 7 is correct for any cardinality formula φ.*

Proof. Observe that by Preconditions E1 and E2 the transition t_0 is never enabled in any reachable marking from M_0 as $M_0(p_0) < w(p_0, t_0)$ and ${}^+p_0 = \emptyset$. Hence removing t_0 does not the change behaviour of N. Moreover, should the place p_0 become isolated after the removal of t_0, it can be removed too by UE1, provided that it is not used in the formula φ and it is not connected to any inhibitor arc. □

Lemma 8. *Rule F in Figure 8 is correct for any cardinality formula φ.*

Proof. By F3 we know that ${p_0}^- = \emptyset$ and $M(p_0) \geq w(p_0, t)$ for all $t \in T$. Hence the number of tokens in p_0 can never drop and p_0 never disables



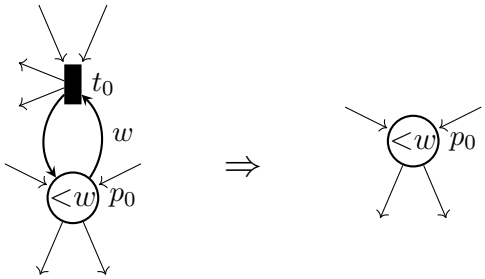
Precondition	Update
Fix t_0, t_1 where $t_0 \neq t_1$ s.t. D1) $^\circ t_0 = {}^\circ t_1 = \emptyset$ D2) there is $k \geq 1$ s.t. for all $p \in P$ (a) $W(p, t_0) = W(p, t_1) \cdot k$ (b) $W(t_0, p) = W(t_1, p) \cdot k$	UD1) Remove t_0.

Figure 6: Rule D: Parallel transition removal

any transition connected to it. Due to Precondition F1 and F2 it is so safe to remove the place p_0 without changing the behaviour of the net. □

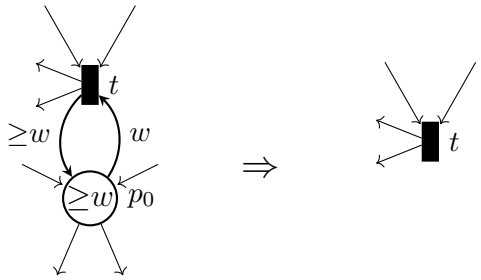
Lemma 9. *Rule G in Figure 9 is correct for any cardinality formula φ.*

Proof. Assume a given net N, a marking M_0, and a cardinality formula φ. Let N' and M_0' be the net and the initial marking after one application of Rule G. We shall argue that Rule G is correct. By G2 and G3 we get that $t_0 \notin {}^+p$ for all $p \in P$, and $t_0 \notin p^-$ for all $p \in p(\varphi)$. Hence the firing of t_0 does not change the number of tokens in any place that appears in φ and can only preserve or decrease the number of tokens in any other connected place. Let $M_0 \xrightarrow{w} M$ in N and let w' be w with all occurrences of t_0 removed. By G1 and the previous argument, we get that also $M_0' \xrightarrow{w'} M'$ and clearly $M(p) = M'(p)$ for all $p \in p(\varphi)$, implying



Precondition	Update
Fix p_0 and t_0 s.t. E1) $M_0(p_0) < w(p_0, t_0)$ E2) $w(t, p_0) \leq w(p_0, t)$ or $M_0(p_0) < w(p_0, t)$ for all $t \in T$	UE1) If $p_0^\bullet = \{t_0\}$, $p_0^\circ = \emptyset$ and $p_0 \notin p(\varphi)$ then remove p_0. UE2) Remove t_0.

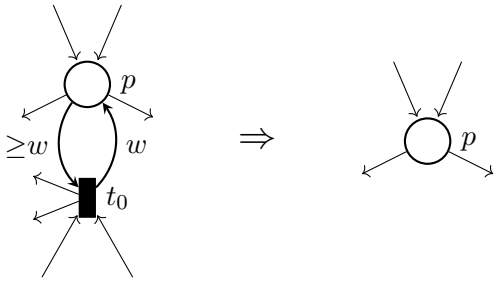
Figure 7: Rule E: Dead transition removal



Precondition	Update
Fix p_0 s.t. F1) $p_0^\circ = \emptyset$ F2) $p_0 \notin p(\varphi)$ F3) $w(t, p_0) \geq w(p_0, t)$ and $M_0(p_0) \geq w(p_0, t)$ for all $t \in T$	UF1) Remove p_0.

Figure 8: Rule F: Redundant place removal

that $M \models \varphi$ iff $M' \models \varphi$. The other direction where $M_0' \rightarrow^* M'$ in N' is trivial as the same marking M' can be reached also from M_0 in N. $\square$



Precondition	Update
Fix t_0 s.t. G1) $^\circ t_0 = \emptyset$ and $p^\circ = \emptyset$ for all $p \in {}^\bullet t_0$ G2) $t_0^\bullet \subseteq {}^\bullet t_0$ G3) for all $p \in {}^\bullet t_0$ we have either • $W(p, t_0) = W(t_0, p)$, or • $W(p, t_0) > W(t_0, p)$ and $p \notin p(\varphi)$	UG1) Remove t_0.

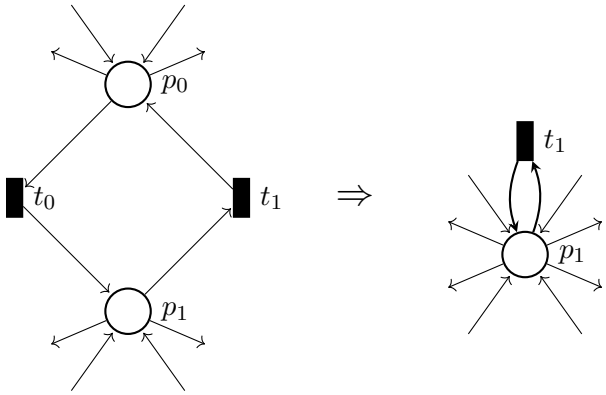
Figure 9: Rule G: Redundant transition removal

Lemma 10. *Rule H in Figure 10 is correct for any cardinality formula φ.*

Proof. Assume a given net N, a marking M_0, and a cardinality formula φ. Let N' and M_0' be the net and the initial marking after one application of Rule H. We shall argue that Rule H is correct. Let us define $\equiv_H \subseteq \mathcal{M}(N) \times \mathcal{M}(N')$ such that $M \equiv_H M'$ if and only if $M(p) = M'(p)$ for all $p \in P \setminus \{p_0, p_1\}$ and $M'(p_1) = M(p_1) + M(p_0)$. By Precondition H4 we know that $M \models \varphi$ iff $M' \models \varphi$ whenever $M \equiv_H M'$.

Let $M_0 \xrightarrow{w} M$ such that $M \models \varphi$. Let w' be w with all occurrences of t_0 removed. Due to the construction of the updated net and UH4, together with Precondition H3 we know that also $M_0' \xrightarrow{w'} M'$ such that $M \equiv_H M'$, giving us $M' \models \varphi$.

For the other direction, let $M_0' \xrightarrow{w'} M'$ such that $M' \models \varphi$. We want to find a sequence w such that $M_0 \xrightarrow{w} M$ and $M \equiv_H M'$, which implies that $M \models \varphi$. We prove this by induction on the length of w'. The base case follows from UH4. Let $M_0' \xrightarrow{w'} M' \xrightarrow{t} M_1'$ and assume by the induction hypothesis that $M_0 \xrightarrow{w} M$ for some w such that $M \equiv_H M'$. Clearly, if t



Precondition	Update
Fix different p_0, p_1, t_0, t_1 s.t. H1) $^\bullet t_0 = t_1^\bullet = \{p_0\}$ H2) $^\bullet t_1 = t_0^\bullet = \{p_1\}$ H3) $p_0^\circ = p_1^\circ = {}^\circ t_0 = {}^\circ t_1 = \emptyset$ H4) $p_0 \notin p(\varphi)$, $p_1 \notin p(\varphi)$ H5) $w(p_0, t_0) = w(t_0, p_1) = w(p_1, t_1) = w(t_1, p_0) = 1$	UH1) $w'(t, p_1) = w(t, p_1) + w(t, p_0)$ for all $t \in T$ UH2) $w'(p_1, t) = w(p_1, t) + w(p_0, t)$ for all $t \in T$ UH3) $w'(t_1, p_1) = w'(p_1, t_1) = 1$ UH4) $M_0'(p_1) = M_0(p_1) + M_0(p_0)$ UH5) Remove t_0. UH6) Remove p_0.

Figure 10: Rule H: Simple cycle removal

is enabled from M then we let $M \xrightarrow{t} M_1$ and reach a marking M_1 such that $M_1 \equiv_H M_1'$. If t is not enabled from M then we can rearrange the tokens in the places p_0 and p_1 by firing the transitions t_0 and t_1 so that we reach a marking where t becomes enabled (this is possible due to the construction of the net N'). After firing t we get a marking M_1 such that $M_1 \equiv_H M_1'$ and we are done with the inductive argument. □

Finally, we present a slightly different structural reduction rule that in one run computes a set of places and transitions that are safe to remove for the validity of a given cardinality formula φ. This Rule I is given in Algorithm 4.

Lemma 11. *Rule I in Algorithm 4 is correct for any cardinality formula φ.*

Algorithm 4: Rule I: Removal of irrelevant places and transitions

input : A net $N = (P, T, W, I)$, initial marking M_0, and a cardinality formula φ.

output : A reduced net N' and its initial marking M'_0.

1 $X := \emptyset$; $Y := \bigcup_{p \in p(\varphi)} p^- \cup {}^+p$;
2 **while** $Y \neq \emptyset$ **do**
3 pick any $t \in Y$;
4 $Y := Y \cup ({}^+({}^\bullet t) \setminus X)$;
5 $Y := Y \cup ((^\circ t)^- \setminus X)$;
6 $Y := Y \setminus \{t\}$;
7 $X := X \cup \{t\}$;
8 Let $P' = {}^\bullet X \cup {}^\circ X \cup p(\varphi)$.
9 Let $T' = X$.
10 Modify N by removing all places $P \setminus P'$ and all transitions $T \setminus T'$.
11 Let N' be the modified net and let $M'_0(p) = M_0(p)$ for all $p \in P'$.
12 **return** N' *and* M'_0

Proof. Assume a given net N, a marking M_0, and a cardinality formula φ. Let N' and M'_0 be the net and the initial marking after the application of Rule I (Algorithm 4). Let P' and T' be the set of places and transitions in the net N', respectively. We shall argue that Rule I is correct. Let us first define a relation $\equiv_I \subseteq \mathcal{M}(N) \times \mathcal{M}(N')$ such that $M \equiv_I M'$ if and only if

- $M(p) = M'(p)$ for all $p \in p(\varphi)$,
- $M(p) \leq M'(p)$ for all $p \in {}^\bullet T'$, and
- $M(p) \geq M'(p)$ for all $p \in {}^\circ T'$.

Let $M \equiv_I M'$. Then clearly $M \models \varphi$ iff $M' \models \varphi$ due to the definition of $\equiv_I$ and the fact that $p(\varphi) \subseteq P'$. Moreover, trivially also $M_0 \equiv_I M'_0$.

Let $M \equiv_I M'$. We will show that if $M \xrightarrow{t} M_1$ then either $M' \xrightarrow{t} M'_1$ such that $M_1 \equiv_I M'_1$, or $M_1 \equiv_I M'$. There are two cases to consider.

- Case $t \in T'$. Due to the second and third condition in the definition of $\equiv_I$, we know that t is also enabled in the marking M' and we

can fire $M' \xrightarrow{t} M'_1$. After firing t both from M and M' we clearly preserve all three conditions of $\equiv_I$ and $M_1 \equiv_I M'_1$.

- Case $t \notin T'$. We want to argue that $M_1 \equiv_I M'$. We notice that Algorithm 4 returns a net where ${}^+({}^\bullet T') \subseteq T'$ and $({}^\circ T')^- \subseteq T'$. Hence firing of $t \notin T'$ in the net N cannot increase the number of tokens in any place from ${}^\bullet T'$ and cannot decrease the number of tokens in any place from ${}^\circ T'$. Moreover, the firing of t cannot change the number of tokens in $p(\varphi)$ as $p^- \cup {}^+p \subseteq T'$ for every $p \in p(\varphi)$. As a result, all three conditions of definition $\equiv_I$ are met and we can conclude that $M_1 \equiv_I M'$.

For the other direction, we notice that N' is a subnet of N. Hence whenever $M'_0 \xrightarrow{w} M'$ then also $M_0 \xrightarrow{w} M$ such that $M \equiv_I M'$. □

We can now summarize the correctness of the structural rules for any given cardinality formula in the following theorem. Moreover, we also notice that all rules, except for G and I, preserve also the presence of a reachable deadlock marking. An application of Rule G can create a deadlock in the modified net and Rule I can both create new deadlocks as well as remove existing deadlocks.

Theorem 4. *Rules A to I are correct for any given cardinality formulae. Rules A to F and H are moreover correct also for the deadlock formula.*

5 Experimental Evaluation

We implemented the stubborn set reduction and structural reduction rules in the verification engine `verifypn` [6] (source code is available at `code.launchpad.net/~verifypn-cpn/verifypn/struct_vs_stub`) as a part of the model checking tool TAPAAL [3]. Our experiments are executed using the database of Petri net models and reachability queries from MCC'17 [8]. For each model (there are in total 438 Petri nets, including the known and surprise nets) there are three categories of queries: reachability cardinality (RC), reachability fireability (RF) and reachability deadlock (RD). In the RC and RF category there are 16 queries for each model and in RD there is only a single query. In some tables, we further subdivide the queries into RC+, RF+ and RD+ whenever there exists evidence (finite trace) proving the property and into RC–, RF–

		Number of solved queries			
	Queries	**Base**	**Stub**	**Struct**	**StubStruct**
RC	7008	3733	4807	4794	5325
RF	7008	4864	5503	5403	5820
RD	438	288	344	333	367
Total	14454	8885	10654	10530	11512

Table 3: Number of queries solved by each algorithm

and RD– where the whole state space must be explored before the validity of the property can be established. All of the above experiments were run on AMD Opteron 6376 processors with a 14 GB memory limit.

Comparison of Stubborn vs. Structural Reduction

In Table 3 we can see the number of queries solved by each algorithm (we set a 20 minute timeout for RC and RF, and a 1 hour timeout for RD). Here **Base** stands for the standard TAPAAL without stubborn and structural reductions, **Stub** adds to **Base** the stubborn set reduction, and **Struct** adds to **Base** the structural reductions. Finally, **StubStruct** stands for the engine that first applies structural reductions while preprocessing the nets and then uses the stubborn set reduction during the state-space search. In all cases, we use the heuristic search strategy implemented in TAPAAL. The practical applicability of both the stubborn and structural reduction is clear as each method independently allows to solve more than 1600 supplementary queries. However, more interestingly, the combination of both methods solves 858 additional queries compared to **Stub** and 982 additional queries compared to **Struct**. This demonstrates, even though the two methods both reduce the concurrency present in the model, they are not conflicting with each other and it is beneficial to apply them both during model checking.

A detailed pairwise comparison of the methods is presented in Table 4. For each query, an algorithm gets a point relative to another algorithm, as follows. **Exclusive:** answers the query while the opponent algorithm does not provide any answer. **Time:** answers the query at least 50% faster, disregarding queries that are solved in less than 10 seconds by both algorithms. **Memory:** answers the query by using at least 50% less peak memory. If an algorithm solves a query exclusively, it also gets a point in the time and memory comparison. As already discussed, the addition of stubborn and structural reduction significantly

Base vs Stub						
	exclusive		time		memory	
RC+	33	336	97	553	52	627
RC–	5	776	6	901	5	913
RF+	16	490	78	867	41	948
RF–	5	170	7	248	5	256
RD+	2	4	8	12	6	11
RD–	0	52	0	68	0	69
SUM	61	1828	196	2649	109	2824

(a)

Base vs Struct						
	exclusive		time		memory	
RC+	11	304	97	578	40	571
RC–	0	768	0	907	0	904
RF+	12	367	133	739	37	723
RF–	0	184	0	286	0	279
RD+	0	1	8	5	0	7
RD–	0	42	0	52	0	53
SUM	23	1666	238	2567	77	2537

(b)

Base vs StubStruct						
	exclusive		time		memory	
RC+	13	470	96	797	43	854
RC–	5	1140	6	1308	5	1318
RF+	19	657	142	1122	52	1195
RF–	4	322	5	453	4	454
RD+	2	4	12	9	6	13
RD–	0	77	0	93	0	94
SUM	43	2670	261	3782	110	3928

(c)

Stub vs Struct						
	exclusive		time		memory	
RC+	149	139	325	342	346	264
RC–	266	263	336	330	346	315
RF+	256	137	537	332	527	263
RF–	105	124	149	180	163	166
RD+	4	3	16	9	7	7
RD–	25	15	37	25	39	25
SUM	805	681	1400	1218	1428	1040

(d)

Stub vs StubStruct						
	exclusive		time		memory	
RC+	10	164	91	392	38	322
RC–	0	364	35	460	0	473
RF+	18	182	151	431	47	382
RF–	0	153	0	229	0	232
RD+	0	0	8	5	0	2
RD–	0	25	0	36	0	37
SUM	28	888	285	1553	85	1448

(e)

Struct vs StubStruct						
	exclusive		time		memory	
RC+	21	185	42	306	39	402
RC–	7	374	22	528	7	570
RF+	29	312	54	503	58	611
RF–	10	144	13	219	10	254
RD+	3	4	6	7	7	8
RD–	0	35	0	49	0	51
SUM	70	1054	137	1612	121	1896

(f)

Table 4: Algorithms comparison (7008 queries in RC and RF and 438 queries in RD)

improves the verification of **Base** as indicated in Table 4a and 4b. In both cases, **Base** still provides some exclusive answers. For **Stub** this is the case as there are nets where the stubborn sets include almost all enabled transitions, meaning that the construction of stubborn sets leaves us only with an overhead. In the case of **Struct** there are only a few exclusive answers and only for the cases where there exists a witness trace. Due to the changed structure of the net after the reduction, the search strategy gets modified and in 23 cases **Base** was lucky to find the witness trace faster even though the state space is larger. Clearly, the combination of stubborn and structural reduction computes the largest number of exclusive answers as shown in Figure 4c. The comparison of **Stub** and **Struct** in Table 4d shows a slightly higher number of exclusive answers when only stubborn set reduction is used, which is reflected also by the points for the time and memory comparison. The advantage of

	Rule applications (×1000)									% Reduction
Disabled Rule	A	B	C	D	E	F	G	H	I	Trans+Places
A	0	5557	115	10760	121	705	5294	598	8	38.9686
B	6613	0	181	11279	120	812	6086	624	11	42.9040
C	6594	1	0	11279	120	757	5957	624	10	42.4008
D	6318	1	129	0	120	733	5712	669	10	30.3359
E	6610	1	180	11279	0	825	6082	624	11	41.9651
F	6503	1	674	10727	250	0	6012	622	11	41.9089
G	6471	1	36	14879	104	664	0	636	15	39.9450
H	6598	1	129	11212	120	757	2824	0	6	37.2231
I	7213	1	199	11712	611	1648	8755	631	0	40.3852
None	6613	1	181	11279	120	812	6074	624	11	42.9053

Table 5: Number of applications of reductions rules (10 minutes timeout)

the combination of both methods, compared to an independent use of each one, is documented by a high number of new exclusive answers in Tables 4f and 4e. Some exclusive answers are lost when combining both techniques, but as this is the case mainly for the queries with a witness trace. As before, we contribute this to the modified search strategy after combining the two methods.

The reason why the combination of the two techniques is indeed beneficial seems to be twofold: (i) in preprocessing a net by applying first the structural reduction, the size of the net usually decreases considerably and this implies less overhead and fewer dependencies when (on-the-fly) computing the stubborn sets, on the other hand (ii) structural reductions remove only the behaviour that is detectable statically (without the knowledge of the actual marking) whereas stubborn reduction computes the pruning of the state-space dynamically by considering also the given marking that we are exploring.

We also remark that our refinement of the stubborn set method via the increasing/decreasing presets/postsets of places demonstrates an average reduction in running time by 13% (measured without the employment of other reduction techniques). On some nets there is not any noticeable improvement while e.g. on the model RAFT-PT we achieve a speedup of 98% (average over all instances of the model).

Comparison of Different Structural Reduction Rules

As we provided a number of new or extended rules for structural reduction, we investigate their potential applicability in Table 5 across the whole database of models from MCC'17. In the last row of the table,

we show the total number of times (in thousands) each rule was applied across all models in all categories. We also disable the application of each single rule and investigate how it influences the applicability of the remaining rules. The most obvious dependency is between rules A and B that can to a large degree substitute each other, in particular in the situations where a net contains a longer sequential chain of transition firings. As in our implementation rule A is applied (as long as possible) before we proceed to reduce by rule B, less than one thousand applications of rule B are observed. However, if rule A is disabled (the first row in the table) then rule B is applied approximately 5557.000 times. Another dependency is between the rules G and H. As seen the table, if rule H is disabled then the applicability of rule G drops considerably to 2824 thousands of applications, caused by the fact that rule H is creating new transition loops that rule G can remove (provided that the preconditions are satisfied). Otherwise the remaining rules are frequently used with rules D and G being the most applicable ones. The only exception is rule I. Due to the different nature of this rule, we achieve a considerable reduction effect on several nets but in general as soon as rule I is applied once on a given net and query, it is unlikely that more than a few further applications become possible.

Finally, we run an experiment with different orders in which the reduction rules are applied. We enumerated all possible rule permutations and executed them on all models and a selected reachability cardinality query. In one day, 252 nets completed the reductions under all given permutations. In 228 nets the size of the reduced net did not depend on the order of application of the reduction rules and only in 24 nets there was a smaller difference in the size of the reduced net. The reduction order used in our experiments was not the optimal one in only 10 cases.

Comparison with LoLA

Finally, we also compare the performance of our tool with LoLA [21], the winner of the MCC'17 competition in the reachability category. We use the current development snapshots of LoLA (based on version 2.0) and Sara (based on version 1.14)—in MCC'17 LoLA was running in parallel with Sara. Here we use the same rules as above for awarding points, but we run two parallel processes (three in the RD-category) for each tool. For RC and RF we run LoLA using (i) `-stateequation=alone` which calls the tool Sara and (ii) `-stateequation=none` which calls the standard engine of LoLA. For RD we run LoLA using (i) `-symmetry`

		Solved queries	
	Queries	TAPAAL	LoLA
RC	7008	6640	6568
RF	7008	6376	6321
RD	438	377	364
Total	14455	13392	13253

Table 6: Comparison between TAPAAL and LoLA

TAPAAL vs LoLA						
	exclusive		**time**		**memory**	
RC+	51	202	150	647	221	693
RC–	235	12	554	47	691	65
RF+	98	266	159	1346	381	826
RF–	207	19	314	150	362	47
RD+	4	20	43	29	39	53
RD–	30	1	48	9	62	7
SUM	625	520	1268	2228	1756	1691

Table 7: Pairwise score comparison between TAPAAL and LoLA

`-symmtimelimit=300 -stubborn=tarjan`, with (ii) `-symmetry -symmtimelimit=300 -findpath=alone` and with (iii) `-symmetry -symmtimelimit=300 -siphondepth=10 -siphontrap=alone` as suggested by LoLA developers as the recommended strategy for the tool. For our tool in the RC and RF categories we use in parallel (i) the default options and (ii) `-tar` enabling a trace abstraction refinement method based on [2]. For RD we run our tool in parallel using (i) the default options, (ii) `-tar` and (iii) `-siphon-trap 3600 -siphon-depth 10`. We terminate the parallel computation as soon as the fastest thread finishes its computation.

Table 6 proves that after implementing the stubborn and structural reductions in TAPAAL, we can solve a higher number of reachability queries than the tool LoLA. More precisely we can answer 139 additional queries over all three categories. A detailed comparison performed in Table 7 reveals that while LoLA is better in answering queries that have a witness trace, TAPAAL achieves a significant margin on the queries where the whole state space must be searched. We contribute this to improved and extended structural reduction rules suggested in this paper.

Moreover, LoLA runs in parallel the tool Sara that uses advanced state equations techniques instead of the explicit state space search, resulting in about 1000 extra points where LoLA solved the query faster. On the other hand, TAPAAL is showing a slightly better performance in terms of memory usage, likely due to the employment of the PTrie [7] data structure for storing the explored state space.

6 Conclusion

We described the stubborn set and structural reduction techniques for the use on reachability queries on weighted Petri nets with inhibitor arcs. The stubborn set reduction was first presented on general labelled transition systems and then specialized for the application on Petri nets. We extended the technique to deal with inhibitor arcs as well as refining its performance to take weighted arcs into account. Similarly, we extended some of the classical structural reduction rules to nets with weighted arcs and inhibitor arcs and suggested a number of additional reductions rules, while demonstrating their applicability on the nets from the annual model checking contest. Both techniques were proved correct and experimentally evaluated.

Our main conclusion is—while it may intuitively seem as contra productive to employ both stubborn and structural reductions at the same time as they both target similar phenomena in order to restrict concurrency—that the combination of the techniques is clearly beneficial and the possible overhead when computing the reductions pays off. As a result, we have an efficient implementation of our verification engine that is now part of the open source model checker TAPAAL, and our engine, in all three reachability subcategories, solves more queries than LoLA, the last year winner of the model checking contest.

In our future work, we plan to add a support for colored Petri nets and extend the techniques discussed in this paper so that they can be applied directly on colored nets before their unfolding into P/T nets.

Acknowledgements We would like to thank Karsten Wolf and Torsten Liebke from Rostock University for providing us with the development snapshot of the latest version of LoLA and Sara and for their help with setting up their tool and answering our questions. The work was funded by the center IDEA4CPS, Innovation Fund Denmark center DiCyPS and ERC Advanced Grant LASSO. The last author is

partially affiliated with FI MU, Brno.

References

[1] C. Baier and J.P. Katoen. *Principles of Model Checking*. The MIT Press, 2009. ISBN: 026202649X, 9780262026499.

[2] F. Cassez, P.G. Jensen, and K.G. Larsen. "Refinement of Trace Abstraction for Real-Time Programs". In: *Reachability Problems*. Vol. 10506. LNCS. Springer International Publishing, 2017, pp. 42–58. DOI: 10.1007/978-3-319-67089-8_4.

[3] A. David et al. "TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 7214. LNCS. Springer Berlin Heidelberg, 2012, pp. 492–497. DOI: 10.1007/978-3-642-28756-5_36.

[4] J. Esparza. "Decidability and Complexity of Petri Net Problems — An Introduction". In: *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*. Vol. 1491. LNCS. Springer Berlin Heidelberg, 1998, pp. 374–428. DOI: 10.1007/3-540-65306-6_20.

[5] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*. Vol. 1032. LNCS. Springer Berlin Heidelberg, 1996. ISBN: 978-3-540-60761-8.

[6] J.F. Jensen et al. "TAPAAL and Reachability Analysis of P/T Nets". In: *Transactions on Petri Nets and Other Models of Concurrency XI*. Vol. 9930. LNCS. Springer Berlin Heidelberg, 2016, pp. 307–318. DOI: 10.1007/978-3-662-53401-4_16.

[7] P.G. Jensen, K.G. Larsen, and J. Srba. "PTrie: Data Structure for Compressing and Storing Sets via Prefix Sharing". In: *Theoretical Aspects of Computing*. Vol. 7214. 10580. Springer International Publishing, 2017, pp. 248–265. DOI: 10.1007/978-3-319-67729-3_15.

[8] F. Kordon et al. *Complete Results for the 2017 Edition of the Model Checking Contest*. http://mcc.lip6.fr/2017/results.php.

[9] L.M. Kristensen, K. Schmidt, and A. Valmari. "Question-Guided Stubborn Set Methods for State Properties". In: *Formal Methods in System Design* 29.3 (2006). Springer, pp. 215–251. DOI: 10.1007/s10703-006-0006-1.

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

[11] T. Murata and J. Koh. "Reduction and Expansion of Live and Safe Marked Graphs". In: *IEEE Transactions on Circuits and Systems* 27.1 (1980). IEEE, pp. 68–71. DOI: 10.1109/TCS.1980.1084711.

[12] C.A Petri. "Kommunikation mit Automaten". In: *Bonn, Institut für Instrumentelle Mathematik* (1962).

[13] K. Schmidt. "Integrating Low Level Symmetries into Reachability Analysis". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 5404. LNCS. Springer Berlin Heidelberg, 2000, pp. 315–330. DOI: 10.1007/3-540-46419-0_22.

[14] K. Schmidt. "Stubborn Sets for Standard Properties". In: *Application and Theory of Petri Nets*. Vol. 1639. LNCS. Springer Berlin Heidelberg, 1999, pp. 46–65. DOI: 10.1007/3-540-48745-X_4.

[15] A. Valmari. "A Stubborn Attack on State Explosion". In: *Formal Methods in System Design* 1.4 (1992). Springer, pp. 297–322. DOI: 10.1007/BF00709154.

[16] A. Valmari. "Stubborn Set Intuition Explained". In: *Transactions on Petri Nets and Other Models of Concurrency XII*. Vol. 10470. LNCS. Springer Berlin Heidelberg, 2017, pp. 140–165. DOI: 10.1007/978-3-662-55862-1_7.

[17] A. Valmari. "Stubborn Sets for Reduced State Space Generation". In: *Advances in Petri Nets 1990*. Vol. 483. LNCS. Springer Berlin Heidelberg, 1991, pp. 491–515. DOI: 10.1007/3-540-53863-1_36.

[18] A. Valmari, M. Hague, and I. Potapov. "Stubborn Sets with Frozen Actions". In: *Reachability Problems*. Vol. 7454. LNCS. Springer International Publishing, 2017, pp. 160–175. DOI: 10.1007/978-3-319-67089-8_12.

[19] A. Valmari and W. Vogler. "Fair Testing and Stubborn Sets". In: *Model Checking Software*. Vol. 9641. LNCS. Springer International Publishing, 2016, pp. 225–243. DOI: 10.1007/978-3-319-32582-8_16.

[20] H. Wimmel and K. Wolf. "Applying CEGAR to the Petri Net State Equation". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 6605. LNCS. Springer Berlin Heidelberg, 2011, pp. 224–238. DOI: 10.1007/978-3-642-19835-9_19.

[21] K. Wolf and M. Koutny. "Running LoLA 2.0 in a Model Checking Competition". In: *Transactions on Petri Nets and Other Models of Concurrency XI*. Vol. 9930. LNCS. Springer Berlin Heidelberg, 2016, pp. 274–285. DOI: 10.1007/978-3-662-53401-4_13.

Paper E

Simplification of CTL Formulae for Efficient Model Checking of Petri Nets

Frederik M. Bønneland, Jakob Dyhr, Peter G. Jensen,
Mads Johannsen, and Jiří Srba

This paper has been published in:
Application and Theory of Petri Nets and Concurrency, LNCS Vol. 10877, pp 143-163, 2018.

Abstract

We study techniques to overcome the state space explosion problem in CTL model checking of Petri nets. Classical state space pruning approaches like partial order reductions and structural reductions become less efficient with the growing size of the CTL formula. The reason is that the more places and transitions are used as atomic propositions in a given formula, the more of the behaviour (interleaving) becomes relevant for the validity of the formula. We suggest several methods to reduce the size of CTL formulae, while preserving their validity. By these methods, we significantly increase the benefits of structural and partial order reductions, as the combination of our techniques can achive up to 60 percent average reduction in formulae sizes. The algorithms are implemented in the open-source verification tool TAPAAL and we document the efficiency of our approach on a large benchmark of Petri net models and queries from the Model Checking Contest 2017.

1 Introduction

Model checking [6] of distributed systems, described in high-level formalisms like Petri nets, is often a time and resource consuming task—attributed mainly to the state space explosion problem. Several techniques like partial order and symmetry reductions [16, 21, 19, 23, 22] and structural reductions [14, 18, 17] were suggested for reducing the size of the state space of a given Petri net in need of exploration to verify different logical specifications. These techniques try to prune the searchable state space and their efficiency is to a high degree influenced by the type and size of the logical formula in question. The larger the formula is and the more atomic propositions (querying the number of tokens in places or the fireability of certain transitions) it has, the less can be pruned away when exploring the state space and hence the effect of these techniques is reduced. It is therefore desirable to design techniques that can reduce the size of a given logical formula, while preserving the model checking answer. For practical applicability, it is important that such formula reduction techniques are computationally less demanding than the actual state space search.

In this paper, we focus on the well-known logic CTL [5] and describe three methods for CTL formula simplification, each preserving the logical equivalence w.r.t. a the given Petri net model. The first two methods

rely on standard logical equivalences of formulae, while the third one uses state equations of Petri nets and linear programming to recursively traverse the structure of a given CTL formula. During this process, we identify subformulae that are either trivially satisfied or impossible to satisfy, and we replace them with easier to verify alternatives. We provide an algorithm for performing such a formula simplification, including the traversal though temporal CTL operators, and prove the correctness of our approach.

The formula simplification methods are implemented and fully integrated into an open-source model checker TAPAAL [10] and its untimed verification engine `verifypn` [14]. We document the performance of our tool on the large benchmark of Petri net models and CTL queries from the Model Checking Contest 2017 (MCC'17) [15]. The data show that for CTL cardinality queries, we are able to achieve on average 60% of reduction of the query size and about 34% of queries are simplified into trivial queries *true* or *false*, hence avoiding completely the state space exploration. For CTL fireability queries, we achieved 50% reduction of the query size and about 10% of queries are simplified into *true* or *false*. Finally, we compare our simplification algorithm with the one implemented in the tool LoLA [25], the winner of MCC'17 in the several categories including the CTL category, documenting a noticeable performance margin in favour of our approach, both in the number of solved queries purely by the CTL simplification as well as when CTL verification follows the simplification process. For completeness, we also present the data for pure reachability queries where the tool Sara [24] (run parallel with LoLA during MCC'17) performs counterexample guided abstraction refinement and contributes to a high number (about twice as high as our tool) of solved reachability queries without the need to run LoLA's state space exploration. Nevertheless, if we also include the actual verification after the formula simplification, TAPAAL now moves 0.4% ahead of the combined performance of LoLA and Sara.

Related work. Traditionally, the conditions generated by the state equation technique [17] express linear constraints on the number of times the events can occur relative to other events of the system, and form a necessary condition for marking reachability. State equations were used in [14] as an over-approximation technique for preprocessing of reachability formulae in earlier editions of the model checking contest. As the technique can be often inconclusive, extensions of state equations

were studied e.g. in [11] where the authors use traps to increase precision of the method, or in [8] where the state equation technique is extended to liveness properties. State equations, as a necessary condition for reachability, were also used in other application domains like concurrent programming [1, 2]. Our work further extends state equations to full CTL logic and improves the precision of the method by a recursive evaluation of integer linear programs for all subformulae, while employing state equations for each subformula and its negation. State equations were also exploited in [20] in order to guide the state space search based on a minimal solution to the equations. This approach is orthogonal with ours as it essentially defines a heuristic search strategy that in the worst case must explore the whole state space. More recently, the state equation technique was also applied to the coverability problem for Petri nets [4, 12].

Formula rewriting techniques (in order to reduce the size of CTL formulae) are implemented in the tool LoLA [25]. The tool performs formula simplification by employing subformula rewriting rules that include a subset of the rules described in Section 3. LoLA also employs the model checking tool Sara [24] that uses state equations in combination with Counter Example Abstraction Refinement (CEGAR) to perform an exact reachability analysis, being able to answer both reachability and non-reachability questions and hence it is close to being a complete model checker. Sara shows a very convincing performance on reachability queries, however, in the CTL category, we are able to simplify to *true* or *false* almost twice as many formulae, compared to the combined performance of Sara and LoLA.

2 Preliminaries

A *labelled transition system* (LTS) is a tuple $G = (\mathcal{S}, A, \rightarrow)$ where $\mathcal{S}$ is a set of states, A is a set of actions (or labels), and $\rightarrow \subseteq \mathcal{S} \times A \times \mathcal{S}$ is a transition relation. We write $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \rightarrow$ and say that a is *enabled* in s. The set of all enabled actions in a state s is denoted $en(s)$. A state s is a *deadlock* if $en(s) = \emptyset$. We write $s \rightarrow s'$ whenever there is an action a such that $s \xrightarrow{a} s'$.

A *run* starting at s_0 is any finite or infinite sequence $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$ where $s_0, s_1, s_2, \ldots \in \mathcal{S}$, $a_0, a_1, a_2 \cdots \in A$ and $(s_i, a_i, s_{i+1}) \in \rightarrow$ for all respective i. We use $\Pi(s)$ to denote the set of all runs starting at the state s. A run is *maximal* if it is either infinite or ends in a state that

is a deadlock. Let $\Pi^{max}(s)$ denote the set of all maximal runs starting at the state s. A *position* i in a run $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$ refers to the state s_i in the path and is written as π_i. If π is infinite then any i, $0 \leq i$, is a position in π. Otherwise $0 \leq i \leq n$ where s_n is the last state in π.

We now define the syntax and semantics of a *computation tree logic* (CTL) [7] as used in the Model Checking Contest [15]. Let AP be a set of atomic propositions. We evaluate atomic propositions on a given LTS $G = (\mathcal{S}, A, \rightarrow)$ by the function $v : \mathcal{S} \rightarrow 2^{AP}$ so that $v(s)$ is the set of atomic propositions satisfied in the state $s \in \mathcal{S}$.

The CTL syntax is given as follows (where $\alpha \in AP$ ranges over atomic propositions):

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid \alpha \mid \mathit{deadlock} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A(\varphi_1 U \varphi_2) \mid E(\varphi_1 U \varphi_2)$$

We use Φ_{CTL} to denote the set of all CTL formulae. The semantics of a CTL formula φ in a state $s \in \mathcal{S}$ is given in Table 1. We do not use only the minimal set of CTL operators because the query simplification tries to push the negation as far as possible to the atomic predicates. This significantly improves the performance of our on-the-fly CTL model checking algorithm and allows for a more refined query rewriting.

We can now define weighted Petri nets with inhibitor arcs. Let $\mathbb{N}^0 = \mathbb{N} \cup \{0\}$ be the set of natural numbers including 0 and let $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$ be the set of natural numbers including infinity.

Definition 1 (Petri net)**.** *A Petri net is a tuple $N = (P, T, W, I)$ where P and T are finite disjoint sets of places and transitions, $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}^0$ is the weight function for regular arcs, and $I : (P \times T) \rightarrow \mathbb{N}^\infty$ is the weight function for inhibitor arcs.*

A *marking* M on N is a function $M : P \rightarrow \mathbb{N}^0$ where $M(p)$ denotes the number of tokens in the place p. The set of all markings of a Petri net N is written as $\mathcal{M}(N)$. Let $M_0 \in \mathcal{M}(N)$ be a given *initial marking* of N.

A Petri net $N = (P, T, W, I)$ defines an LTS $G(N) = (\mathcal{S}, A, \rightarrow)$ where $\mathcal{S} = \mathcal{M}(N)$ is the set of all markings, $A = T$ is the set of labels, and $M \xrightarrow{t} M'$ whenever for all $p \in P$ we have $M(p) < I((p,t))$ and $M(p) \geq W((p,t))$ such that $M'(p) = M(p) - W((p,t)) + W((t,p))$. We inductively extend the relation $\xrightarrow{t}$ to sequences of transitions $w \in T^*$ such that $M \xrightarrow{\epsilon} M$ and $M \xrightarrow{wt} M'$ if $M \xrightarrow{w} M''$ and $M'' \xrightarrow{t} M'$. We

$s \models \mathit{true}$	
$s \not\models \mathit{false}$	
$s \models \alpha$	iff $\alpha \in v(s)$
$s \models \mathit{deadlock}$	iff $en(s) = \emptyset$
$s \models \varphi_1 \wedge \varphi_2$	iff $s \models \varphi_1$ and $s \models \varphi_2$
$s \models \varphi_1 \vee \varphi_2$	iff $s \models \varphi_1$ or $s \models \varphi_2$
$s \models \neg\varphi$	iff $s \not\models \varphi$
$s \models AX\varphi$	iff $s' \models \varphi$ for all $s' \in \mathcal{S}$ s.t. $s \to s'$
$s \models EX\varphi$	iff there is $s' \in \mathcal{S}$ s.t $s \to s'$ and $s' \models \varphi$
$s \models AF\varphi$	iff for all $\pi \in \Pi^{max}(s)$ there is a position i in π s.t. $\pi_i \models \varphi$
$s \models EF\varphi$	iff there is $\pi \in \Pi^{max}(s)$ and a position i in π s.t. $\pi_i \models \varphi$
$s \models AG\varphi$	iff for all $\pi \in \Pi^{max}(s)$ and for all positions i in π we have $\pi_i \models \varphi$
$s \models EG\varphi$	iff there is $\pi \in \Pi^{max}(s)$ s.t. for all positions i in π we have $\pi_i \models \varphi$
$s \models A(\varphi_1 U \varphi_2)$	iff for all $\pi \in \Pi^{max}(s)$ there is a position i in π s.t. $\pi_i \models \varphi_2$ and for all j, $0 \leq j < i$, we have $\pi_j \models \varphi_1$
$s \models E(\varphi_1 U \varphi_2)$	iff there is $\pi \in \Pi^{max}(s)$ and there is a position i in π s.t. $\pi_i \models \varphi_2$ and for all j, $0 \leq j < i$, we have $\pi_j \models \varphi_1$

Table 1: Semantics of CTL formulae

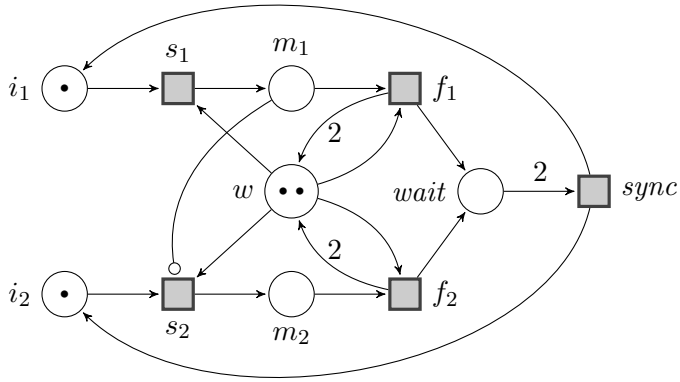


Figure 1: A Petri net modelling two synchronizing processes

write $M \to^* M'$ if there is $w \in T^*$ such that $M \xrightarrow{w} M'$. By $reach(M) = \{M' \in \mathcal{M}(N) \mid M \to^* M'\}$ we denote the set of all markings reachable from M.

Example 1. Figure 1 illustrates an example of a Petri net where places are drawn as circles, transitions as rectangles, regular arcs as arrows with the weight as labels (default weight is 1 and arcs with weight 0 are not depicted) and inhibitor arcs are shown as circle-headed arrows (again the default weight is 1 and arcs with weight ∞ are not depicted). The dots inside places represent the number of tokens (marking). The initial marking in the net can be written by $i_1 i_3 2w$ denoting one token in i_1, one token in i_3 and two tokens in the place w. The net attempts to model two processes that aim to get exclusive access to firing either the transition f_1 or f_2 (making sure that they cannot be enabled concurrently). Once the first process decides to enable transition f_1 by moving the token from i_1 to m_1, the second process is not allowed to place a token into m_2 due to the inhibitor arc connection m_1 to s_2. However, as there is no inhibitor arc in the order direction, it is possible to reach a deadlock in the net by performing $i_1 i_2 2w \xrightarrow{s_2} i_1 m_2 w \xrightarrow{s_1} m_1 m_2$.

Finally, we fix the set of atomic propositions α ($\alpha \in AP$) for Petri nets as used in the MCC Property Language [15]:

$\alpha ::= t \mid e_1 \bowtie e_2$

$e ::= c \mid p \mid e_1 \oplus e_2$

where $t \in T$, $c \in \mathbb{N}^0$, $\bowtie \in \{<, \leq, =, \neq, >, \geq\}$, $p \in P$, and $\oplus \in \{+, -, *\}$. The evaluation function v for a marking M is given as $v(M) = \{t \in T \mid t \in en(M)\} \cup \{e_1 \bowtie e_2 \mid eval_M(e_1) \bowtie eval_M(e_2)\}$ where $eval_M(c) = c$, $eval_M(p) = M(p)$ and $eval_M(e_1 \oplus e_2) = eval_M(e_1) \oplus eval_M(e_2)$.

Formulae that do not use any atomic predicate t for transition firing and *deadlock* are called *CTL cardinality formulae* and formulae that avoid the use of $e_1 \bowtie e_2$ and *deadlock* are called *CTL firability formulate*. Formulae of the form $EF\varphi$ or $AG\varphi$ where φ does not contain any other temporal operator are called *reachability formulae*, and as for CTL can be subdivided into the *reachability cardinality* and *reachability fireability* category.

Example 2. Consider the Petri net in Figure 1 and the reachability fireability formula $EF(f_1 \wedge f_2)$ asking whether there is a reachable marking that enables both f_1 and f_2. By exploring the (finite) part of the LTS reachable from the initial marking $i_1 i_2 2w$, we can conclude that $i_1 i_2 2w \not\models EF(f_1 \wedge f_2)$. However, the slightly modified query $EFAX(f_1 \wedge f_2)$ holds in the initial marking as a deadlocked marking $m_1 m_2$ can be reached, and due to the definition of the universal next modality we have $m_1 m_2 \models AX(f_1 \wedge f_2)$. Another example of a cardinality formula is $E(w \geq 2 \ U \ m_2 = 1)$ asking if there is a computation that marks the place m_2 and before it happens, w must contain at least two tokens. This formula holds in the initial marking by firing the transition s_2.

We shall finish the preliminaries by recalling the basics of linear programming. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables and let $\overline{x} = (x_1, x_2, \ldots, x_n)^T$ be a column vector of the variables. A *linear equation* is of the form $\overline{c} \cdot \overline{x} \bowtie k$ where $\bowtie \in \{=, <, \leq, >, \geq\}$, $k \in \mathbb{Z}$ is an integer, and $\overline{c} = (c_1, c_2, \ldots, c_n)$ is a row vector of integer constants. An *integer linear program* LP is a finite set of linear equations. An (integer) *solution* to LP is a mapping $u : X \to \mathbb{N}^0$ from variables to natural numbers such that for every linear equation $(\overline{c} \cdot \overline{x} \bowtie k) \in LP$, the column vector $\overline{u} = (u(x_1)\ u(x_2) \cdots u(x_n))^T$ satisfies the equation $\overline{c} \cdot \overline{u} \bowtie k$. We use $\mathcal{E}_{lin}^X$ to denote the set of all integer linear programs over the variables X.

An integer linear program with a solution is said to be *feasible*. For our purpose, we only consider feasibility and we are not interested in the optimality of the solution. The feasibility problem of integer linear programs is NP-complete [11, 26], however, there exists a number of efficient linear program solvers (we use `lp_solve` in our implementation [3]).

3 Logical Equivalence of Formulae

Before we give our method for recursive simplification of CTL formulae via the use of state equations in Section 4, we first introduce two other formula simplification techniques. The first method utilizes the initial marking and the second method uses universally valid formulae equivalences. For the rest of this section, we assume a fixed Petri net $N = (P, T, W, I)$ with the initial marking M_0.

For the first simplification, let us define in Table 2 the function $\Omega : \Phi_{CTL} \to \{true, false, ?\}$ that checks if a given formula is trivially satisfiable in the initial marking M_0. Note that we generalize the binary conjunctions and disjunctions to n-ary operations as it corresponds to the implementation in our tool. The correctness of this simplification is expressed in the following theorem.

Theorem 1 (Initial Rewrite)**.** *Let φ be a CTL formula such that $\Omega(\varphi) \neq ?$. Then $M_0 \models \varphi$ if and only if $\Omega(\varphi) = true$.*

For the second simplification, we establish a recursively defined rewrite-function $\rho : \Phi_{CTL} \to \Phi_{CTL}$ given in Table 3 and 4 that is based on logical equivalences for the CTL quantifiers. In the definition of ρ, we assume that the n-ary operators $\vee$ and $\wedge$ are associative and commutative. The correctness is captured in the following theorem.

Theorem 2 (Equivalence Rewriting)**.** *Let $M \in \mathcal{M}(N)$ be a marking on N. Then $M \models \varphi$ if and only if $M \models \rho(\varphi)$.*

4 Formula Simplification via State Equations

We will now describe the main ingredients of our formula simplification algorithm. It is based on a recursive decent on the structure of the formula, checking whether its subformulae and their negations can possibly hold in some reachable marking (here we use the state equation [11, 17] approach) and then propagating back this information through the Boolean and temporal operators.

We use state equations to identify universally true or false subformulae, similarly as e.g. in [14]. The main novelty is that we extend the approach to deal with arbitrary arithmetical expressions and repeatedly solve linear programs for subformulae of the given property so that more significant simplifications can be achieved (we try to solve the state equations both for the subformula and its negation). As a result, we can

$$\Omega(\mathit{true}) = \mathit{true}$$

$$\Omega(\alpha) = M_0 \models \alpha$$

$$\Omega(AX\varphi) = \begin{cases} \mathit{true} & \text{if } M_0 \models \mathit{deadlock} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(\mathit{false}) = \mathit{false}$$

$$\Omega(\mathit{deadlock}) = M_0 \models \mathit{deadlock}$$

$$\Omega(EX\varphi) = \begin{cases} \mathit{false} & \text{if } M_0 \models \mathit{deadlock} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(\neg\varphi) = \begin{cases} \mathit{true} & \text{if } \Omega(\varphi) = \mathit{false} \\ \mathit{false} & \text{if } \Omega(\varphi) = \mathit{true} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(\varphi_1 \wedge \cdots \wedge \varphi_n) = \begin{cases} \mathit{true} & \text{if for all } i,\ 1 \leq i \leq n,\ \text{we have } \Omega(\varphi_i) = \mathit{true} \\ \mathit{false} & \text{if there exists } i,\ 1 \leq i \leq n,\ \text{s.t. } \Omega(\varphi_i) = \mathit{false} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(\varphi_1 \vee \cdots \vee \varphi_n) = \begin{cases} \mathit{true} & \text{if there exists } i,\ 1 \leq i \leq n,\ \text{s.t. } \Omega(\varphi_i) = \mathit{true} \\ \mathit{false} & \text{if for all } i,\ 1 \leq i \leq n,\ \text{we have } \Omega(\varphi_i) = \mathit{false} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(EG\varphi) = \Omega(AG\varphi) = \begin{cases} \mathit{false} & \text{if } \Omega(\varphi) = \mathit{false} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(EF\varphi) = \Omega(AF\varphi) = \begin{cases} \mathit{true} & \text{if } \Omega(\varphi) = \mathit{true} \\ ? & \text{otherwise} \end{cases}$$

$$\Omega(E(\varphi_1 U \varphi_2)) = \Omega(A(\varphi_1 U \varphi_2)) = \begin{cases} \mathit{true} & \text{if } \Omega(\varphi_2) = \mathit{true} \\ \mathit{false} & \text{if } \Omega(\varphi_1) = \Omega(\varphi_2) = \mathit{false} \\ ? & \text{otherwise} \end{cases}$$

Table 2: Simplification rules for a given initial marking M_0

$$
\begin{aligned}
\rho(\alpha) &= \alpha \\
\rho(EG\varphi) &= \rho(\neg AF\rho(\neg\varphi)) \\
\rho(EX\varphi) &= EX\rho(\varphi) \\
\rho(\varphi_1 \wedge \cdots \wedge \varphi_n) &= \rho(\varphi_1) \wedge \cdots \wedge \rho(\varphi_n) \\
\rho(deadlock) &= deadlock \\
\rho(AG\varphi) &= \rho(\neg EF\rho(\neg\varphi)) \\
\rho(AX\varphi) &= AX\rho(\varphi) \\
\rho(\varphi_1 \vee \cdots \vee \varphi_n) &= \rho(\varphi_1) \vee \cdots \vee \rho(\varphi_n)
\end{aligned}
$$

$$
\rho(\neg\varphi) = \begin{cases}
\varphi' & \text{if } \rho(\varphi) = \neg\varphi' \\
AX\rho(\neg\varphi') & \text{if } \rho(\varphi) = EX\varphi' \\
EX\rho(\neg\varphi') & \text{if } \rho(\varphi) = AX\varphi' \\
\rho((\neg\varphi_1) \wedge \cdots \wedge (\neg\varphi_n)) & \text{if } \varphi = \varphi_1 \vee \cdots \vee \varphi_n \\
\rho((\neg\varphi_1) \vee \cdots \vee (\neg\varphi_n)) & \text{if } \varphi = \varphi_1 \wedge \cdots \wedge \varphi_n \\
\neg\rho(\varphi) & \text{otherwise}
\end{cases}
$$

$$
\rho(EF\varphi) = \begin{cases}
\neg deadlock & \text{if } \rho(\varphi) = \neg deadlock \\
EF\varphi' & \text{if } \rho(\varphi) = EF\varphi' \\
\rho(EF\varphi') & \text{if } \rho(\varphi) = AF\varphi' \\
\rho(EF\varphi_2) & \text{if } \rho(\varphi) = E(\varphi_1 U \varphi_2) \\
\rho(EF\varphi_2) & \text{if } \rho(\varphi) = A(\varphi_1 U \varphi_2) \\
\rho(EF\varphi_1 \vee \cdots \vee EF\varphi_n) & \text{if } \rho(\varphi) = \varphi_1 \vee \cdots \vee \varphi_n \\
EF\rho(\varphi) & \text{otherwise}
\end{cases}
$$

Table 3: Equivalence rewriting of CTL formulae, Part 1

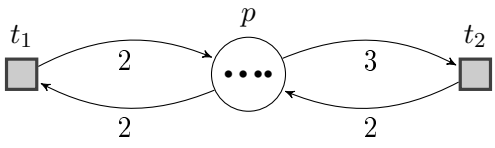


Figure 2: Example Petri net and initial marking for formula simplification

$$\rho(AF\varphi) = \begin{cases} \neg deadlock & \text{if } \rho(\varphi) = \neg deadlock \\ EF\varphi' & \text{if } \rho(\varphi) = EF\varphi' \\ AF\varphi' & \text{if } \rho(\varphi) = AF\varphi' \\ \rho(AF\varphi_2) & \text{if } \rho(\varphi) = A(\varphi_1 U \varphi_2) \\ \rho((EF\varphi_2) \vee (AF\varphi_1)) & \text{if } \rho(\varphi) = \varphi_1 \vee EF\varphi_2 \\ AF\rho(\varphi) & \text{otherwise} \end{cases}$$

$$\rho(A(\varphi_1 U \varphi_2)) = \begin{cases} \neg deadlock & \text{if } \rho(\varphi_2) = \neg deadlock \\ \rho(\varphi_2) & \text{if } \rho(\varphi_1) = deadlock \\ \rho(AF\varphi_2) & \text{if } \rho(\varphi_1) = \neg deadlock \\ EF\varphi_3 & \text{if } \rho(\varphi_2) = EF\varphi_3 \\ AF\varphi_3 & \text{if } \rho(\varphi_2) = AF\varphi_3 \\ \rho((EF\varphi_4) \vee A(\varphi_1 U \varphi_3)) & \text{if } \rho(\varphi_2) = \varphi_3 \vee EF\varphi_4 \\ A(\rho(\varphi_1) U \rho(\varphi_2) & \text{otherwise} \end{cases}$$

$$\rho(E(\varphi_1 U \varphi_2)) = \begin{cases} \neg deadlock & \text{if } \rho(\varphi_2) = \neg deadlock \\ \rho(\varphi_2) & \text{if } \rho(\varphi_1) = deadlock \\ \rho(EF\varphi_2) & \text{if } \rho(\varphi_1) = \neg deadlock \\ EF\varphi_3 & \text{if } \rho(\varphi_2) = EF\varphi_3 \\ \rho((EF\varphi_4) \vee E(\varphi_1 U \varphi_3) & \text{if } \rho(\varphi_2) = \varphi_3 \vee EF\varphi_4 \\ E(\rho(\varphi_1) U \rho(\varphi_2) & \text{otherwise} \end{cases}$$

Table 4: Equivalence rewriting of CTL formulae, Part 2

simplify more formulae into the trivially valid ones (*true*) or invalid ones (*false*) or we can significantly reduce the size of the formulae which can then speed up the state space exploration.

Consider the Petri net in Figure 2 with the initial marking M_0, where $M_0(p) = 4$. The state equation for the reachability formula $EF\ p \geq 5$ (can the place p be marked with at least five tokens) over the variables x_{t_1} and x_{t_2} (representing the number of transition firings of t_1 and t_2 respectively) looks as

$$M_0(p) + \sum_{t \in T} (W(t,p) - W(p,t)) x_t \geq 5$$

which in our example translates to $4 + 0 \cdot x_{t_1} - 1 \cdot x_{t_2} \geq 5$. The inequality clearly does not have a solution in nonnegative integers, hence we can conclude without exploring the state space that $EF\ p \geq 5$ does not hold in the initial marking. Moreover, consider now the formula $EF\ (p \geq 5) \vee (p = 2 \wedge p \leq 7)$. By recursively analyzing the subformulae, we can conclude using the state equations that $p \geq 5$ cannot be satisfied in any reachable marking, hence the formula simplifies to $EF\ (p = 2 \wedge p \leq 7)$. Moreover, by continuing the recursive decent and looking at the subformula $p \leq 7$, we can determine by using state equations, that its negation $p > 7$ cannot be satisfied in any reachable marking. Hence $p \leq 7$ is universally true and the formula further simplifies to an equivalent formula $EF\ p = 2$ for which we have to apply conventional verification techniques.

In what follows, we formally define our formula simplification procedure and extend it to the full CTL logic so that e.g. the formula $EF\ AX\ p \geq 5$ simplifies to the reachability formula *EF deadlock* for which we can use specialized algorithms for deadlock detection (e.g. using the siphon-trap property [13]) instead of the more expensive CTL verification algorithms. Even if a CTL formula does not simplify to a pure reachability property, the reduction in the size of the CTL formula has still a positive effect on the efficiency of the CTL verification algorithms as the state space grows with the number of different subformulae.

Simplification Procedure

Let $N = (P, T, W, I)$ be a fixed Petri net with the initial marking M_0 and φ a given CTL formula. Before we start, we assume that the formula φ has been rewritten into an equivalent one by recursively applying the

rewriting rules in Table 5. Clearly, these rules preserve logical equivalence and they push the negation down to either the atomic propositions or in front of the existential or universal until operators. Moreover, the fireability predicate for a transition t is rewritten to the equivalent cardinality formula.

φ	rewritten φ
t	$p_1 \geq W(p_1,t) \wedge \cdots \wedge p_n \geq W(p_n,t) \wedge$ $p_1 < I(p_1,t) \wedge \cdots \wedge p_n < I(p_n,t)$ where $P = \{p_1, p_2, \ldots, p_n\}$
$e_1 \neq e_2$	$e_1 > e_2 \vee e_1 < e_2$
$e_1 = e_2$	$e_1 \leq e_2 \wedge e_1 \geq e_2$
$\neg(\varphi_1 \wedge \varphi_2)$	$\neg\varphi_1 \vee \neg\varphi_2$
$\neg(\varphi_1 \vee \varphi_2)$	$\neg\varphi_1 \wedge \neg\varphi_2$
$\neg AX\varphi$	$EX\neg\varphi$
$\neg EX\varphi$	$AX\neg\varphi$
$\neg AF\varphi$	$EG\neg\varphi$
$\neg EF\varphi$	$AG\neg\varphi$
$\neg AG\varphi$	$EF\neg\varphi$
$\neg EG\varphi$	$AF\neg\varphi$

Table 5: Rewriting rules

Let $\mathcal{E}^X_{lin}$ be the set of all integer linear programs over the set of variables $X = \{x_t \mid t \in T\}$. Let $LPS \subseteq \mathcal{E}^X_{lin}$ be a finite set of integer linear programs. We say that LPS has a solution, if there exists a linear program $LP \in LPS$ that has a solution.

We will now define a simplification function that, for a given formula $\varphi \in \Phi_{CTL}$, produces a simplified formula and two sets of integer linear programs. The function is of the form

$$simp : \Phi_{CTL} \to \Phi_{CTL} \times 2^{\mathcal{E}^X_{lin}} \times 2^{\mathcal{E}^X_{lin}}$$

and we write $simp(\varphi) = (\varphi', LPS, \overline{LPS})$ when the formula φ is simplified to an equivalent formula φ', and where the following invariant holds:

- if $M \models \varphi$ for some M reachable from M_0 then LPS has a solution, and
- if $M \not\models \varphi$ for some M reachable from M_0 then $\overline{LPS}$ has a solution.

Algorithm 5: Simplify $e_1 \bowtie e_2$

Function $simp(e_1 \bowtie e_2)$
  if e_1 *is not linear or* e_2 *is not linear* **then**
    return $(e_1 \bowtie e_2, \{\{0 \leq 1\}\}, \{\{0 \leq 1\}\})$
  $LPS \leftarrow \{\{const(e_1) \bowtie const(e_2)\}\}$
  $\overline{LPS} \leftarrow \{\{const(e_1) \overline{\bowtie} const(e_2)\}\}$
  if $\{LP \cup BASE \mid LP \in LPS\}$ *has no solution* **then**
    return $simp(false)$
  else if $\{LP \cup BASE \mid LP \in \overline{LPS}\}$ *has no solution* **then**
    return $simp(true)$
  else
    return $(e_1 \bowtie e_2, LPS, \overline{LPS})$

In order to define the simplification function, we use the function $merge : 2^{\mathcal{E}^X_{lin}} \times 2^{\mathcal{E}^X_{lin}} \to 2^{\mathcal{E}^X_{lin}}$ that combines two set of integer linear programs and is defined as $merge\ (LPS_1, LPS_2) = \{LP_1 \cup LP_2 \mid LP_1 \in LPS_1,\ LP_2 \in LPS_2\}$. Finally, let $BASE$ denote the integer linear program with the following equations

$$M_0(p) + \sum_{t \in T} (W(t,p) - W(p,t)) \cdot x_t \geq 0 \quad \text{for all } p \in P$$

that ensures that any solution to $BASE$ must leave a nonnegative number of tokens in every place of N.

First, we postulate $simp(true) = (true, \{\{0 \leq 1\}\}, \emptyset)$, $simp(false) = (false, \emptyset, \{\{0 \leq 1\}\})$, and $simp(deadlock) = (deadlock, \{\{0 \leq 1\}\}, \{\{0 \leq 1\}\})$ and these definitions clearly satisfy our invariant.

Algorithm 6: Simplify $\neg\varphi$

Function $simp(\neg\varphi)$
  $(\varphi', LPS, \overline{LPS}) \leftarrow simp(\varphi)$
  if $\varphi' = true$ **then**
    return $simp(false)$
  if $\varphi' = false$ **then**
    return $simp(true)$
  return $(\neg\varphi', \overline{LPS}, LPS)$

Algorithm 5 describes how to simplify the atomic predicates, where the function *const* takes as input an arithmetic expression e and returns one side of the linear equation as follows:

$$\begin{aligned}
&const(c) && = c \\
&const(p) && = M_0(p) + \sum_{t \in T} (W(t,p) - W(p,t)) \cdot x_t \\
&const(e_1 + e_2) && = const(e_1) + const(e_2) \\
&const(e_1 - e_2) && = const(e_1) - const(e_2) \\
&const(e_1 \cdot e_2) && = const(e_1) \cdot const(e_2).
\end{aligned}$$

In the algorithm we let $\overline{\bowtie}$ denote the dual operation to $\bowtie$, for example $\overline{>}$ becomes $\leq$ and $\overline{\geq}$ becomes $<$. There is a special case that we must handle here. If in either of the expressions e_1 or e_2 we have a multiplication that includes more than one place (i.e. the expression is not linear) then we would return a nonlinear program that cannot be solved by linear program solvers. To handle this situation, if either side of the comparison in nonlinear, we return the formula unchanged and two singleton sets of linear programs $\{\{0 \leq 1\}\}$ that trivially have a solution (any variable assignment is a solution to the linear program $0 \leq 1$) and hence satisfy our invariant.

The simplification of negation $\neg\varphi$ is given in Algorithm 6. It first recursively computes the simplification φ' of φ and if the answer is conclusive then the negated conclusive answer is returned, otherwise we return $\neg\varphi'$ and swap the two sets of linear programs.

In Algorithm 7 we show how to simplify conjunctions and disjunctions of formulae. We give the simplification function for n-ary operators to mimic the implementation closely. We present both conjunction and disjunction in the same pseudocode in order to clarify the symmetry in handling the Boolean connectives. The algorithm recursively simplifies the subformulae and one by one adds the simplified formulae into the resulting proposition φ', unless a conclusive answer (*true*/*false*) can be given immediately or the subformula can be omitted. Note that for conjunction we merge the current LPS and LPS_i returned for the subformula φ_i as if the conjunction is satisfied in some reachable marking then there must be an $LP \in LPS$ and an $LP_i \in LPS_i$ such that $LP \cup LP_i$ has a solution. Symmetrically, we do the merge also for disjunction and the negated sets of linear programs. Finally, we check whether the created systems of linear programs have solutions and in the negative cases we can sometimes draw a conclusive answer.

Algorithm 7: Simplify $\varphi_1 \Diamond \ldots \Diamond \varphi_n$ for $\Diamond \in \{\wedge, \vee\}$

```
1  Function simp(φ_1 ◇ ... ◇ φ_n)
2      Let φ' be an empty formula.
3      if ◇ = ∧ then
4          LPS ← {{0 ≤ 1}};  LPS̄ ← ∅
5      if ◇ = ∨ then
6          LPS ← ∅;  LPS̄ ← {{0 ≤ 1}}
7      for i := 1 to n do
8          (φ'_i, LPS_i, LPS̄_i) ← simp(φ_i)
9          if ◇ = ∧ and φ'_i = false then
10             return simp(false)
11         if ◇ = ∧ and φ'_i ≠ true then
12             φ' ← φ' ∧ φ'_i
13             LPS ← merge(LPS, LPS_i)
14             LPS̄ ← LPS̄ ∪ LPS̄_i
15         if ◇ = ∨ and φ'_i = true then
16             return simp(true)
17         if ◇ = ∨ and φ'_i ≠ false then
18             φ' ← φ' ∨ φ'_i
19             LPS ← LPS ∪ LPS_i
20             LPS̄ ← merge(LPS̄, LPS̄_i)
21     if φ' is empty formula and ◇ = ∧ then
22         return simp(true)
23     if φ' is empty formula and ◇ = ∨ then
24         return simp(false)
25     if ◇ = ∧ and {LP ∪ BASE | LP ∈ LPS} has no solution
         then
26         return simp(false)
27     if ◇ = ∨ and {LP ∪ BASE | LP ∈ LPS̄} has no solution
         then
28         return simp(true)
29     return (φ', LPS, LPS̄)
```

Algorithm 8: Simplify $QX\varphi$ where $Q \in \{A, E\}$

```
1  Function simp(QXφ)
2      (φ', LPS, LPS̄) ← simp(φ)
3      if Q = A and φ' = true then
4          return simp(true)
5      if Q = A and φ' = false then
6          return simp(deadlock)
7      if Q = E and φ' = true then
8          return simp(¬deadlock)
9      if Q = E and φ' = false then
10         return simp(false)
11     return (QXφ', {{0 ≤ 1}}, {{0 ≤ 1}})
```

Algorithm 9: Simplify $QP\varphi$ where $QP \in \{AG, EG, AF, EF\}$

```
1  Function simp(QPφ)
2      (φ', LPS, LPS̄) ← simp(φ)
3      if φ' = true then
4          return simp(true)
5      if φ' = false then
6          return simp(false)
7      return (QPφ', {{0 ≤ 1}}, {{0 ≤ 1}})
```

Simplification of the next operators is given in Algorithm 8. It is worth noticing that for certain situations, the next operator can be removed and replaced with the deadlock proposition (and hence possibly change the CTL formula into a reachability formula). If none of the simplification cases applies, we return the next operator with the simplified formula together with two sets of linear programs with trivial solutions in order to satisfy our invariant. Similarly, the simplification of the unary CTL temporal operators is given in Algorithm 9.

Finally, in Algorithm 10 we present the simplification of binary CTL temporal operators. Here we first simplify φ_2 and see if we can draw some straightforward conclusions. If this is not the case, we also simplify φ_1 and if it evaluates to *true* or *false*, we can either reduce the binary temporal operator into a unary one or completely remove the unary

Algorithm 10: Simplify $Q(\varphi_1 U \varphi_2)$ where $Q \in \{A, E\}$

Function $simp(Q(\varphi_1 U \varphi_2))$
  $(\varphi_2', LPS_2, \overline{LPS}_2) \leftarrow simp(\varphi_2)$
  if $\varphi_2' = true$ **then**
    return $simp(true)$
  if $\varphi_2' = false$ **then**
    return $simp(false)$
  $(\varphi_1', LPS_1, \overline{LPS}_1) \leftarrow simp(\varphi_1)$
  if $\varphi_1' = true$ **then**
    return $(QF\varphi_2', \{\{0 \leq 1\}\}, \{\{0 \leq 1\}\})$
  if $\varphi_1' = false$ **then**
    return $(\varphi_2', LPS_2, \overline{LPS}_2)$
  return $(Q(\varphi_1' U \varphi_2'), \{\{0 \leq 1\}\}, \{\{0 \leq 1\}\})$

operator, respectively.

Example 3. Consider again the net from Example 2. We can simplify the formula $EFAX(f_1 \wedge f_2)$ as follows. Let $X = \{x_{s_1}, x_{s_2}, x_{f_1}, x_{f_2}, x_{sync}\}$ be the variables. Using the rewriting rules from Table 5 we have that $EFAX(f_1 \wedge f_2)$ is equivalent to $EFAX(m_1 \geq 1 \wedge w \geq 1 \wedge m_2 \geq 1)$. The linear equations LPS generated by Algorithm 5 and 7 are as follows.

$$\begin{aligned} x_{s_1} - x_{f_1} &\geq 1 \\ 2 + x_{f_1} + x_{f_2} - x_{s_1} - x_{s_2} &\geq 1 \\ x_{s_2} - x_{f_2} &\geq 1 \end{aligned}$$

We do not include $BASE$ here, as the equations above are already unfeasible (have no integer solution). This follows from the observation that the first and third equation imply that $x_{s_1} > x_{f_1}$ and $x_{s_2} > x_{f_2}$, respectively, and this contradicts the second equation $2 + x_{f_1} + x_{f_2} > x_{s_1} + x_{s_2}$. Therefore, Algorithm 7 simplifies $EFAX(f_1 \wedge f_2)$ to $EFAXfalse$ and by Algorithm 8, we simplify it further to $EFdeadlock$. No further reduction is possible, however, we simplified a CTL formula into a simple reachability formula for which we can now use specialized algorithms for deadlock detection.

We conclude this section with a theorem stating the correctness of the simplification, meaning that for $simp(\varphi) = (\varphi', LPS, \overline{LPS})$ we have

$M_0 \models \varphi$ if and only if $M_0 \models \varphi'$. In order to do so, we prove a stronger claim that allows us to formally introduce the invariant on the sets of linear programs returned by the function *simp*.

Theorem 3 (Formula Simplification Correctness). *Let $N = (P, T, W, I)$ be a Petri net, M_0 an initial marking on N, and $\varphi \in \Phi_{CTL}$ a CTL formula. Let $simp(\varphi) = (\varphi', LPS, \overline{LPS})$. Then for all markings $M \in \mathcal{M}(N)$ such that $M_0 \xrightarrow{w} M$ holds:*

1. *$M \models \varphi$ iff $M \models \varphi'$*
2. *if $M \models \varphi$ then there is $LP \in LPS$ such that $\wp(w)$ is a solution to LP*
3. *if $M \not\models \varphi$ then there is $LP \in \overline{LPS}$ such that $\wp(w)$ is a solution to LP*

where $\wp(w)$ is a solution that assigns to each variable x_t the number of occurrences of the transition t in the transition sequence w.

5 Implementation and Experiments

The formula simplification techniques are implemented in C++ in the `verifypn` engine [14] of the tool TAPAAL [10] and distributed in the latest release at `www.tapaal.net`. The source code is available at `code.launchpad.net/verifypn`.

After parsing the PNML model and the formula, TAPAAL applies sequentially the simplification procedures as depicted in Figure 3, where we first attempt to restructure the formulae to a simpler form using ρ followed by the application of Ω. After this, the main *simp* procedure is called. The simplification can create a formula where additional applications of ρ and Ω are possible and can further reduce the formula size. After the simplification is completed, TAPAAL applies structural reductions to the model, removing or merging redundant transitions and places as described in [14]. The engine now proceeds as follows.

1. If the formulae is of the form *EF deadlock* then siphon-trap analysis is attempted, followed by normal explicit-state verification in case of an inconclusive answer.
2. If the formulae falls within pure reachability category ($EF\varphi$ or $\neg EF\varphi$, where φ does not contain further temporal operators), then

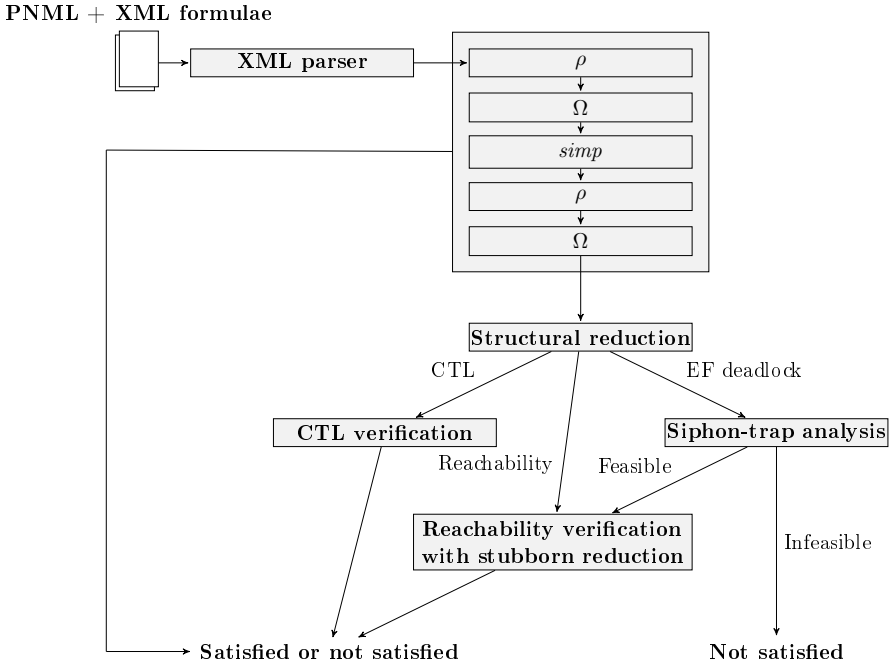


Figure 3: TAPAAL tool-chain and control flow

we call a specialized reachability engine that uses stubborn set reduction.

3. For the general CTL formula, the verification is performed via a translation to a dependency graph and performing on-the-fly computation of its minimum fixed-point assignment as described in [9].

Implementation Details of the Simplification Procedure

During implementation and subsequent experimentation, we discovered that the construction of linear programs for large models can be both time and memory-consuming. In particular, the *merge*-operation causes a quadratic blowup both in the size and the number of linear programs. To remedy this, we have implemented a "lazy" construction of the linear programs—similar to lazy evaluation known from functional programming languages. Instead of computing the full set of linear programs up front, we simply remember the basic linear programs and the tree of operations making up the merged or unioned linear program. Using this construction, we then extract a single linear program on demand,

CTL Cardinality

Algorithm	Solved	% Solved	Reachability	% Reachability	% Reduction
Ω	117	2.3	1834	36.6	27.2
ρ	7	0.1	1437	28.7	24.1
simp	1437	28.7	2425	48.4	45.7
all	**1724**	**34.4**	**2993**	**59.8**	**60.3**

CTL Fireability

Ω	194	3.9	1701	34.0	27.1
ρ	0	0.0	1319	26.3	30.0
simp	255	5.1	1422	28.4	11.0
all	**495**	**9.9**	**2022**	**40.4**	**49.7**

Table 6: Formula simplification for CTL cardinality and fireability

and thus avoid the up-front time and memory overhead of computing the merge and union operations at the call time.

Experimental Setup

To evaluate the performance of our approach, we conduct two series of experiments on the models and formulae from MCC'17 [15]. First, we investigate the effect of the three different simplification methods proposed in this paper along with their combination as depicted in Figure 3. In the second experiment we compare the performance of our simplification algorithms to those used by LoLA, the winner of MCC'17. We also conduct a full run of the verification engines after the formula simplification in order to assess the impact of the simplification on the state space search. All experiments were run on AMD Opteron 6376 Processors, restricted to 14 GB of memory on 313 P/T nets from the MCC'17 benchmark. Each category contains 16 different queries which yields a total of 5008 executions for a given category.

Evaluation of Formula Simplification Techniques

We compare the performance of Ω, ρ and *simp* functions along with their combined version referred to as *all* (applying sequentially ρ, Ω, *simp*, ρ and Ω). The execution of each simplification was limited to 20 minutes per formula (excluding the model parsing time) and a timeout for finding a solution to a linear program using `lp_solve` [3] was set to 120 seconds.

CTL Simplification Only

	TAPAAL		LoLA		LoLA+Sara	
	Solved	% Solved	Solved	% Solved	Solved	% Solved
Cardinality	1724	34	236	5	904	18
Fireability	495	10	173	3	488	10
Total	**2219**	**22**	**409**	**4**	**1392**	**14**

CTL Simplification Followed by Verification

Cardinality	4232	85	3634	73	3810	76
Fireability	3712	74	3663	73	3690	74
Total	**7944**	**79**	**7297**	**73**	**7500**	**75**

Table 7: Tool comparison on CTL formulae

Table 6 reports the numbers (and percentages) of formulae that were solved (simplified to either *true* or *false*), the number of formulae converted from a complex CTL formula into a pure reachability formula and the average formula reduction in percentages (where the formula size before and after the reductions is measured as a number of nodes in its parse tree).

We can observe that the combination of our techniques simplifies about 34% of cardinality queries and 10% of fireability queries into *true* or *false*, while a significant number of queries are simplified from CTL formula into pure reachability problems (60% of cardinality queries and 40% of fireability ones). The average reduction in the query size is 60% for cardinality and 50% for fireability queries. The results are encouraging, though the performance on fireability formulae is considerably worse than for cardinality formulae. The reason is that fireability predicates are translated into Boolean combinations of cardinality predicates and the expanded formulae are less suitable for the simplification procedures due to their increased size. This is also reflected by the time it took to compute the simplification. For CTL cardinality, half of the simplifications terminate in less than 0.05 seconds, 75% simplifications terminate in less than 0.98 seconds and 90% of simplifications terminate in less than 9.46 seconds. The corresponding numbers for CTL fireability are 0.22 seconds, 13.70 seconds and 538.34 seconds.

Comparison with LoLA

We compare the performance of our tool-chain, presented in Figure 3, with the tool LoLA [25] and the combination of LoLA and its linear

Reachability Simplification Only

	TAPAAL		**LoLA**		**LoLA+Sara**	
	Solved	% Solved	Solved	% Solved	Solved	% Solved
Cardinality	2256	45	277	6	3734	75
Fireability	1073	21	296	6	2880	58
Total	**3329**	**33**	**573**	**6**	**6614**	**66**

Reachability Simplification Followed by Verification

Cardinality	4638	93	3734	75	4628	92
Fireability	4402	88	2880	58	4372	87
Total	**9040**	**90**	**6614**	**66**	**9000**	**90**

Table 8: Tool comparison on reachability formulae

program solver Sara [24] that uses the CEGAR approach. In the CTL simplification experiment, we allow 20 minutes for formula simplification (excluding the net parsing time) and count how many solved (simplified to *true* or *false*) queries each tool computed[1]. For CTL verification, we allow the tools first simplify the query and then proceed with the verification according to the best setup the tools provide, again with a 20 minute timeout excluding the parsing time. We run LoLA and Sara in parallel (in their advantage), each of them having 20 minute timeout per execution. The results are presented in Table 7. We can observe that in simplification of CTL cardinality formulae, we are able to provide an answer for 34% of queries while the combination of LoLA and Sara solves only 18% of them. The performance on the CTL fireability simplification is comparable. If we follow the simplification with an actual verification, TAPAAL solves in total 79% of queries and LoLA with Sara 75%. As a result, TAPAAL with the new query simplification algorithms now outperforms the CTL category winner of the last year.

For completeness, in Table 8, we also include the results for the simplification and verification of reachability queries, even though our method is mainly targeted towards CTL formulae. We can notice that thanks to Sara, a fully functional model checker implementing the CEGAR approach, LoLA in combination with Sara solves twice as many queries by simplification as we do. However, once followed by the actual verification (and due to our simplification technique that significantly

[1]We use the current development snapshots of LoLA (based on version 2.0) and Sara (based on version 1.14), kindly provided by the LoLA and Sara development team.

reduces formula sizes), both tools now show essentially comparable performance with a small margin towards TAPAAL, solving 40 additional formulae.

6 Conclusion

We presented techniques for reducing the size of a CTL formula interpreted over the Petri net model. The motivation is to speed up the state space search and to provide a beneficial interplay with other techniques like partial order and structural reductions. The experiential results—compared with LoLA, the winner of MCC'17 competition—document a convincing performance for simplification of CTL formulae as well as for CTL verification. The techniques were not designed specifically for the simplification of reachability formulae, hence the number of solved reachability queries by employing only the simplification is much lower than that by the specialized tools like Sara (being in fact a complete model checker). However, when combined with the state space search followed after the formula simplification, the benefits of our techniques become apparent as we now solve 40 additional formulae compared to the combined performance of LoLA and Sara.

The simplification procedure is less efficient for CTL fireability queries than for CTL cardinality queries. This is the case both for our tool as well as LoLA and Sara. The reason is that we do not handle fireability predicates directly and unfold them into Boolean combination of cardinality predicates. This often results in significant explosion in query sizes. The future work will focus on overcoming this limitation and possibly handling the fireability predicates directly in the engine.

Acknowledgements. We would like to thank Karsten Wolf and Torsten Liebke from Rostock University for providing us with the development snapshot of the latest version of LoLA and for their help with setting up the tool and answering our questions. The last author is partially affiliated with FI MU, Brno.

References

[1] G.S Avrunin, U.A. Buy, and J.C. Corbett. "Integer Programming in the Analysis of Concurrent Systems". In: *Computer Aided Veri-*

fication. Vol. 575. LNCS. Springer Berlin Heidelberg, 1992, pp. 92–102. DOI: 10.1007/3-540-55179-4_10.

[2] G.S. Avrunin et al. "Automated Analysis of Concurrent Systems With the Constrained Expression Toolset". In: *IEEE Transactions on Software Engineering* 17.11 (1991). IEEE, pp. 1204–1222. DOI: 10.1109/32.106975.

[3] M. Berkelaar, K. Eikland, and P. Notebaert. *lpsolve: Open Source (Mixed-Integer) Linear Programming System*. http://lpsolve.sourceforge.net/5.5/. 2004.

[4] M. Blondin et al. "Approaching the Coverability Problem Continuously". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 9636. LNCS. Springer Berlin Heidelberg, 2016, pp. 480–496. DOI: 10.1007/978-3-662-49674-9_28.

[5] E.M. Clarke and E.A. Emerson. "Design and Synthesis of Synchronization Skeletons Ssing Branching Time Temporal Logic". In: *Logics of Programs*. Vol. 131. LNCS. Springer Berlin Heidelberg, 1982, pp. 52–71. DOI: 10.1007/BFb0025774.

[6] E.M. Clarke, E.A. Emerson, and J. Sifakis. "Model Checking: Algorithmic Verification and Debugging". In: *Commun. ACM* 52.11 (2009). Association for Computing Machinery, pp. 74–84. DOI: 10.1145/1592761.1592781.

[7] E.M. Clarke, E.A. Emerson, and A.P. Sistia. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications". In: *ACM Trans. Program. Lang. Syst.* 8.2 (1986). Association for Computing Machinery, pp. 244–263. DOI: 10.1145/5397.5399.

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

[9] A.E. Dalsgaard et al. "Extended Dependency Graphs and Efficient Distributed Fixed-Point Computation". In: *Application and Theory of Petri Nets*. Vol. 10258. LNCS. Springer Berlin Heidelberg, 2017, pp. 139–158. DOI: 10.1007/978-3-319-57861-3_10.

[10] A. David et al. "TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 7214. LNCS. Springer Berlin Heidelberg, 2012, pp. 492–497. DOI: 10.1007/978-3-642-28756-5_36.

[11] J. Esparza and S. Melzer. "Verification of Safety Properties Using Integer Programming: Beyond the State Equation". In: *International Journal on Software Tools for Technology Transfer* 16.2 (2000). Springer, pp. 159–189. DOI: 10.1023/A:1008743212620.

[12] T. Geffroy, J. Leroux, and G. Sutre. "Occam's Razor Applied to the Petri Net Coverability Problem". In: *Reachability Problems*. Vol. 9899. LNCS. Springer International Publishing, 2016, pp. 77–89. DOI: 10.1007/978-3-319-45994-3_6.

[13] M. Hack. *Analysis of Production Schemata by Petri Nets*. Tech. rep. 1972.

[14] J.F. Jensen et al. "TAPAAL and Reachability Analysis of P/T Nets". In: *Transactions on Petri Nets and Other Models of Concurrency XI*. Vol. 9930. LNCS. Springer Berlin Heidelberg, 2016, pp. 307–318. DOI: 10.1007/978-3-662-53401-4_16.

[15] F. Kordon et al. *Complete Results for the 2017 Edition of the Model Checking Contest*. http://mcc.lip6.fr/2017/results.php.

[16] L.M. Kristensen, K. Schmidt, and A. Valmari. "Question-Guided Stubborn Set Methods for State Properties". In: *Formal Methods in System Design* 29.3 (2006). Springer, pp. 215–251. DOI: 10.1007/s10703-006-0006-1.

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

[18] T. Murata and J. Koh. "Reduction and Expansion of Live and Safe Marked Graphs". In: *IEEE Transactions on Circuits and Systems* 27.1 (1980). IEEE, pp. 68–71. DOI: 10.1109/TCS.1980.1084711.

[19] K. Schmidt. "Integrating Low Level Symmetries into Reachability Analysis". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 5404. LNCS. Springer Berlin Heidelberg, 2000, pp. 315–330. DOI: 10.1007/3-540-46419-0_22.

[20] K. Schmidt. "Narrowing Petri Net State Spaces Using the State Equation". In: *Fundamenta Informaticae* 47.3-4 (2001). IOS Press, pp. 325–335.

[21] K. Schmidt. "Stubborn Sets for Standard Properties". In: *Application and Theory of Petri Nets*. Vol. 1639. LNCS. Springer Berlin Heidelberg, 1999, pp. 46–65. DOI: 10.1007/3-540-48745-X_4.

[22] A. Valmari. "Stubborn Set Intuition Explained". In: *Transactions on Petri Nets and Other Models of Concurrency XII*. Vol. 10470. LNCS. Springer Berlin Heidelberg, 2017, pp. 140–165. DOI: 10.1007/978-3-662-55862-1_7.

[23] A. Valmari. "Stubborn Sets for Reduced State Space Generation". In: *Advances in Petri Nets 1990*. Vol. 483. LNCS. Springer Berlin Heidelberg, 1991, pp. 491–515. DOI: 10.1007/3-540-53863-1_36.

[24] H. Wimmel and K. Wolf. "Applying CEGAR to the Petri Net State Equation". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 6605. LNCS. Springer Berlin Heidelberg, 2011, pp. 224–238. DOI: 10.1007/978-3-642-19835-9_19.

[25] K. Wolf and M. Koutny. "Running LoLA 2.0 in a Model Checking Competition". In: *Transactions on Petri Nets and Other Models of Concurrency XI*. Vol. 9930. LNCS. Springer Berlin Heidelberg, 2016, pp. 274–285. DOI: 10.1007/978-3-662-53401-4_13.

[26] L.A. Wolsey and G.L. Nemhauser. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1999. ISBN: 978-0-471-35943-2.

F Proof of Theorem 1

Proof. The proof proceeds by structural induction on φ.

$\varphi = \mathit{true}$: Trivial.

$\varphi = \mathit{false}$: Trivial.

$\varphi = \alpha$: Trivial.

$\varphi = \mathit{deadlock}$: Trivial.

$\varphi = \varphi_1 \wedge \cdots \wedge \varphi_n$: For all $i \in \{1, \ldots, n\}$, we know by the structural induction hypothesis that if $\Omega(\varphi_i) \neq ?$ then $M_0 \models \varphi_i$ iff $\Omega(\varphi_i) = \mathit{true}$. Assume that $\Omega(\varphi) \neq ?$ is true. We need to show the following two implications: (1) if $M_0 \models \varphi_1 \wedge \cdots \wedge \varphi_n$ then $\Omega(\varphi_1 \wedge \cdots \wedge \varphi_n) = \mathit{true}$, and (2) if $\Omega(\varphi_1 \wedge \cdots \wedge \varphi_n) = \mathit{true}$ then $M_0 \models \varphi_1 \wedge \cdots \wedge \varphi_n$.

- Implication (1): Assume $M_0 \models \varphi_1 \wedge \cdots \wedge \varphi_n$. Then for all $i \in \{1, \ldots, n\}$ we must have that $M_0 \models \varphi_i$. Assume for the sake of contradiction that there exists $i \in \{1, \ldots, n\}$ s.t. $\Omega(\varphi_i) = \mathit{false}$. Then by the induction hypothesis we have $M_0 \not\models \varphi_i$, which is a contradiction. Therefore we must have $\Omega(\varphi_i) \neq \mathit{false}$ for all $i \in \{1, \ldots, n\}$, and from the definition of $\Omega(\varphi)$ in Table 2 we have $\Omega(\varphi) \neq \mathit{false}$. By assumption we have $\Omega(\varphi) \neq ?$, leaving only $\Omega(\varphi_1 \wedge \cdots \wedge \varphi_n) = \mathit{true}$ as the conclusion.
- Implication (2): Assume $\Omega(\varphi) = \mathit{true}$. Then we have $\Omega(\varphi_i) = \mathit{true}$ for all $i \in \{1, \ldots, n\}$, from the definition of $\Omega(\varphi)$ in Table 2. Due to the induction hypothesis we have $M_0 \models \varphi_i$ for all i and $M_0 \models \varphi_1 \wedge \cdots \wedge \varphi_n$ follows from the semantics of $\varphi_1 \wedge \cdots \wedge \varphi_n$.

$\varphi = \varphi_1 \vee \cdots \vee \varphi_n$: This case is analogous to the conjunction case.

$\varphi = AX\varphi'$: Assume that $\Omega(\varphi) \neq ?$ is true. Since we by assumption have $\Omega(\varphi) \neq ?$, and $\Omega(\varphi) = \mathit{false}$ can never occur due to the definition of $\Omega(\varphi)$ in Table 2, we must have $M_0 \models \mathit{deadlock}$ and $\Omega(AX\varphi') = \mathit{true}$. If $M_0 \models \mathit{deadlock}$ then $M_0 \models AX\varphi'$ trivially follows from the semantics of $AX\varphi'$. We therefore have $M_0 \models AX\varphi'$ iff $\Omega(AX\varphi') = \mathit{true}$ follows.

$\varphi = EX\varphi'$: This case is analogous to the $AX\varphi'$ case.

$\varphi = EG\varphi'$: By structural induction we have if $\Omega(\varphi') \neq ?$ then $M_0 \models \varphi'$ iff $\Omega(\varphi') = \mathit{true}$. Assume that $\Omega(\varphi) \neq ?$ is true. Since we by assumption have $\Omega(\varphi) \neq ?$, and $\Omega(\varphi) = \mathit{true}$ can never occur due to the definition of $\Omega(\varphi)$ in Table 2, we must have $\Omega(\varphi') = \mathit{false}$ and $\Omega(EG\varphi') = \mathit{false}$. Due to the induction hypothesis we have $M_0 \not\models \varphi'$, and $M_0 \not\models EG\varphi'$ follows from the semantics of $EG\varphi'$. We therefore have $M_0 \models EG\varphi'$ iff $\Omega(EG\varphi') = \mathit{true}$ follows.

$\varphi = AG\varphi'$: This case is analogous to the $EG\varphi'$ case.

$\varphi = EF\varphi'$: By structural induction we have if $\Omega(\varphi') \neq ?$ then $M_0 \models \varphi'$ iff $\Omega(\varphi') = \mathit{true}$. Assume that $\Omega(\varphi) \neq ?$ is true. Since we by assumption have $\Omega(\varphi) \neq ?$, and $\Omega(\varphi) = \mathit{false}$ can never occur due to the definition of $\Omega(\varphi)$ in Table 2, we must have $\Omega(\varphi') = \mathit{true}$ and $\Omega(EF\varphi') = \mathit{true}$. Due to the induction hypothesis we have $M_0 \models \varphi'$, and $M_0 \models EF\varphi'$ follows from the semantics of $EF\varphi'$. We therefore have $M_0 \models EF\varphi'$ iff $\Omega(EF\varphi') = \mathit{true}$ follows.

$\varphi = AF\varphi'$: This case is analogous to the $EF\varphi'$ case.

$\varphi = E(\varphi_1 U \varphi_2)$: For all $i \in \{1, 2\}$ by structural induction we have if $\Omega(\varphi_i) \neq ?$ then $M_0 \models \varphi_i$ iff $\Omega(\varphi_i) = \mathit{true}$. Assume that $\Omega(\varphi) \neq ?$ is true. We need to show the following two implications: (1) if $M_0 \models E(\varphi_1 U \varphi_2)$ then $\Omega(E(\varphi_1 U \varphi_2)) = \mathit{true}$, and (2) if $\Omega(E(\varphi_1 U \varphi_2)) = \mathit{true}$ then $M_0 \models E(\varphi_1 U \varphi_2)$.

- Implication (1): Assume $M_0 \models E(\varphi_1 U \varphi_2)$. Since we by assumption have $\Omega(\varphi) \neq ?$, there are two additional cases from the definition of $\Omega(\varphi)$ in Table 2: $\Omega(\varphi_2) = \mathit{true}$ or $\Omega(\varphi_1) = \Omega(\varphi_2) = \mathit{false}$. Assume for the sake of contradiction $\Omega(\varphi_1) = \Omega(\varphi_2) = \mathit{false}$ is true. Then from the induction hypothesis we have $M_0 \not\models \varphi_1$ and $M_0 \not\models \varphi_2$, implying $M_0 \not\models E(\varphi_1 U \varphi_2)$ from the semantics of $E(\varphi_1 U \varphi_2)$. This contradicts our assumption that $M_0 \models E(\varphi_1 U \varphi_2)$, and leaves us only with the first case $\Omega(\varphi_2) = \mathit{true}$. We have $\Omega(E(\varphi_1 U \varphi_2)) = \mathit{true}$ trivially follows from the definition of $\Omega(\varphi)$ in Table 2.
- Implication (2): Assume $\Omega(E(\varphi_1 U \varphi_2)) = \mathit{true}$. Then we have $\Omega(\varphi_2) = \mathit{true}$ from the definition of $\Omega(\varphi)$ in Table 2. Due to the induction hypothesis we have $M_0 \models \varphi_2$, and $M_0 \models E(\varphi_1 U \varphi_2)$ follows from the semantics of $E(\varphi_1 U \varphi_2)$.

$\varphi = A(\varphi_1 U \varphi_2)$: This case is analogous to the $E(\varphi_1 U \varphi_2)$ case.

□

G Proof of Theorem 2

Proof. The proof is by structural induction on φ. As a sketch, we will here prove the correctness of the rules for $EF\varphi$ and $E\varphi_1 U \varphi_2$. The proofs for the other rules are analogous.

$\varphi = EF\varphi'$: By structural induction we have $M \models \varphi'$ iff $M \models \rho(\varphi')$. We need to show the following two implications: (1) if $M \models EF\varphi'$ then $M \models \rho(EF\varphi')$, and (2) if $M \models \rho(EF\varphi')$ then $M \models EF\varphi'$.

- Implication (1): Assume $M \models EF\varphi'$. Then there exists $M' \in \mathcal{M}(N)$ s.t. $M \rightarrow^* M'$ and $M' \models \varphi'$. Due to the induction hypothesis we have $M' \models \rho(\varphi')$. There are now 6 cases given by the definition of $\rho(EF\varphi')$ in Table 3. The otherwise case is trivial due to the induction hypothesis.
  - Case $\rho(\varphi') = \neg deadlock$: If $M' \models \neg deadlock$ then we must also have $M \models \neg deadlock$, as $M \rightarrow^* M'$ and $en(M) \neq \emptyset$.
  - Case $\rho(\varphi') = EF\varphi''$: There exists $M'' \in \mathcal{M}(N)$ s.t. $M' \rightarrow^* M''$ and $M'' \models \varphi''$. Since we have $M \rightarrow^* M'$ and $M' \rightarrow^* M''$ we must also have $M \rightarrow^* M''$ is the case, implying that $M \models EF\varphi''$ is true.
  - Case $\rho(\varphi') = AF\varphi''$: Clearly there exists $M'' \in \mathcal{M}(N)$ s.t. $M' \rightarrow^* M''$ and $M'' \models \varphi''$ by the semantics of $AF\varphi''$. Since we have $M \rightarrow^* M'$ and $M' \rightarrow^* M''$ we must also have $M \rightarrow^* M''$ is the case, implying that $M \models EF\varphi''$ is true.
  - Case $\rho(\varphi') = E(\varphi_1 U \varphi_2)$: Clearly there exists $M'' \in \mathcal{M}(N)$ s.t. $M' \rightarrow^* M''$ and $M'' \models \varphi_2$ by the semantics of $E(\varphi_1 U \varphi_2)$. Since we have $M \rightarrow^* M'$ and $M' \rightarrow^* M''$ we must also have $M \rightarrow^* M''$ is the case, implying that $M \models EF\varphi_2$ is true.
  - Case $\rho(\varphi') = A(\varphi_1 U \varphi_2)$: Clearly there exists $M'' \in \mathcal{M}(N)$ s.t. $M' \rightarrow^* M''$ and $M'' \models \varphi_2$ by the semantics of $A(\varphi_1 U \varphi_2)$. Since we have $M \rightarrow^* M'$ and $M' \rightarrow^* M''$

we must also have $M \rightarrow^* M''$ is the case, implying that $M \models EF\varphi_2$ is true.

- Case $\rho(\varphi') = \varphi_1 \vee \cdots \vee \varphi_n$: Due to the semantics of $\varphi_1 \vee \cdots \vee \varphi_n$ there exists i s.t. $1 \leq i \leq n$ and $M' \models \varphi_i$. From $M' \models \varphi_i$ we have $M \models EF\varphi_i$, and $M \models EF\varphi_1 \vee \cdots \vee EF\varphi_n$ follows trivially from disjunction introduction.

- Implication (2): Assume $M \models \rho(EF\varphi')$. There are 6 cases given by the definition of $\rho(EF\varphi')$ in Table 3. The otherwise case is trivial due to the induction hypothesis.

  - Case $\rho(\varphi') = \neg deadlock$: Trivially we have that $M \models \neg deadlock$ implies $M \models EF\neg deadlock$ by the semantics of φ.
  - Case $\rho(\varphi') = EF\varphi''$: Trivially we have that $M \models EF\varphi''$ implies $M \models EFEF\varphi''$ by the semantics of φ.
  - Case $\rho(\varphi') = AF\varphi''$: By the induction hypothesis if $M \models \rho(AF\varphi'')$ then we have $M \models AF\varphi''$. Trivially we have that $M \models AF\varphi''$ implies $M \models EFAF\varphi''$ by the semantics of φ.
  - Case $\rho(\varphi') = E(\varphi_1 U \varphi_2)$: By the induction hypothesis if $M \models \rho(E(\varphi_1 U \varphi_2))$ then we have $M \models E(\varphi_1 U \varphi_2)$. Trivially we have that $M \models E(\varphi_1 U \varphi_2)$ implies $M \models EFE(\varphi_1 U \varphi_2)$ by the semantics of φ.
  - Case $\rho(\varphi') = A(\varphi_1 U \varphi_2)$: By the induction hypothesis if $M \models \rho(A(\varphi_1 U \varphi_2))$ then we have $M \models A(\varphi_1 U \varphi_2)$. Trivially we have that $M \models A(\varphi_1 U \varphi_2)$ implies $M \models EFA(\varphi_1 U \varphi_2)$ by the semantics of φ.
  - Case $\rho(\varphi') = \varphi_1 \vee \cdots \vee \varphi_n$: By the induction hypothesis if $M \models \rho(EF\varphi_1 \vee \cdots \vee EF\varphi_n)$ then we have $M \models EF\varphi_1 \vee \cdots \vee EF\varphi_n$. Due to the semantics of $EF\varphi_1 \vee \cdots \vee EF\varphi_n$ there exists i s.t. $1 \leq i \leq n$ and $M \models EF\varphi_i$. There exists $M' \in \mathcal{M}(N)$ s.t. $M \rightarrow^* M'$ and $M' \models \varphi_i$. By disjunction introduction we have $M' \models \varphi_1 \vee \cdots \vee \varphi_n$, and $M \models EF(\varphi_1 \vee \cdots \vee \varphi_n)$ follows since $M \rightarrow^* M'$.

$\varphi = E(\varphi_1 U \varphi_2)$: By structural induction we have $M \models \varphi_1$ iff $M \models \rho(\varphi_1)$ and $M \models \varphi_2$ iff $M \models \rho(\varphi_2)$. We need to show the following two implications: (1) if $M \models E(\varphi_1 U \varphi_2)$ then $M \models \rho(E(\varphi_1 U \varphi_2))$, and (2) if $M \models \rho(E(\varphi_1 U \varphi_2))$ then $M \models E(\varphi_1 U \varphi_2)$.

- Implication (1): Assume $M \models E(\varphi_1 U \varphi_2)$. Then there exists $\pi \in \Pi^{max}(M)$ and a position i s.t. $\pi_i \models \varphi_2$ and for all $j \in \{0, \ldots, i-1\}$ we have $\pi_j \models \varphi_1$. There are 5 cases given by the definition of $\rho(E(\varphi_1 U \varphi_2))$ in Table 3. The otherwise case is trivial due to the induction hypothesis.
  - Case $\rho(\varphi_2) = \neg deadlock$: If $\pi_i \models \neg deadlock$ then we must also have $M \models \neg deadlock$, as $M \to^* \pi_i$ and $en(\pi_i) \neq \emptyset$.
  - Case $\rho(\varphi_1) = deadlock$: Then the only case where $M \models E(\varphi_1 U \varphi_2)$ can be true is when $i = 0$, implying $M \models \varphi_2$. By the induction hypothesis we conclude with $M \models \rho(\varphi_2)$.
  - Case $\rho(\varphi_1) = \neg deadlock$: Clearly, for any path we have $\neg deadlock$ always holds in every intermidiary marking due to the definition of paths, giving us $M \models E(true\, U \varphi_2)$. This is the definition of $M \models EF\varphi_2$ for the minimal set of CTL operators.
  - Case $\rho(\varphi_2) = EF\varphi_3$: There exists $M' \in \mathcal{M}(N)$ s.t. $\pi_i \to^* M'$ and $M' \models \varphi_3$. Since we have $M \to^* \pi_i$ and $\pi_i \to^* M'$ we must also have $M \to^* M'$ is the case, implying that $M \models EF\varphi_3$ is true.
  - Case $\rho(\varphi_2) = \varphi_3 \vee EF\varphi_4$: Either we have $\pi_i \models \varphi_3$ or there exists $M' \in \mathcal{M}(N)$ s.t. $\pi_i \to^* M'$ and $M' \models EF\varphi_4$. In the former case clearly we have $M \models E(\varphi_1 U \varphi_3)$ since the path π exists, and we can conlude with $M \models (EF\varphi_4) \vee E(\varphi_1 U \varphi_3)$ by disjunction introduction. In the latter case since $M \to^* \pi_i$ and $\pi_i \to^* M'$ we must also have $M \to^* M'$ is the case, implying that $M \models EF\varphi_4$ is true, and we can conclude with $M \models (EF\varphi_4) \vee E(\varphi_1 U \varphi_3)$ by disjunction introduction.
- Implication (2): Assume $M \models \rho(E(\varphi_1 U \varphi_2))$. There are 5 cases given by the definition of $\rho(E(\varphi_1 U \varphi_2))$ in Table 3. The otherwise case is trivial due to the induction hypothesis.
  - Case $\rho(\varphi_2) = \neg deadlock$: Trivially we have that $M \models \neg deadlock$ implies $M \models E(\varphi_1 U \neg deadlock)$ by the semantics of φ.
  - Case $\rho(\varphi_1) = deadlock$: Trivially we have that $M \models \varphi_2$ implies $M \models E(\varphi_1 U \varphi_2)$ by the semantics of φ.
  - Case $\rho(\varphi_1) = \neg deadlock$: Trivially we have that $M \models$

$EF\varphi_2$ implies $M \models E(\neg \mathit{deadlock}\, U \varphi_2)$ by the semantics of φ.

- Case $\rho(\varphi_2) = EF\varphi_3$: Trivially we have that $M \models EF\varphi_3$ implies $M \models E(\neg \mathit{deadlock}\, U\, EF\varphi_3)$ by the semantics of φ.
- Case $\rho(\varphi_2) = \varphi_3 \vee EF\varphi_4$: If $M \models (EF\varphi_4) \vee E(\varphi_1 U \varphi_3)$ then either we have $M \models EF\varphi_4$ or $M \models E(\varphi_1 U \varphi_3)$. In the former case by disjunction introduction we have $M \models \varphi_3 \vee EF\varphi_4$, and trivially we have $M \models E(\varphi_1 U \varphi_3 \vee EF\varphi_4)$ by the semantics of φ. In the later case there exists $\pi \in \Pi^{max}(M)$ and a position i s.t. $\pi_i \models \varphi_3$ and for all $j \in \{0, \ldots, i-1\}$ we have $\pi_j \models \varphi_1$. By disjunction introduction we have $\pi_i \models M \models \varphi_3 \vee EF\varphi_4$, and clearly since the path π exists we have $M \models E(\varphi_1 U \varphi_3 \vee EF\varphi_4)$.

□

H Proof of Theorem 3

Proof. We prove the three claims by structural induction on φ.

Base Cases:

$\varphi = \mathit{true}$: Since $\mathit{simp}(\mathit{true}) = (\mathit{true}, \{\{0 \leq 1\}\}, \emptyset)$ the formula remains unchanged and Condition 1 trivially holds. Condition 2 holds because for $\{0 \leq 1\}$ any variable assignment is a solution and Condition 3 is a vacuous case.

$\varphi = \mathit{false}$: Since $\mathit{simp}(\mathit{false}) = (\mathit{false}, \emptyset, \{\{0 \leq 1\}\})$ the formula remains unchanged and Condition 1 trivially holds. Condition 2 is a vacuous case and Condition 3 holds as any variable assignment is a solution to $\{0 \leq 1\}$.

$\varphi = \mathit{deadlock}$: Since $\mathit{simp}(\mathit{deadlock}) = (\mathit{deadlock}, \{\{0 \leq 1\}\}, \{\{0 \leq 1\}\})$ the formula remains unchanged and Condition 1 trivially holds. Conditions 2 and 3 hold as any variable assignment is a solution to $\{0 \leq 1\}$.

$\varphi = e_1 \bowtie e_2$: If either $\mathit{const}(e_1)$ or $\mathit{const}(e_2)$ is not linear, then $\mathit{simp}(e_1 \bowtie e_2) = (e_1 \bowtie e_2, \{\{0 \leq 1\}\}, \{\{0 \leq 1\}\})$ and all three conditions trivially hold. Otherwise, we have $\mathit{LPS} = \{\{\mathit{const}(e_1) \bowtie \mathit{const}(e_2)\}\}$ and $\overline{\mathit{LPS}} = \{\{\mathit{const}(e_1) \mathbin{\overline{\bowtie}} \mathit{const}(e_2)\}\}$. Let M be a

marking such that $M_0 \xrightarrow{w} M$. We will now argue that the three conditions of the theorem hold. There are three subcases to consider:

- Algorithm 5 returns $simp(false) = (false, \emptyset, \{\{0 \leq 1\}\})$ because $\{LP \cup BASE \mid LP \in LPS\}$ has no solution. Then $M \not\models e_1 \bowtie e_2$ as otherwise $\wp(w)$ would be a solution both to $BASE$ as well as $\{const(e_1) \bowtie const(e_2)\}$ due to the construction of the state equations for e_1 and e_2. This means that Condition 1 holds, Condition 2 is vacuous, and Condition 3 holds as $\wp(w)$ is clearly a solution to $\{0 \leq 1\}$.
- Algorithm 5 returns $simp(true) = (true, \{\{0 \leq 1\}\}, \emptyset)$ because $\{LP \cup BASE \mid LP \in \overline{LPS}\}$ has no solution. Then $M \not\models e_1 \overline{\bowtie} e_2$ as otherwise $\wp(w)$ would be a solution both to $BASE$ as well as $\{const(e_1) \overline{\bowtie} const(e_2)\}$ due to the construction of the state equations for e_1 and e_2. This implies that $M \models e_1 \overline{\bowtie} e_2$ and hence Condition 1 holds. Condition 2 holds as $\wp(w)$ is clearly a solution to $\{0 \leq 1\}$ and Condition 3 is vacuous.
- Algorithm 5 returns $(e_1 \bowtie e_2, LPS, \overline{LPS})$ and because the formula was unchanged, Condition 1 trivially holds. Due to the construction of the linear programs based on state equations, it is clear that if $M \models e_1 \bowtie e_2$ then $\wp(w)$ is a solution to both $BASE$ and LPS, implying Condition 2. Symmetrically, if $M \not\models e_1 \bowtie e_2$ then $M \models e_1 \overline{\bowtie} e_2$ and hence $\wp(w)$ is a solution to both $BASE$ and $\overline{LPS}$, meaning that Condition 3 holds too.

Inductive Cases (where $M_0 \xrightarrow{w} M$):

$\varphi = \neg\varphi_1$: Let $simp(\varphi_1) = (\varphi_1', LPS, \overline{LPS})$. By structural induction hypothesis we know that $M \models \varphi_1$ iff $M \models \varphi_1'$ which implies that $M \models \neg\varphi_1$ iff $M \models \neg\varphi_1'$ and Condition 1 is thus satisfied for all three possible returns. Conditions 2 and 3 clearly hold if $simp(false)$ or $simp(true)$ is returned. In case the return value is $(\neg\varphi_1', \overline{LPS}, LPS)$, we use the induction assumption that Conditions 2 and 3 hold for φ_1 and by adding the negation to φ_1 and swapping the sets of linear programs, the conditions hold also for $\neg\varphi_1$.

$\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$: Let $simp(\varphi_i) = (\varphi_i', LPS_i, \overline{LPS}_i)$ for all i, $1 \leq i \leq n$. By structural induction hypothesis, for all i we have

$M \models \varphi_i$ iff $M \models \varphi'_i$. Hence if for some i it is the case that $\varphi'_i = \mathit{false}$ we can terminate and return $\mathit{simp}(\mathit{false})$ since it is clear $M \not\models \varphi$ and all three conditions hold as required. Similarly, if $\varphi'_i = \mathit{true}$ for some i, then this conjunct can be safely skipped over as it will not change the validity of φ'. Moreover, should this be the case for all i, $1 \leq i \leq n$, then we can safely return $\mathit{simp}(\mathit{true})$ and all three conditions still hold.

Assume that $M \models \varphi$, then clearly $M \models \varphi_i$ for all i and by the induction hypothesis $\wp(w)$ is a solution to each LPS_i, meaning that for each i there is an $LP_i \in LPS_i$ for which $\wp(w)$ is a solution. By the definition of the merge operation, we know that there exists an LP such that $LP \subseteq LP_1 \cup LP_2 \cup \ldots \cup LP_n$, where $LP \in LPS$ and $\wp(w)$ is a solution to LP. As a consequence, LPS has $\wp(w)$ as a solution and Condition 2 is thus satisfied. Conversely, if LPS has no solution, this implies $M \not\models \varphi$ and in this case we can safely return $\mathit{simp}(\mathit{false})$.

Let us assume that $M \not\models \varphi$, implying that $M \not\models \varphi_i$ for at least one i. By induction hypothesis, there is $LP \in \overline{LPS}_i$ such that $\wp(w)$ is a solution to LP and because we perform unions of all $\overline{LPS}_i$, clearly $LP \in \overline{LPS}$ and Condition 3 therefore holds.

$\varphi = \varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_n$: Let $\mathit{simp}(\varphi_i) = (\varphi'_i, LPS_i, \overline{LPS}_i)$ for all i, $1 \leq i \leq n$. By structural induction hypothesis, for all i we have $M \models \varphi_i$ iff $M \models \varphi'_i$. Hence if for some i it is the case that $\varphi'_i = \mathit{true}$ we can terminate and return $\mathit{simp}(\mathit{true})$ since it is clear $M \models \varphi$ and all three conditions hold as required. Similarly, if $\varphi'_i = \mathit{false}$ for some i, then this conjunct can be safely skipped over as it will not change the validity of φ'. Moreover, should this be the case for all i, $1 \leq i \leq n$, then we can safely return $\mathit{simp}(\mathit{false})$ and all three conditions still hold.

Assume that $M \models \varphi$, then clearly $M \models \varphi_i$ for some i and by the induction hypothesis $\wp(w)$ is a solution to some $LP \in LPS_i$. Then the algorithm either returns $\mathit{simp}(\mathit{true})$ and all three conditions hold, or $LP \in LPS$ as we perform the union operation on LPS and this guarantees that Condition 2 holds once the algorithm returns $(\varphi', LPS, \overline{LPS})$.

Let us assume that $M \not\models \varphi$, implying that $M \not\models \varphi_i$ for all i. By induction hypothesis, for all i there is $LP_i \in \overline{LPS}_i$ such that $\wp(w)$ is a solution to LP. By the definition of the merge operation, we know

that there exists an LP such that $LP \subseteq LP_1 \cup LP_2 \cup \ldots \cup LP_n$, where $LP \in \overline{LPS}$ and $\wp(w)$ is a solution to LP. As a consequence, $\overline{LPS}$ has $\wp(w)$ as a solution and Condition 3 is thus satisfied. Conversely, if $\overline{LPS}$ has no solution, this implies $M \models \varphi$ and in this case we can safely return $simp(true)$.

$\varphi = QX\varphi_1$, where $Q \in \{A, E\}$: Let $simp(\varphi_1) = (\varphi_1', LPS, \overline{LPS})$. By structural induction hypothesis, we have $M \models \varphi_1$ iff $M \models \varphi_1'$. In case that φ_1' is either *true* or *false*, the four cases in the algorithm clearly preserve logical equivalence and all three conditions are satisfied. Otherwise we return $QX\varphi_1'$ which is equivalent to $QX\varphi_1$ and Condition 1 remaind satisfied. Both sets of linear programs that are returned have any assignment as a solution, so Conditions 2 and 3 hold too.

$\varphi = QP\varphi_1$ where $QP \in \{AG, EG, AF, EF\}$: This case is analogous to the next operators discussed above.

$\varphi = Q(\varphi_1 U \varphi_2)$ where $Q \in \{A, E\}$: Let $simp(\varphi_1) = (\varphi_1', LPS_1, \overline{LPS}_1)$ and $simp(\varphi_2) = (\varphi_2', LPS_2, \overline{LPS}_2)$. By structural induction hypothesis, we have $M \models \varphi_1$ iff $M \models \varphi_1'$, and $M \models \varphi_2$ iff $M \models \varphi_2'$. If $\varphi_2' = true$ then $Q(\varphi_1 U \varphi_2)$ is equivalent to *true* and we can return $simp(true)$ while satisfying all three conditions. Similarly if $\varphi_2' = false$ we can safely return $simp(false)$. If $\varphi_1' = true$ then $Q(\varphi_1 U \varphi_2)$ becomes logically equivalent to $QF\varphi_2'$ and both sets of linear programs that are returned have any assignment as a solution, so all three conditions are satisfied. In case $\varphi_1' = false$ then necessarily φ_2 must hold immediately and we can return $(\varphi_2', LPS_2, \overline{LPS}_2)$ that satisfies all three conditions by the induction hypothesis. Otherwise we return $Q(\varphi_1' U \varphi_2')$ that is equivalent to φ and the two returned linear programs admit all assignments as solutions, so all three conditions hold.

□

ISSN (online): 2446-1628
ISBN (online): 978-87-7210-907-7

AALBORG UNIVERSITY PRESS