Summary

This thesis is written by Nikolaj Jensen Ulrik and Simon Mejlby Virenfeldt within the field of formal verification. The thesis deals with explicit state model checking of Petri net models using the property specification language Linear Temporal Logic (LTL). It introduces novel techniques for partial order reductions using stubborn sets. The idea focuses on the individual states in the Nondeterministic Büchi Automaton (NBA) which the LTL formula is translated into. Here we determine a subset of states in the NBA in which we are able to employ reachability-preserving stubborn set methods. This has the advantage over classic stubborn set methods for LTL that it works on all LTL formulae and not only the subcategory of next-free formulae.

The thesis also investigates using guided search in order to increase the performance of the depth first explicit model checking algorithms. Here we adapt existing methods from reachability analysis and use them on local information from the NBA to prioritise short-term important information. An alternative heuristic that penalises transitions that have been fired many times, which we call the fire count heuristic, is also investigated and combined with the other heuristics.

All these methods are implemented in the tool TAPAAL and evaluated experimentally against a large and well-established Petri net dataset used in the annual Model Checking Contest (MCC). Evaluation indicates improved performance for the novel reachability-based stubborn method compared to the classic method. For heuristics they all show better performance compared to not using guided search, and combining the fire count heuristic with the one based on local NBA information fares the best.

Finally we compare the entire LTL model checker of TAPAAL against the state of the art model checker ITS-LoLA in the setup of the MCC. The comparison shows a significant improvement compared to ITS-LoLA, especially in queries that contain a counterexmaple.

Automata-Driven Techniques for Partial-Order Reductions and Guided Search of Petri Nets

Nikolaj Jensen Ulrik and Simon Mejlby Virenfeldt

Master Thesis
Aalborg University, Denmark
{nulrik16,sviren16}@student.aau.dk

Abstract. Automaton-based model checking, in which a system under verification is paired with a Nondeterministic Büchi Automaton (NBA) describing illegal behaviour, is the main method of model checking Linear Temporal Logic (LTL). However, while techniques for optimising the system under verification or the NBA independently are well known, there is less knowledge on techniques using both. We present a stubborn set method and heuristics for guided search, both of which use local information in the NBA to optimise the process of exploring the state space of the system being verified. We implement these techniques as an extension to the open source model checking engine `verifypn` used by TAPAAL and evaluate them using the dataset from the 2020 edition of the Model Checking Contest (MCC). We find that the guided search technique improves performance compared to unguided search, and that the NBA-based stubborn set method performs better than the classic stubborn set method. Additionally, we show that the classic method can be used whenever the novel method does not apply, forming a mixed method that performs even better. We find that the improvements gained from combining stubborn sets and heuristics are almost equal to the sum of the improvements from stubborn sets and heuristics individually. Lastly, we compare our LTL model checker to ITS-LoLA, the winner of LTL category of the 2020 edition of the MCC, finding that our model checker answers 56.8 % of queries that ITS-LoLA did not, corresponding to a 10.8 % increase in the number of answers.

1 Introduction

The state space explosion problem is one of the main barriers to model checking large systems. The problem arises because systems descriptions can generate an exponential number of states, and in the case of Petri nets [33] some descriptions even generate infinite state spaces. Addressing this problem has been the subject of much research, with directions including partial order reductions [32,21,43], symbolic model checking [9,3], guided searches using heuristics [15,16], and symmetry reductions [10,35]. Some system description languages afford specialised techniques in addition to the above. For example, state space explosion of Petri nets can be addressed with e.g. structural reductions [31,19,7] or unfolding into occurrence nets [30,17].

Partial order reductions are a family of techniques designed to prune the state space based on interleaving executions. An important category of partial order reduction techniques are the ample set [32], persistent set [21], and stubborn set methods [44]. We focus on stubborn set methods. The goal of the technique is, given a specific state, to determine a subset of actions to explore such that all representative executions are preserved with respect to some desired property. This subset of actions is called the stubborn set of a state. While the technique does have the potential to exponentially reduce the size of state spaces, more complex verification questions such as model checking limit the possible reduction. For example, the portfolio of classic stubborn set methods developed by Valmari does not allow for Linear Temporal Logic (LTL) formulae containing the next-step operator X [44]. Stubborn sets and other partial order reduction techniques are supported in several well-established tools, e.g. Tapaal [12], LoLA 2 [48], and Spin [23], and have proven to be useful [7,27,24].

The main approach to LTL model checking is automata-based model checking, which is based on a translation of LTL formulae into Nondeterministic Büchi Automata (NBAs), which are then synchronised with the system being verified. The goal is then to find a reachable accepting cycle in the synchronised system. While much research has been done on optimising NBAs [47,1,18], and much work has been done on state space reductions as described above, few state space techniques take the automaton into account. For example, the aforementioned next-free LTL preserving method by Valmari is based on the syntax of the formula and is completely agnostic to the choice of verification algorithm [45]. Some of the work done within the field of stubborn sets includes a specialised, automaton-driven approach for a subclass of LTL formulae called simple LTL formulae [27], and more recently an automaton-based stubborn set approach for arbitrary LTL formulae [28], although to the authors' knowledge the latter does not have an implementation as of writing. This idea of automaton-driven techniques is interesting and worth more research, as the automaton can provide more detailed local information compared to looking at the entire LTL formula.

In state space exploration, the choice of which successor state to explore first can have a big impact on the performance of depth-first algorithms such as Nested Depth First Search (NDFS) [11] and Tarjan's algorithm [20], which are important LTL verification algorithms. A poor choice of successor can cause a lot of time to be wasted exploring executions without relevant behaviour. A way of addressing this problem is by using heuristics to guide the search in a direction more likely to be relevant. Previous work in this direction includes [15,14] in which A^* is used as a search algorithm with heuristics based on finite state machine representations, and [25] presents a best-first search algorithm using a syntax-driven heuristic as a guide for reachability analysis of Petri nets.

We contribute a novel automata-driven stubborn set method and automata-based heuristics for guided search for model checking LTL formulae on Petri nets. The stubborn set method is a non-trivial adaptation of the stubborn set method for reachability analysis presented in [7]. This new method determines a subset of non-accepting NBA states from which we are able to use the existing

method, with the goal of leaving non-accepting NBA states earlier than without our method. The guided search is based on the heuristics of [25] describing the distance between a state and the satisfaction of a formula. We extend this method to use in non-accepting NBA states to determine the distance between a state and a state which is able to leave the current NBA state. Common to our techniques is a desire to leave non-accepting NBA state as quickly as possible to find a accepting state earlier than otherwise. We provide an implementation of these novel techniques as an extension of the open source model checker `verifypn` used in Tapaal. We evaluate the performance of the techniques using the LTL dataset of the 2020 edition of the Model Checking Contest (MCC). We also compare our model checker to the state of the art Petri net model checker ITS-LoLA [39,48], answering an additional 10.8 % of LTL queries which makes up 56.8 % of LTL queries that ITS-LoLA could not answer.

The structure of the thesis is as follows. In section 2 the automata-based approach to LTL model checking is presented as well as the syntax and semantics of Petri nets. section 3 describes both the classic stubborn set method known in the literature and a novel approach to stubborn set reductions, alongside the implementation of these techniques for Petri nets. section 4 describes our approach to guided search and the heuristics used for guided search. Sections 3 and 4 also conclude with experimental evaluation of the presented methods. section 5 evaluates combinations of stubborn sets and heuristics and contains comparison to a state of the art LTL model checker for Petri nets, ITS-LoLA [39,48]. We conclude the thesis in section 6.

Related Work Stubborn set methods have been applied to a wide range of problems outside of the previously mentioned work. In [34] stubborn set methods are presented for many Petri net properties such as home marking or transition liveness among others. There are also reachability-preserving stubborn sets for timed systems [22,4] and more recently for timed games [5]. The stubborn set methods of [27,28], also mentioned above, represent existing work on automaton guided stubborn set methods. While our method has similar goals as [28], the approaches differ. In particular, our method does not explicitly restrict stubborn sets based on the self-looping formula of NBA states, and we precompute in which states the method is applicable.

In [15] guided search strategies for LTL model checking using variants of A^* search are presented. Their guided search addresses situation where an accepting state has been found and a cycle needs to be closed, in contrast with the heuristics in the present work which guides the search toward any form of state change in the NBA, and which are not applied in accepting states. However, they assume that individual processes are given at finite state machines and have heuristics dependent on being able to compute exact distances to specific states in individual processes, an approach that is not compatible with Petri nets. Another approach to guided search was presented in [37] where the state equations, a system of linear equations that overapproximate the behaviour of a Petri net, was used to guide the search based on the assumption that the overapproximation is reasonably accurate. While this work is interesting, it was not

applied to LTL, and solving potentially large linear programs often is a source of overhead whenever the method does not help. In contrast, we emphasise simpler heuristics that are fast to compute.

2 Preliminaries

We now define basic concepts of LTL model checking in an abstract setting. Afterwards we introduce Petri nets and their translation to labelled transition systems. In the following, let $\mathbb{N}^0$ denote the natural numbers including zero, let ∞ be such that $x < \infty$ for all $x \in \mathbb{N}^0$, and let *tt* and *ff* denote true and false respectively.

2.1 Labelled Transition Systems

We will now introduce Labelled Transition System (LTS), on which we will define the semantics of LTL. The standard way of defining LTL semantics is with Kripke structures, but when we introduce our stubborn set methods we need to reason about the actions taken. Let AP a fixed set of *atomic propositions*, which are basic properties exhibited by states. A Labelled Transition System (LTS) is a tuple $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ where

- S is a possibly infinite set of states,
- Σ is a finite set of actions,
- $\rightarrow \subseteq S \times \Sigma \times S$ is a transition relation,
- $L : S \rightarrow 2^{AP}$ is a labelling function, and
- $s_0 \in S$ is a designated initial state.

We write $s \xrightarrow{\alpha} s'$ if $(s, \alpha, s') \in \rightarrow$, and $s \rightarrow s'$ if there exists α such that $s \xrightarrow{\alpha} s'$. We write $s \xrightarrow{\varepsilon} s$ where ε is the empty string, and $s \xrightarrow{\alpha w} s'$ if $s \xrightarrow{\alpha} s''$ and $s'' \xrightarrow{w} s'$ for some $w \in \Sigma^*$. For $s \in S$, if no state s' exists such that $s \rightarrow s'$, we call s a *deadlock* state, written $s \not\rightarrow$, and if s is not a deadlock state we write $s \rightarrow$. We use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$. We say that α is *enabled* in s, written $s \xrightarrow{\alpha}$, if there exists s' such that $s \xrightarrow{\alpha} s'$, and the set of all enabled actions in s is denoted $\mathrm{en}(s) = \{\alpha \in \Sigma \mid s \xrightarrow{\alpha}\}$. For any $a \in AP$ we say that s *satisfies* a, written $s \models a$, if $a \in L(s)$, and define $[\![a]\!] = \{s \in S \mid s \models a\}$ to be the set of states satisfying a.

Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS. A *run* π in $\mathcal{T}$ is a potentially infinite sequence of states $s_1 s_2 \ldots$ such that for all $i \geq 1$, either $s_i \rightarrow s_{i+1}$ or s_i is a deadlock state and $s_{i+i} = s_i$. An infinite run $\pi = s_1 s_2 \ldots$ induces an infinite word $\sigma_\pi = L(s_1)L(s_2) \ldots \in (2^{AP})^\omega$. We say that words $\sigma, \sigma' \in (2^{AP})^\omega$ are *stuttering equivalent*, written $\sigma \approx \sigma'$, if there exists a word $\sigma_{\min} \in (2^{AP})^\omega = A_0 A_1 \ldots$ such that $\sigma = A_0^{n_1} A_1^{n_2} \ldots$ and $\sigma' = A_0^{m_1} A_1^{m_2} \ldots$ for some $n_1, n_2, \ldots, m_1, m_2 \in \mathbb{N}^+$, i.e. σ and σ' differ only by finite repetitions of any A_i. For example, the word $abab \ldots \approx aabbaabb \ldots$ since the latter string only differs from the former by repeating each a and each b once. We define $\mathrm{Runs}(s)$ as the set of runs starting in s, and $\mathrm{Runs}(\mathcal{T}) = \mathrm{Runs}(s_0)$ where s_0 is the initial state of $\mathcal{T}$. We define the language of s as $\mathcal{L}(s) = \{\sigma_\pi \in (2^{AP})^\omega \mid \pi \in \mathrm{Runs}(s)\}$. For a word $\sigma = A_0 A_1 \ldots$ we define $\sigma^i = A_i A_{i+1} \ldots$ to be the ith suffix of σ for $i \geq 0$.

2.2 Linear Temporal Logic

The syntax of Linear Temporal Logic (LTL) is given by the abstract grammar

$$\varphi_1, \varphi_2 ::= a \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi_1 \mid \mathsf{F}\varphi_1 \mid \mathsf{G}\varphi_1 \mid \mathsf{X}\varphi_1 \mid \varphi_1 \,\mathsf{U}\, \varphi_2$$

where φ_1 and φ_2 range over LTL formulae and $a \in AP$ ranges over atomic propositions. An infinite word $\sigma = A_0 A_1 \ldots \in (2^{AP})^\omega$ *satisfies* an LTL formula φ, written $\sigma \models \varphi$, if and only if the following inductive definition is satisfied:

$$\begin{aligned}
\sigma \models a &\iff a \in A_0 \\
\sigma \models \varphi_1 \wedge \varphi_2 &\iff \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma \models \varphi_1 \vee \varphi_2 &\iff \sigma \models \varphi_1 \text{ or } \sigma \models \varphi_2 \\
\sigma \models \neg\varphi_1 &\iff \text{not } \sigma \models \varphi_1 \\
\sigma \models \mathsf{F}\varphi_1 &\iff \exists i \geq 0 \,.\, \sigma^i \models \varphi_1 \\
\sigma \models \mathsf{G}\varphi_1 &\iff \forall i \geq 0 \,.\, \sigma^i \models \varphi_1 \\
\sigma \models \mathsf{X}\varphi_1 &\iff \sigma^1 \models \varphi_1 \\
\sigma \models \varphi_1 \,\mathsf{U}\, \varphi_2 &\iff \exists j \geq 0 \,.\, \sigma^j \models \varphi_2 \text{ and } \forall i \in [0; j[\,.\, \sigma^i \models \varphi_1
\end{aligned}$$

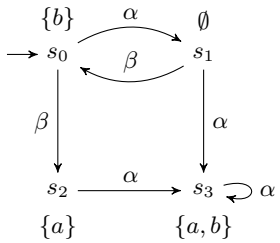
The set of LTL formulae without the X operator is denoted $\mathrm{LTL}_{\mathsf{X}}$. Such formulae cannot distinguish stuttering equivalent words, so we call $\mathrm{LTL}_{\mathsf{X}}$ the *stuttering-insensitive* fragment of LTL. Let $\mathcal{T} = (S, \Sigma, \to, L, s_0)$ be an LTS. For a state $s \in S$, we say that $s \models \varphi$ if and only if for all words $\sigma \in \mathcal{L}(s)$ we have $\sigma \models \varphi$, and we say that $\mathcal{T} \models \varphi$ if and only if $s_0 \models \varphi$.

Example 2.1. Figure 1a illustrates an LTS $\mathcal{T} = (S, \Sigma, \to, L, s_0)$ with the set of actions $\Sigma = \{\alpha, \beta\}$ and the set of atomic propositions $AP = \{a, b\}$. States are annotated with their labels—for example, $L(s_2) = \{a\}$ and $L(s_1) = \emptyset$. The sequence of states $\pi = s_0 s_1 s_0 s_1 (s_3)^\omega$ is a run starting in s_0 which loops in s_3 forever, and the induced word of π is $\sigma_\pi = \{b\}\emptyset\{b\}\emptyset(\{a, b\})^\omega$. We can see that σ_π satisfies the LTL formula $\varphi = \mathsf{FG}a$, which means "from some point onward, always a", since for all $i \geq 5$ we have $\sigma_\pi^i \models \mathsf{G}a$. However, s_0 does not satisfy the formula since $\pi' = (s_0 s_1)^\omega$ is a run starting in s_0 that never reaches any of the states s_2 or s_3 where a holds. We call π' a counterexample to the formula φ on the system $\mathcal{T}$.

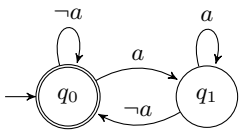
Example 2.2. The LTS in Figure 1a satisfies $\varphi = \mathsf{X}\neg b$, since all the successors of s_0 do not have the label b.

2.3 Nondeterministic Büchi Automata

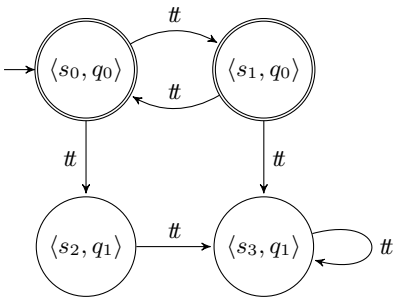
The standard procedure for verifying whether $s \models \varphi$ for some state s and LTL formula φ is automata-based model checking, see e.g. [2]. This technique seeks to find counterexamples to φ by means of synchronisation with a Nondeterministic Büchi Automaton (NBA) equivalent to $\neg\varphi$.



(a) An LTS $\mathcal{T}$. For state s, the label $L(s)$ is denoted beside it, and edges are labelled using the relevant action.



(b) NBA $\mathcal{A}_{\neg \mathsf{FG}a}$ equivalent to the formula $\neg \mathsf{FG}a$. The accepting state $q_0 \in F$ is denoted by two circles, and the initial state $q_0 \in Q_0$ is denoted by the unconnected arrow.



(c) The product system $\mathcal{T} \otimes \mathcal{A}_{\neg \mathsf{FG}a}$.

Fig. 1: Example LTS $\mathcal{T}$ and NBA $\mathcal{A}_{\neg \mathsf{FG}a}$ for model checking whether $\mathcal{T} \models \varphi = \mathsf{FG}a$. Since the product system $\mathcal{T} \otimes \mathcal{A}_{\neg \mathsf{FG}a}$ has an accepting run $(\langle s_0, q_0 \rangle \langle s_1, q_0 \rangle)^\omega$, we can conclude that $\mathcal{T} \not\models \varphi$.

First we introduce the propositions we may find on the guards of the NBA. We let $\mathcal{B}(AP)$ denote the set of propositions over the set of atomic propositions AP, given by the grammar

$$b_1, b_2 ::= tt \mid ff \mid a \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b_1$$

where $a \in AP$ and $b_1, b_2 \in \mathcal{B}(AP)$. We define satisfaction of a proposition b by a set of atomic propositions $A \subseteq AP$, written $A \models b$, inductively as:

$$\begin{aligned} A &\models tt \\ A &\not\models ff \\ A \models a &\iff a \in A \\ A \models b_1 \wedge b_2 &\iff A \models b_1 \text{ and } A \models b_2 \\ A \models b_1 \vee b_2 &\iff A \models b_1 \text{ or } A \models b_2 \\ A \models \neg b_1 &\iff A \not\models b_1 \ . \end{aligned}$$

For any proposition $b \in \mathcal{B}(AP)$ and any LTS state $s \in S$, we write $s \models b$ if $L(s) \models b$. We let the denotation of a proposition be the set of sets of atomic propositions given by $[\![b]\!] = \{A \in 2^{AP} \mid A \models b\}$

Remark 2.3. We define the equality of propositions based on their denotation. That is, for any propositions $b_1, b_2 \in \mathcal{B}(AP)$, we write $b_1 = b_2$ iff $[\![b_1]\!] = [\![b_2]\!]$.

An NBA is a tuple $\mathcal{A} = (Q, \delta, Q_0, F)$ where

- Q is a possibly infinite set of states,
- $\delta \subseteq Q \times \mathcal{B}(AP) \times Q$ is a transition relation such that for each $q \in Q$, there exists only finitely many $b \in \mathcal{B}(AP)$ and $q' \in Q$ such that $(q, b, q') \in \delta$,
- $Q_0 \subseteq Q$ is a finite set of initial states, and
- $F \subseteq Q$ is a set of accepting states.

We write $q \xrightarrow{b} q'$ if $(q, b, q') \in \delta$. We assume a normal form where for any pair of states $q, q' \in Q$, if $q \xrightarrow{b} q'$ and $q \xrightarrow{b'} q'$ then $b = b'$. This normal form can be ensured by merging the transitions $q \xrightarrow{b} q'$ and $q \xrightarrow{b'} q'$ into the single transition $q \xrightarrow{b \vee b'} q'$. For a state $q \in Q$ we define the set of *progressing propositions* to be $\mathrm{Prog}(q) = \{b \in \mathcal{B}(AP) \mid q \xrightarrow{b} q' \text{ for some } q' \in Q \setminus \{q\}\}$, and the *retarding proposition* to be $\mathrm{Ret}(q) = b \in \mathcal{B}(AP)$ such that $q \xrightarrow{b} q$ or ff if no such b exists.

Let $\sigma = A_0 A_1 \ldots \in (2^{AP})^\omega$ be an infinite word. We say that $\mathcal{A}$ *accepts* σ if and only if there exists an infinite sequence of states $q_0 q_1 \ldots$ such that

- $q_0 \in Q_0$,
- for $i \geq 0$, there exists a transition $q_i \xrightarrow{b_i} q_{i+1}$ such that $A_i \models b_i$, and
- for infinitely many $i \geq 0$, $q_i \in F$.

The language of an NBA $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{\sigma \in (2^{AP})^\omega \mid \mathcal{A} \text{ accepts } \sigma\}$.

Automata-based model checking of LTL formulae is possible due to the following theorem relating LTL formulae to NBAs.

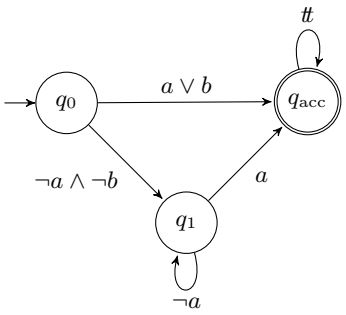


Fig. 2: NBA $\mathcal{A}_\varphi$ where $\varphi = ((\mathsf{G}a)\ \mathsf{U}\ (\mathsf{F}a)) \vee b$, featuring more complex edge propositions.

Theorem 2.4. [2] *Let φ be an LTL formula. There exists an NBA $\mathcal{A}_\varphi$ with finitely many states such that $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$.*

Example 2.5. Figure 2 shows an NBA equivalent to the formula $((\mathsf{G}a)\,\mathsf{U}\,(\mathsf{F}a)) \vee b$. The set of progressing propositions from q_0 is $\text{Prog}(q_0) = \{a \vee b, \neg a \wedge \neg b\}$, and it has no retarding proposition. The set of progressing propositions of q_1 is the singleton set $\text{Prog}(q_1) = \{a\}$, and the retarding proposition is $\text{Ret}(q_1) = \neg a$.

As a corollary to Theorem 2.4, any infinite word σ that satisfies φ must be accepted by $\mathcal{A}_\varphi$ and vice versa. Recall that an LTS $\mathcal{T} = (S, \Sigma, \to, L, s_0)$ satisfies φ if and only if for all $\sigma \in \mathcal{L}(s_0)$ we have $\sigma \models \varphi$. Conversely, if there exists a word $\sigma \in \mathcal{L}(s_0)$ such that $\sigma \not\models \varphi$ then $\mathcal{T} \not\models \varphi$, and σ is accepted by $\mathcal{A}_{\neg\varphi}$. We therefore synchronise $\mathcal{T}$ with $\mathcal{A}_{\neg\varphi}$ and look for counterexamples. We define this synchronisation as follows.

Definition 2.6 (Product). *Let $\mathcal{T} = (S, \Sigma, \to, L, s_0)$ be an LTS and let $\mathcal{A}_\varphi = (Q, \delta, Q_0, F)$ be an NBA. Then the* product *$\mathcal{T} \otimes \mathcal{A}_\varphi = (Q', \delta', Q_0', F')$ of $\mathcal{T}$ and $\mathcal{A}_\varphi$ is an NBA such that*

- *$Q' = S \times Q$,*
- *$\langle s, q\rangle \xrightarrow{tt} \langle s', q'\rangle$ if either $s \to s'$ or s is a deadlock and $s = s'$, and $q \xrightarrow{b} q'$ for some $b \in \mathcal{B}(AP)$ such that $s' \models b$,*
- *$Q_0' = \{\langle s_0, q\rangle \in Q' \mid \exists q_0 \in Q_0 \,.\, q_0 \xrightarrow{b} q$ for some $b \in \mathcal{B}(AP)$ where $s_0 \models b\}$, and*
- *$F' = \{\langle s, q\rangle \in Q' \mid q \in F\}$.*

The following theorem states the key property of the product construction.

Theorem 2.7. [2] *Let $\mathcal{T}$ be an LTS with initial state s_0, φ be an LTL formula and $\mathcal{A}_{\neg\varphi}$ be an NBA such that $\mathcal{L}(A_{\neg\varphi}) = \mathcal{L}(\neg\varphi)$. Then $s_0 \models \varphi$ if and only if $\mathcal{L}(\mathcal{T} \otimes \mathcal{A}_{\neg\varphi}) = \emptyset$.*

In other words, the product construction is suitable for verifying whether $\mathcal{T} \models \varphi$. The model checking procedure consists of constructing the product $\mathcal{T} \otimes \mathcal{A}_{\neg\varphi}$ and searching for accepting runs. In practice this becomes a search for reachable cycles containing accepting states, since such cycles generate infinite accepting runs. This is an under-approximation of the language emptiness check since the state space may be infinite, and as such there may be runs which contain infinitely many accepting states without ever looping. We use a specialised variant of Tarjan's connected component algorithm described in [20] for model checking.

Example 2.8. As described in Example 2.1, the LTS $\mathcal{T}$ depicted in Figure 1a does not satisfy the LTL formula $\mathsf{FG}a$. We now derive the same result using automata-based model checking. Figure 1b shows the NBA $\mathcal{A}_{\neg\mathsf{FG}a}$ equivalent to the LTL formula $\neg\mathsf{FG}a$, and Figure 1c shows the reachable part of the product $\mathcal{T} \otimes \mathcal{A}_{\neg\mathsf{FG}a}$. Since the looping run $(\langle s_0, q_0\rangle\langle s_1, q_0\rangle)^\omega$ visits the accepting state $\langle s_0, q_0\rangle$ infinitely often, we can conclude that $\mathcal{T} \not\models \mathsf{FG}a$, and the run $(s_0 s_1)^\omega$ can be used as a diagnostic counterexample.

2.4 Petri Nets

A Petri net is a 4-tuple $N = (P, T, W, I)$ where

- P is a finite set of places,
- T is a finite set of transitions such that $P \cap T = \emptyset$,
- $W : (P \times T) \cup (T \times P) \to \mathbb{N}^0$ is a set of arc weights, and
- $I : (P \times T) \to \mathbb{N} \cup \{\infty\}$ is a set of inhibitor arc weights.

The semantics of a Petri net $N = (P, T, W, I)$ is given by markings on the form $M : P \to \mathbb{N}^0$ and a firing relation $\to \subseteq \mathcal{M}(N) \times T \times \mathcal{M}(N)$ where $(M, t, M') \in \to$ if for all $p \in P$ we have $M(p) \geq W(p,t)$, $M(p) < I(p,t)$, and $M'(p) = M(p) - W(p,t) + W(t,p)$. We use the following abbreviations.

- $M \xrightarrow{t} M'$ if $(M, t, M') \in \to$,
- $M \xrightarrow{t}$ if there exists M' such that $M \xrightarrow{t} M'$,
- $M \not\xrightarrow{t}$ if not $M \xrightarrow{t}$,
- $M \to$ if there exists t such that $M \xrightarrow{t}$, and
- $M \not\to$ if not $M \to$.

If $M \not\to$ we say that M is deadlocked. We denote the set of *successors* of M as $\mathrm{suc}(M) = \{M' \in \mathcal{M}(N) \mid \exists t \in T \,.\, M \xrightarrow{t} M'\}$. We write $\mathcal{M}(N)$ to denote the set of all markings of Petri net N. If $M(p) = n$ for some marking M we say that p has n *tokens* in M. For $x \in P \cup T$, we write ${}^\bullet x$ to mean $\{y \in T \cup P \mid W(y,x) > 0\}$, called the preset, and $x^\bullet$ to mean $\{y \in T \cup P \mid W(x,y) > 0\}$, called the postset. We straightforwardly extend this to sets $X \subseteq T$ and $X \subseteq P$ such that ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$. For a place $p \in P$ we define the *increasing preset* of p as ${}^+p = \{t \in {}^\bullet p \mid W(t,p) > W(p,t)\}$, and the *decreasing*

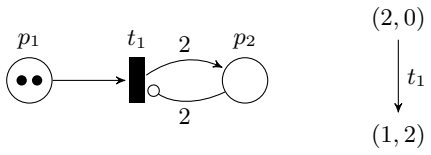


Fig. 3: Example Petri net and its state space.

postset of p as $p^- = \{t \in p^\bullet \mid W(t,p) < W(p,t)\}$. The *inhibitor postset* of $p \in P$ is $p^\circ = \{t \in T \mid I(p,t) < \infty\}$ and the *inhibitor preset* of $t \in T$ is ${}^\circ t = \{p \in P \mid I(p,t) < \infty\}$

A net $N = (P, T, W, I)$ gives rise to an LTS $\mathcal{T} = (\mathcal{M}(N), T, \to, L, M_0)$ where M_0 is a designated initial marking. We call $\mathcal{T}$ the *LTS of* N. The set AP of atomic propositions is formed using the grammar

$$a ::= t \mid e_1 \bowtie e_2$$
$$e ::= p \mid c \mid e_1 \oplus e_2$$

where $t \in T$, $p \in P$, $c \in \mathbb{N}^0$, $\bowtie \in \{<, \leq, \neq, =, >, \geq\}$, and $\oplus \in \{\cdot, +, -\}$. An atomic proposition in which the predicate p never occurs is called a fireability proposition, and an atomic proposition in which the predicate t never occurs is called a cardinality proposition. Likewise, an LTL formula in which there only appears cardinality propositions is called a cardinality formula, and a formula in which there only appears fireability propositions is called a fireability formula. Given a Petri net $N = (P, T, W, I)$, the satisfaction of a marking $M \in \mathcal{M}(N)$ of an atomic proposition $a \in AP$ is given by

$$\begin{aligned} M \models t &\iff M \xrightarrow{t} \\ M \models e_1 \bowtie e_2 &\iff \mathrm{eval}_M(e_1) \bowtie \mathrm{eval}_M(e_2) \\ \mathrm{eval}_M(p) &= M(p) \\ \mathrm{eval}_M(c) &= c \\ \mathrm{eval}_M(e_1 \oplus e_2) &= \mathrm{eval}_M(e_1) \oplus \mathrm{eval}_M(e_2) \ . \end{aligned}$$

For $t \in T$, the fireability proposition t can be rewritten into the cardinality proposition $\bigwedge_{p \in {}^\bullet t}(p \geq W(p,t)) \wedge \bigwedge_{p \in {}^\circ t}(p < I(p,t))$ requiring that all pre-places of t are sufficiently marked and no inhibitor arc of t is sufficiently marked. In the following we assume that all propositions are cardinality propositions.

Example 2.9. Figure 3 demonstrates the graphical representation of a Petri net $N = (P, T, W, I)$. The set of places is $P = \{p_1, p_2\}$ and the set of transitions is $T = \{t_1\}$. The function W is $W(p_1, t_1) = 1$, $W(t_1, p_2) = 2$ and for all other $x, y \in P \cup T$ we have $W(x, y) = 0$. The function I is $I(p_2, t_1) = 2$ and for all other $p \in P$ and $t \in T$ we have $I(p, t) = \infty$. The initial marking is $M_0(p_1) = 2$ and $M_0(p_2) = 0$. In the initial marking t_1 is enabled since p_1 is sufficiently marked, i.e. $M_0(p_1) \geq W(p_1, t_1)$, and p_2 is insufficiently marked for the inhibitor arc, i.e.

$M_0(p_2) < I(p_2, t_1)$. Firing t_1 yields $M_0 \xrightarrow{t_1} M$ where $M(p_1) = 1$ and $M(p_2) = 2$ since the arc $W(t_1, p_2)$ produces 2 tokens. The new marking M is deadlocked since although $M(p_1) \geq W(p_1, t_1)$, we have $M(p_2) \geq I(p_1, t_1)$, inhibiting the firing of t_1.

3 Partial Order Reductions

Partial order reductions are techniques that address the state space explosion problem by reducing the number of interleavings of concurrent actions. When a state has many actions that are concurrent, the state space can contain states for any permutation of these actions, even when each permutation leads to the same state. Thus choosing only a representative permutation can result in exponential reductions in the size of the state space (for examples of this see e.g. [44,46]). We consider techniques that accomplish this by selecting in each state a subset of available actions for exploration. Such techniques include ample sets [32], persistent sets [21], and stubborn sets [44,45]. Of these we consider the stubborn set technique.

3.1 Classic Stubborn Set Method

We will now present the classical LTL$_\mathsf{X}$-preserving stubborn set method, which is due to Valmari [44,45].

Definition 3.1 (Reduction). [44] *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS. A* reduction *is a function $St : S \rightarrow 2^\Sigma$, and the* reduced LTS *of $\mathcal{T}$ given by St is $\mathcal{T}_{St} = (S, \Sigma, \rightarrow_{St}, L, s_0)$ where $s \xrightarrow{\alpha}_{St} s'$ iff $s \xrightarrow{\alpha} s'$ and $\alpha \in St(s)$.*

The goal of the method is to find a suitable reduction St such that for any LTL$_\mathsf{X}$ formula φ, $\mathcal{T} \models \varphi$ iff $\mathcal{T}_{St} \models \varphi$, i.e. the reduction preserves LTL$_\mathsf{X}$.

Definition 3.2 (COM and KEY). *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS and let St be a reduction on $\mathcal{T}$. The commutation rule **COM** and key transition rule **KEY** are defined as follows for all states $s \in S$:*

COM *For all $s' \in S$, if $s \xrightarrow{\alpha_1\alpha_2\ldots\alpha_n\alpha} s'$ where $\alpha \in St(s)$ and $\alpha_i \notin St(s)$ then $s \xrightarrow{\alpha\alpha_1\alpha_2\ldots\alpha_n} s'$.*

KEY *If $\mathrm{en}(s) \neq \emptyset$, then there is some* key action *$\alpha_{\mathrm{key}} \in St(s)$ such that for all $n \geq 0$, if $\alpha_1, \ldots, \alpha_n \notin St(s)$ and $s \xrightarrow{\alpha_1\ldots\alpha_n} s_n$ then $s_n \xrightarrow{\alpha_{\mathrm{key}}}$, otherwise $St(s) = \Sigma$.*

The **COM** rule forms the core of stubborn set methods. It asserts that a sequence consisting of non-stubborn actions and a stubborn action will lead to the same destination state regardless of when the stubborn action is taken. By exploring only the execution where the stubborn action is taken first, we reduce the number of interleavings in the state space. The rule **KEY** ensures that if there is some enabled action in s then $St(s)$ also contains some enabled action

(let $n = 0$). This rule functions as a starting point for stubborn set generation. Normally, the main property of these rules is a deadlock preservation theorem—any deadlock state in the full state space is reachable and a deadlock state in the reduced state space, and vice versa [43]. However, since we assume deadlock-free semantics this property is not relevant to us.

To preserve LTL_X it is important to ensure that a run π in the full state space has an equivalent run π' in the reduced state space such that the induced words σ_π and $\sigma_{\pi'}$ are stuttering equivalent. To ensure this, we require a notion of visible transitions.

Definition 3.3 (Visibility). *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS and let φ be an LTL formula. We say that $\alpha \in \Sigma$ is a* visible action *wrt. φ if there exists states $s, s' \in S$ such that $s \xrightarrow{\alpha} s'$ and for some atomic proposition a mentioned in φ, either $s \not\models a \wedge s' \models a$ or $s \models a \wedge s' \not\models a$. A set $\Sigma_\varphi \subseteq \Sigma$ is a* visible set *if for all actions $\alpha \in \Sigma$ that are visible wrt. φ, $\alpha \in \Sigma_\varphi$.*

An action is visible if it can change the truth value of an atomic proposition. The set Σ_φ is not required to be exactly the set of visible actions for ease of implementation. The following rules ensure that the reduced state space is stutter-trace equivalent with the full state space.

Definition 3.4 (VIS and IGN). *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS, let St be a reduction on $\mathcal{T}$ and let φ be an LTL formula. The visibility rule **VIS(φ)** and non-ignoring rule **IGN(φ)** are defined as follows for all states $s \in S$.*

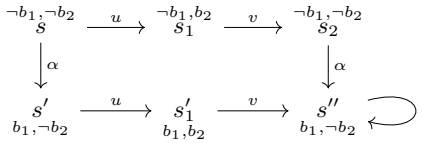
VIS(φ) *If $St(s)$ contains an enabled, visible action $\alpha \in \Sigma_\varphi$, then $St(s) = \Sigma$.*
IGN(φ) *If $s_0 \xrightarrow{\alpha_1}_{St} s_1 \xrightarrow{\alpha_2}_{St} \ldots$ is an infinite execution for some $s_0, s_1, \ldots \in S$, then for each visible action $\alpha_{\text{vis}} \in \Sigma_\varphi$ there must be i such that $\alpha_{\text{vis}} \in St(s_i)$.*

VIS(φ) ensures that if $\alpha, \beta \in \Sigma_\varphi$, i.e. are contained in a visible set, and both $s \xrightarrow{\alpha\beta} s'$ and $s \xrightarrow{\beta\alpha} s'$, then both interleavings are explored. This is necessary since both orderings are potentially important with respect to φ. The rule **IGN(φ)** ensures, roughly speaking, that visible actions are not perpetually ignored, a situation that could otherwise occur if e.g. $St(s) = \{\alpha_1\}$ and $s \xrightarrow{\alpha_1} s$. In this case, **IGN($\varphi$)** would ensure that if some action α_2 would result in progress then it is also taken. **IGN(φ)** is implemented by including all $\alpha_\text{v} \in \Sigma_\varphi$ in $St(s)$ when exploring an edge $s \xrightarrow{\alpha} s'$ where s' already occurs in the current depth-first search path, i.e. when the edge closes a cycle in the state space.

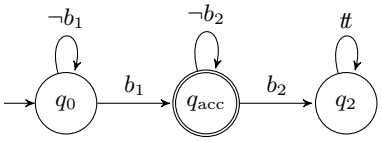
By using rules **COM**, **KEY**, **VIS(φ)**, and **IGN(φ)**, we obtain the classic LTL_X-preserving stubborn set method, the main result of which is the following theorem.

Theorem 3.5. [44] *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS, φ be an LTL_X formula, and St be a reduction satisfying **COM**, **KEY**, **VIS(φ)**, and **IGN(φ)**. Then the reduced LTS $\mathcal{T}_{St} = (S, \Sigma, \rightarrow_{St}, L, s_0)$ satisfies φ iff $\mathcal{T}$ satisfies φ.*

The original proof of the theorem demonstrated that the relative ordering of visible transitions is preserved in the reduced state space, which means that



(a) Commutation diagram of an LTS. For each state the satisfied propositions are denoted.



(b) NBA where q_0 is a reachability state.

Fig. 4: Motivation for safe actions. If a reduced state space causes the state q_{acc} to be reached before $\neg b_2$ is satisfied, accepting runs may be lost due to overshooting accepting states.

any word in the reduced state space is stuttering equivalent with some word in the full state space. However, the X operator can distinguish between stuttering equivalent words, meaning that the theorem is only enough to guarantee LTL_{X} preservation.

3.2 Reachability Stubborn Set Method

The classical approach to stubborn sets for LTL_{X}, presented in Section 3.1, focuses on the entire LTL formula. In this section we present a stubborn set approach that depends on local information in the NBA and is applicable to full LTL. The idea is to apply the reachability-perserving stubborn set method from [34,7] in non-accepting NBA states $q \notin F$ where for all $A \in 2^{AP}$ there is a successor $q \xrightarrow{b} q'$ where $A \models b$. To ensure this we look at states q where given the retarding proposition $\text{Ret}(q)$ and progressing propositions $\text{Prog}(q)$ we have $\text{Ret}(q) \vee \bigvee_{b \in \text{Prog}(q)} b = tt$. With this requirement any state s in the LTS will have some successor in the NBA, so the only important aspect in q is to find a state satisfying some progressing proposition. This motivates us to apply the reachability-preserving stubborn set method with states satisfying some $b \in \text{Prog}(q)$ as goal states.

We now define the method formally. We call states in which we can use the method reachability states.

Definition 3.6 (Reachability states). *Let $\mathcal{A} = (Q, \delta, Q_0, F)$ be an NBA. The set* $\text{Reach}(\mathcal{A})$ *of* reachability states *in $\mathcal{A}$ is the set of all $q \in Q$ such that*

- *$q \notin F$ and*
- *$\left(\bigvee_{b \in \text{Prog}(q)} b\right) \vee \text{Ret}(q) = tt$ where* $\text{Prog}(q)$ *is the set of progressing propositions of q and* $\text{Ret}(q)$ *is the retarding proposition of q.*

Remark 3.7. Determining whether the disjunction of the progressing propositions and the retarding propositions is a tautology can be done efficiently in practice using BDDs, which is already the format we use for propositions.

We need to be careful of when we perform the reduction. This example illustrates why we cannot always reduce if we are in a reachability state.

Example 3.8. Consider the NBA and LTS in Figure 4. Figure 4a shows part of the LTS $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$, where the propositions satisfied by each state are noted beside them. Furthermore $\alpha \in \Sigma$ and $u, v \in \Sigma^*$. The NBA state q_0 is a reachability state since $\neg b_1 \vee b_1 = tt$. We consider the product state $\langle s, q_0 \rangle$. If we follow the upper transition sequence in Figure 4a we get the product state sequence $\langle s, q_0 \rangle \xrightarrow{u} \langle s_1, q_0 \rangle \xrightarrow{v} \langle s_2, q_0 \rangle \xrightarrow{\alpha} \langle s'', q_{\text{acc}} \rangle$. Since we can loop in s'' indefinitely, this is an accepting run. However since q_0 is a reachability state we try to apply the reachability stubborn method in s. If we determine the action α to be stubborn and that none of the actions in u and v are stubborn, then we commute α to the start of the sequence and follow the lower transition sequence. This yields the product state sequence $\langle s, q_0 \rangle \xrightarrow{\alpha} \langle s', q_{\text{acc}} \rangle \xrightarrow{u} \langle s'_1, q_2 \rangle \xrightarrow{v} \langle s'', q_2 \rangle$. This run is not accepting, and thus the commutation is not correct.

To alleviate this problem we introduce safe actions.

Definition 3.9 (Safe action). *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS and let $\mathcal{A} = (Q, \delta, Q_0, F)$ be an NBA. For a state $s \in S$ and proposition $b \in \mathcal{B}(AP)$, a set* $\text{Safe}(s, b) \subseteq \Sigma$ *is* safe *wrt. b if for all $\alpha \in \text{Safe}(s, b)$ and $w \in \text{Safe}(s, b)^*$, if $s \xrightarrow{w} s'$, $s \xrightarrow{\alpha w} s''$, and $s' \not\models b$, then $s'' \not\models b$. For states $s \in S$ and $q \in Q$, a set* $\text{Safe}(s, q) \subseteq \Sigma$ *is* safe *wrt. q if for all progressing propositions $b \in \text{Prog}(q)$,* $\text{Safe}(s, b) \subseteq \text{Safe}(s, q)$.

The property of a safe action α is that if we in a state s fire a sequence of safe actions w after which we do not satisfy b then firing α followed by w will still not satisfy b. In particular, when w is empty, if $s \not\models b$ and $s \xrightarrow{\alpha} s'$, then $s' \not\models b$. The idea of a safe action is inspired by [5] but adapted for our purposes and made looser. In particular, we do not require of a safe action α and transition sequence w that w can consist of actions different from α, but restrict w to only concern itself with explicitly safe actions.

In this new method the reductions are no longer only dependent on the current LTS state, we also need to know in what NBA state we are at the moment. To handle this we now define reductions on the product state space.

Definition 3.10 (Product reduction). *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS and $\mathcal{A} = (Q, \delta, Q_0, F)$ be an NBA. Then the* product reduction *is a function $St : S \times Q \rightarrow 2^\Sigma$ and the* reduced product state space *of $\mathcal{T} \otimes \mathcal{A}$ given by St is $\mathcal{T} \otimes_{St} \mathcal{A}$ where $\langle s, q \rangle \rightarrow_{St} \langle s', q' \rangle$ if and only if $\langle s, q \rangle \rightarrow \langle s', q' \rangle$ and there exists an $\alpha \in St(s, q)$ such that $s \xrightarrow{\alpha} s'$.*

The stubborn set method is based on the following rules.

Definition 3.11 (COM, R, and SAFE). *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS, $\mathcal{A} = (Q, \delta, Q_0, F)$ be an NBA and let $St : S \times Q \rightarrow 2^\Sigma$ be a product reduction. The axioms **COM**, **R**, and **SAFE** are defined as, for all $s \in S$ and all $q \in Q$*

COM *If* $\alpha \in St(s,q)$ *and* $\alpha_1, \alpha_2, \ldots, \alpha_n \in \overline{St(s,q)}^*$, *if* $s \xrightarrow{\alpha_1 \ldots \alpha_n \alpha} s'$ *then* $s \xrightarrow{\alpha \alpha_1 \ldots \alpha_n} s'$.

R *If* $\alpha_1 \ldots \alpha_n \in \overline{St(s,q)}^*$ *and for all* $b \in \mathrm{Prog}(q)$ *we have* $s \not\models b$ *then* $s \xrightarrow{\alpha_1 \ldots \alpha_n} s'$ *implies that for all* $b \in \mathrm{Prog}(q)$ *we have* $s' \not\models b$.

SAFE *For all* $s \in S$, *either* $q \in \mathrm{Reach}(\mathcal{A})$, $\mathrm{en}(s) \cap St(s,q) \subseteq \mathrm{Safe}(s,q)$, *and for all progressing propositions* $b \in \mathrm{Prog}(q)$ *we have* $s \not\models b$, *or* $St(s,q) = \Sigma$.

Rules **COM** and **R** are adapted from standard reachability-preserving stubborn set methods, see [34,7]. **R** asserts that no progressing proposition is satisfied without first taking a stubborn action, ensuring that states satisfying progressing propositions are preserved. The rule **SAFE** asserts that we cannot reduce if some unsafe action is enabled or if some progressing proposition is already satisfied, ensuring that states in the NBA are reached at the right times so we avoid the problem shown in Figure 4. It also ensures that no reduction is done in accepting states since the point of the method is to reach accepting states, and it is not safe to reduce there since anything can lead to an accepting cycle.

We are now ready to prove the correctness of this stubborn set method for use in LTL model checking, not just $\mathrm{LTL}_{\mathsf{X}}$.

Theorem 3.12. *Let* $\mathcal{T} = (S, \Sigma, \to, L, s_0)$ *be an LTS,* $\mathcal{A} = (Q, \delta, Q_0, F)$ *be an NBA,* $St : S \times Q \to 2^{\Sigma}$ *be a product reduction satisfying* ***COM***, ***R***, *and* ***SAFE***, *and* $\mathcal{T} \otimes_{St} \mathcal{A}$ *be the reduced state space of* $\mathcal{T} \otimes \mathcal{A}$ *given by* St. *Then* $\mathcal{T} \otimes \mathcal{A}$ *contains an accepting run if and only if* $\mathcal{T} \otimes_{St} \mathcal{A}$ *contains an accepting run.*

Proof. " $\Longleftarrow$ ": Let $\pi = \langle s_0, q_0 \rangle \to_{St} \ldots$ be some accepting run in the reduced state space. If π contains no state $\langle s_i, q_i \rangle$ where s_i is a deadlock state, then π must also be an accepting run in the full state space since St does not add any new actions at any point. If π contains a state $\langle s_i, q_i \rangle$ where s_i is a deadlock state then since the run is accepting, either q_i is accepting or some progressing proposition $b \in \mathrm{Prog}(q_i)$ to q_i is satisfied by s_i. In both cases, by rule **SAFE** no reduction is performed, so s_i is also a deadlock state in the full state space, hence π is also an accepting run in the full state space.

" $\Longrightarrow$ ": Assume for sake of contradiction that the full state space contains an accepting run $\pi = \langle s_0, q_0 \rangle \ldots$ but the reduced state space does not. Then there must be a partitioning of $\pi = \pi_1 \pi_2$ such that π_1 is the longest prefix of π that is executable in the reduced state space. Let $\langle s_1, q_1 \rangle$ be the last state in π_1. Since π_2 is not executable, we have $St(s_1, q_1) \neq \Sigma$, so there is some reduction. By **SAFE** we therefore know that $q_1 \notin F$, so π_2 must change NBA state at some point, and no progressing proposition of q_1 is satisfied by s_1. Let $w \in \Sigma^*$ be the sequence of actions of length $i-1$ such that for each action $\alpha_k \in w$ in the sequence $\langle s_1, q_1 \rangle \xrightarrow{\alpha_1} \langle s_2, q_2 \rangle \xrightarrow{\alpha_{i-1}} \langle s_i, q_i \rangle$ is a prefix of π_2, and $q_i \neq q_1$ and $q_j = q_1$ for all $1 \leq j < i$. Since $q_i \neq q_1$, some s_j must satisfy a progressing proposition b where $q_1 \xrightarrow{b} q_i$, so by **R** there must be a stubborn action $\alpha \in St(s_1, q_1)$ in w. Let $w = u \alpha v$ such that $u \in \overline{St(s_1, q_1)}^*$ and $\alpha \in St(s_1, q_1)$. By **COM** $s_1 \xrightarrow{u\alpha} s' \xrightarrow{v} s_i$ implies $s_1 \xrightarrow{\alpha} s_1' \xrightarrow{u} s' \xrightarrow{v} s_i$. Additionally, since $St(s_1, q_1) \neq \Sigma$ and α is an enabled, stubborn action, by **SAFE** α is a safe action, so by Definition 3.9

s'_1 does not satisfy any progressing propositions. Since u does not contain any stubborn actions and s'_1 does not satisfy any progressing proposition, by **R** no intermediate state along the run $s'_1 \xrightarrow{u} s'$ satisfies any progressing proposition. Hence we have $\langle s_1, q_1\rangle \rightarrow_{St} \langle s'_1, q_1\rangle \rightarrow^*_{St} \langle s', q_1\rangle$, an extension of the executable part of π_1 in the reduced state space. From $\langle s', q_1\rangle$, the execution can continue to follow π_2 by repeating this argument as necessary.

We thus obtain an accepting run in the reduced state space, contradicting the assumption that there was no accepting run in the reduced state space. □

3.3 Mixed Stubborn Set Method

The new stubborn set method presented above relies on the notion of reachability states, where in all states that are not reachability states no reduction is performed. In this section we introduce a mixed stubborn set method that uses the reachability stubborn set method when possible and falls back to the classic method outside reachability states.

Definition 3.13 (Mixed reduction). *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS, let $\mathcal{A}_{\neg\varphi} = (Q, \delta, Q_0, F)$ be an NBA corresponding to LTL$_\mathsf{X}$ formula φ. Then given a reduction $St_{\text{classic}} : S \rightarrow 2^\Sigma$ satisfying **COM**, **KEY**, **VIS**(φ), and **IGN**(φ), and a product reduction $St_{\text{reach}} : S \times Q \rightarrow 2^\Sigma$ satisfying **COM**, **R**, and **SAFE**, the* mixed reduction *is the product reduction such that*

$$St(s,q) = \begin{cases} St_{\text{reach}}(s,q) & \text{if } q \in \text{Reach}(\mathcal{A}) \text{ and for no } b \in \text{Prog}(q) s \models b \\ St_{\text{classic}}(s) & \text{otherwise} \end{cases}.$$

We now prove that this reduction is LTL$_\mathsf{X}$ preserving.

Theorem 3.14. *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS, let φ be an LTL$_\mathsf{X}$ formula, let $\mathcal{A}_{\neg\varphi} = (Q, \delta, Q_0, F)$ be an NBA corresponding to $\neg\varphi$, let $St_{\text{classic}} : S \rightarrow 2^\Sigma$ be a reduction satisfying the rules **COM**, **KEY**, **VIS**(φ), and **IGN**(φ), let $St_{\text{reach}} : S \times Q \rightarrow 2^\Sigma$ be a product reduction satisfying the rules **COM**, **R**, and **SAFE**, and let $St : S \times Q \rightarrow 2^\Sigma$ be the mixed reduction of St_{classic} and St_{reach}. Then $\mathcal{T} \otimes \mathcal{A}_{\neg\varphi}$ contains an accepting run if and only if $\mathcal{T} \otimes_{St} \mathcal{A}_{\neg\varphi}$ contains an accepting run.*

Proof. $\Longleftarrow$ Let $\pi = \langle s_0, q_0\rangle \rightarrow_{St} \ldots$ be some accepting run in the reduced state space. If π contains no state $\langle s_i, q_i\rangle$ where s_i is a deadlock state in the reduced state space, then π must also be an accepting run in the full state space since St adds no new actions. If π does contain a $\langle s_i, q_i\rangle$ where s_i is a deadlock state and St_{reach} was not used in $\langle s_i, q_i\rangle$ then by **KEY** s_i is also a deadlock state in the full state space. If St_{reach} was used in $\langle s_i, q_i\rangle$, then since π is accepting either q_i is accepting or $s_i \models b$ for some progressing proposition $b \in \text{Prog}(q_i)$. In both cases, **SAFE** prevents any reduction, so $St(s_i, q_i) = \Sigma$, hence $\langle s_i, q_i\rangle$ is also a deadlock in the full state space.

$\Longrightarrow$ Assume for the sake of contradiction that the full state space contains an accepting run but the reduced state space does not contain any accepting

runs. Theorem 3.5 tells us that a state space $\mathcal{T}_{St_{\text{classic}}}$ reduced by St_{classic} satisfies φ if and only if $\mathcal{T}$ satisfies φ. As a corollary to this, $\mathcal{T}_{St_{\text{classic}}} \otimes \mathcal{A}_{\neg\varphi}$ has an accepting run if and only if $\mathcal{T} \otimes \mathcal{A}_{\neg\varphi}$ has an accepting run. Let π be an accepting run in $\mathcal{T}_{St_{\text{classic}}} \otimes \mathcal{A}_{\neg\varphi}$, and assume for the sake of contradiction that $\mathcal{T} \otimes_{St} \mathcal{A}_{\neg\varphi}$ does not contain any accepting run. Let $\pi = \pi_1\pi_2$ such that π_1 is the longest prefix of π executable in $\mathcal{T} \otimes_{St} \mathcal{A}_{\neg\varphi}$, and let $\langle s, q\rangle$ be the last state in π_1. Since π_2 is not executable, $\langle s, q\rangle$ must be a state in which St_{reach} is used, so q is a reachability state and thus not accepting. Therefore there must be some fragment $\langle s, q\rangle \rightarrow_{St_{\text{classic}}} \langle s_1, q_1\rangle \rightarrow_{St_{\text{classic}}} \ldots \rightarrow_{St_{\text{classic}}} \langle s_n, q_n\rangle$ of π such that $q_i = q$ for $1 \leq i < n$ and $q_n \neq q$. By Definition 3.13, no progressing proposition of q is satisfied, so since $q_n \neq q$ by **R** some stubborn transition must have been fired in the fragment. Let $\langle s, q\rangle \xrightarrow{w} \langle s_n, q_n\rangle$ such that $w = u\alpha v$ where $u \in \overline{St_{\text{reach}}(s, q)}^*$, $\alpha \in St_{\text{reach}}(s, q)$, and $\langle s, q\rangle \xrightarrow{u\alpha}_{St_{\text{classic}}} \langle s', q\rangle \xrightarrow{v}_{St_{\text{classic}}} \langle s_n, q_n\rangle$. By **COM** we also have $s \xrightarrow{\alpha} s'' \xrightarrow{u} s'$. Since $\alpha \in \text{en}(s)$ it must also be a safe action, so s'' does not satisfy any progressing proposition, and since u does not contain any stubborn actions, no intermediate state in the run satisfies any progressing proposition. Hence we get $\langle s, q\rangle \xrightarrow{\alpha u}_{St_{\text{reach}}} \langle s', q\rangle$, and since we remain in q the entire time and no progressing proposition is satisfied, we get, by Definition 3.13, $\langle s, q\rangle \xrightarrow{\alpha u}_{St} \langle s', q\rangle$. From $\langle s', q\rangle$ we can continue executing π_2, repeating this argument as necessary. We thus get an accepting run in $\mathcal{T} \otimes_{St} \mathcal{A}_{\neg\varphi}$ as a consequence of an accepting run in the original state space $\mathcal{T} \otimes \mathcal{A}_{\neg\varphi}$. □

3.4 Stubborn Sets for Petri nets

We now present a syntax-driven method for computing stubborn sets for a marking in a Petri net. We start by defining a COM-saturated set, which implements the **COM** rule, and the set of increasing or decreasing transitions wrt. an expression.

Definition 3.15 describes necessary conditions for ensuring the rule **COM**. These conditions are due to [7] and describes a set of transitions that can be safely commuted with any transitions not in the set.

Definition 3.15 (COM-saturation). *Let $N = (P, T, W, I)$ be a Petri net and $M \in \mathcal{M}(N)$ be a marking. We say that a set $T' \subseteq T$ is* COM-saturated in M *if*

1. *For all $t \in T'$, if $M \xrightarrow{t}$ then*
   - *for all $p \in {}^\bullet t$ where $t \in p^-$ we have $p^\bullet \subseteq T'$, and*
   - *for all $p \in t^\bullet$ where $t \in {}^+p$ we have $p^\circ \subseteq T'$.*
2. *For all $t \in T'$, if $M \not\xrightarrow{t}$ then*
   - *there exists a $p \in {}^\bullet t$ such that $M(p) < W(p, t)$ and ${}^+p \subseteq T'$, or*
   - *there exists a $p \in {}^\circ t$ such that $M(p) \geq I(p, t)$ and $p^- \subseteq T'$.*

The intuition of the conditions in Definition 3.15 is as follows. According to Condition 1, if t is enabled and decreases the number of tokens in the place $p \in {}^\bullet t$, then any t' that has p as a pre-place, i.e. $p \in {}^\bullet t \cap {}^\bullet t'$, is in conflict with t since t can disable t'. Likewise if t increases the number of tokens in a place p with

outgoing inhibitor arcs, the transitions inhibited by p are also in conflict with t. Condition 2 states that a transition t' that could cause a disabled transition t to become enabled cannot be commuted with t, as t is then dependent on t'. This is the case if either t' adds tokens to some insufficiently marked pre-place $p \in {}^\bullet t$ or if t' removes tokens from a sufficiently marked place $p \in {}^\circ t$ that has an inhibitor arc to t. The following lemma states that transitions from a COM-saturated set T' can be commuted with any sequence of transitions that are not in T', or in other words that T' satisfies the **COM** axiom of stubborn set theory.

Lemma 3.16. *Let $N = (P, T, W, I)$ be a Petri net, let $M \in \mathcal{M}(N)$ be a marking and let $T' \subseteq T$ be COM-saturated in M. Then for any $t \in T'$, $t_1, \ldots, t_n \in T \setminus T'$ and any marking $M' \in \mathcal{M}(N)$, if $M \xrightarrow{t_1 \ldots t_n t} M'$ then $M \xrightarrow{t t_1 \ldots t_n} M'$.*

Proof. Assume $M \xrightarrow{t_1 \ldots t_n} M_n \xrightarrow{t} M'$ for $t \in T'$ and $t_1, \ldots, t_n \in T \setminus T'$, and assume for sake of contradiction that $M \not\xrightarrow{t}$. Then there must be (i) some $p \in {}^\bullet t$ such $M(p) < W(p,t)$ and ${}^+p \subseteq T'$, or (ii) some $p \in {}^\circ t$ such that $M(p) \geq I(p,t)$ and $p^- \subseteq T'$.

- **(i)** since $M_n \xrightarrow{t}$, for all p where $M(p) < W(p,t)$ some t_i must have increased the number of tokens in p, i.e. $t_i \in {}^+p$ for some $i < n$. However, by Condition 2 we have ${}^+p \subseteq T'$, so $t_i \notin {}^+p$, hence no t_i increased the number of tokens in p. Therefore since $M_n \xrightarrow{t}$, also $M \xrightarrow{t}$, a contradiction.
- **(ii)** since $M_n \xrightarrow{t}$ for all p where $M(p) \geq I(p,t)$ some t_i must have decreased the number of tokens in p, i.e. $t_i \in p^-$. However, by Condition 2 we have $p^- \subseteq T'$, so $t_i \notin p^-$, hence no t_i decreased the number of tokens in p. Therefore since $M_n \xrightarrow{t}$, also $M \xrightarrow{t}$, a contradiction.

Thus we conclude that $M \xrightarrow{t}$.

Now let $M \xrightarrow{t} M_t$. We now show that $M_t \xrightarrow{t_1 \ldots t_n} M'$. Assume for sake of contradiction that some t_i cannot be fired. Then there must either be a place $p \in {}^\bullet t_i$ such that $M(p) > M_t(p)$ and thus $t \in p^-$, or some $p \in {}^\circ t_i$ such that $M(p) < M_t(p)$ and thus $t \in {}^+p$. In the former case, by Condition 1 $p^\bullet \subseteq T'$, so since $t_i \notin T'$ we have $t_i \notin p^\bullet$, and in the latter case by Condition 1 $p^\circ \subseteq T'$, so since $t_i \notin T'$ we have $t_i \notin p^\circ$. In both cases t cannot have disabled t_i, so t_i can be fired, a contradiction.

In conclusion, both $M \xrightarrow{t} M_t$ and $M_t \xrightarrow{t_1 \ldots t_n} M'$ is possible. □

The conditions in Definition 3.15 give rise to a straightforward closure algorithm that starting from some set T' iteratively includes additional transitions as required by Conditions 1 and 2. Because of Lemma 3.16, this algorithm can be used to ensure **COM** starting from any initial set of transitions.

The following definition of increasing and decreasing transitions of an expression is relevant to constructing visible and safe sets and to rule **R**.

Definition 3.17 (Increasing and Decreasing transitions).
Let $N = (P, T, W, I)$ be a Petri net and let $e \in E$ be an expression. The set of

increasing transitions $\mathrm{incr}(e)$ *and* decreasing transitions $\mathrm{decr}(e)$ *are defined as follows.*

$$\begin{aligned}
\mathrm{incr}(p) &= {}^{+}p \\
\mathrm{decr}(p) &= p^{-} \\
\mathrm{incr}(c) &= \mathrm{decr}(c) = \emptyset \\
\mathrm{incr}(e_1 + e_2) &= \mathrm{incr}(e_1) \cup \mathrm{incr}(e_2) \\
\mathrm{decr}(e_1 + e_2) &= \mathrm{decr}(e_1) \cup \mathrm{decr}(e_2) \\
\mathrm{incr}(e_1 - e_2) &= \mathrm{incr}(e_1) \cup \mathrm{decr}(e_2) \\
\mathrm{decr}(e_1 - e_2) &= \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2) \\
\mathrm{decr}(e_1 \cdot e_2) = \mathrm{incr}(e_1 \cdot e_2) &= \mathrm{incr}(e_1) \cup \mathrm{incr}(e_2) \cup \mathrm{decr}(e_1) \cup \mathrm{decr}(e_2)
\end{aligned}$$

The sets $\mathrm{incr}(e)$ and $\mathrm{decr}(e)$ define the sets of transitions that increase or decrease the value of an expression $e \in E$. They have the property that if the valuation of e increased after firing some transition t then $t \in \mathrm{incr}(e)$, and if the valuation decreased after firing t then $t \in \mathrm{decr}(e)$..

Lemma 3.18. [7] *Let $N = (P, T, W, I)$ be a Petri net, let $e \in E$ be an expression, and let $M, M' \in \mathcal{M}(N)$ be markings such that $M \xrightarrow{t_1 \dots t_n} M'$ for $t_1, \dots, t_n \in T$. If $\mathrm{eval}_M(e) < \mathrm{eval}_{M'}(e)$ then for some i, $t_i \in \mathrm{incr}(e)$, and if $\mathrm{eval}_M(e) > \mathrm{eval}_{M'}(e)$ then for some i, $t_i \in \mathrm{decr}(e)$.*

Corollary 3.19. *Let $N = (P, T, W, I)$ be a Petri net, $M, M' \in \mathcal{M}(N)$ be markings such that $M \xrightarrow{t} M'$ for $t \in T$, and let $a \in AP$ be an atomic proposition. Then either $M \models a$ iff $M' \models a$ or for some expression e mentioned in a, $t \in \mathrm{incr}(e) \cup \mathrm{decr}(e)$.*

Proof. If $a = tt$ then we trivially have $M \models a \iff M' \models a$.

Let $a = e_1 \bowtie e_2$. If for each $i \in \{1, 2\}$ we have $\mathrm{eval}_M(e_i) = \mathrm{eval}_{M'}(e_i)$ then clearly $M \models a$ iff $M' \models a$. Otherwise for some i either $\mathrm{eval}_M(e_i) < \mathrm{eval}_{M'}(e_i)$ or $\mathrm{eval}_M(e_i) > \mathrm{eval}_{M'}(e_i)$. In the former case, by Lemma 3.18 we have $t \in \mathrm{incr}(e_i)$, and in the latter case $t \in \mathrm{decr}(e_i)$. □

We now show implementations of each stubborn set method wrt. Petri nets.

Classic Stubborn Set Method We now show how to implement the classic stubborn set method on Petri nets. The only missing part is a definition of visible sets of an LTL formula φ that is suitable for the rule **VIS(φ)**. The following is a suitable definition.

Definition 3.20 (Visible transitions). *Let $N = (P, T, W, I)$ be a Petri net, and let φ be an LTL formula. The set* $\mathrm{vis}(\varphi) \subseteq T$ *of* visible transitions *of φ is*

defined inductively as

$$\mathrm{vis}(\varphi) = \begin{cases} \mathrm{incr}(e_1) \cup \mathrm{decr}(e_1) \cup & \text{if } \varphi = e_1 \bowtie e_2 \text{ where } e_1, e_2 \in E \\ \mathrm{incr}(e_2) \cup \mathrm{decr}(e_2) & \text{and } \bowtie \in \{<, \leq, \neq, =, >, \geq\} \\ \mathrm{vis}(\varphi_1) \cup \mathrm{vis}(\varphi_2) & \text{if } \varphi = \varphi_1 \mathcal{R} \varphi_2 \text{ where } \mathcal{R} \in \{\mathsf{U}, \wedge, \vee\} \\ \mathrm{vis}(\varphi') & \text{if } \varphi = \mathsf{Q}\varphi' \text{ where } \mathsf{Q} \in \{\mathsf{G}, \mathsf{F}, \mathsf{X}, \neg\} \\ \emptyset & \text{otherwise .} \end{cases}$$

The desired property of $\mathrm{vis}(\varphi)$ is that it is a visible set wrt. φ. By Corollary 3.19 and by construction $\mathrm{vis}(\varphi)$ is indeed a visible set since for any expression e mentioned in φ we have $\mathrm{incr}(e) \subseteq \mathrm{vis}(\varphi)$ and $\mathrm{decr}(e) \subseteq \mathrm{vis}(\varphi)$. Therefore, using Lemma 3.16 and Corollary 3.19 we can now implement the classic stubborn set method for Petri nets, as shown by the following theorem.

Theorem 3.21. *Let $N = (P, T, W, I)$ be a Petri net, $\mathcal{T} = (\mathcal{M}(N), T, \to, L, M_0)$ be the transition system of N, let $St : \mathcal{M}(N) \to 2^T$ be a reduction, and let φ be an LTL formula. Then St satisfies **COM**, **KEY**, and **VIS(φ)** if for all markings $M \in \mathcal{M}(N)$:*

1. *Either $St(M) = T$ or there exists a $t_{\mathrm{key}} \in St(M)$ such that $M \xrightarrow{t_{\mathrm{key}}}$,*
2. *$St(M)$ is a COM-saturated set in M, and*
3. *Either $(\mathrm{en}(s) \cap St(s)) \cap \mathrm{vis}(\varphi) = \emptyset$ or $St(s) = T$.*

Proof. By Lemma 3.16, Condition 2 ensures **COM**. Condition 1 ensures that, if M is not deadlocked, there is an enabled transition in $St(M)$, and since $St(M)$ satisfies **COM**, **KEY** is also satisfied. Finally, by Definition 3.20, we have $\mathrm{incr}(e_1) \cup \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2) \cup \mathrm{decr}(e_2) \subseteq St(M)$ for all atomic propositions $e_1 \bowtie e_2$ mentioned in φ. Thus as an immediate consequence of Corollary 3.19, $\mathrm{vis}(\varphi)$ is a visible set, so Condition 3 ensures **VIS(φ)**. □

As mentioned previously, the rule **IGN(φ)** is ensured during state space exploration by checking that if $M \xrightarrow{t} M'$ where M' is on the current search path then all successors $t \in \mathrm{en}(M)$ should be explored. Thus by Theorem 3.5 the present method for stubborn set generation for Petri nets is suitable for model checking $\mathrm{LTL}_{\mathsf{X}}$ formulae.

Reachability Stubborn Set Method To implement the reachability-based stubborn set method for Petri nets we need ways to ensure the axioms **R** and **SAFE**. To do this we now give syntax-directed definitions of the *interesting transitions* of a marking and a formula which ensures the axioms **R** and **SAFE**.

Definition 3.22 (Interesting transitions). [7] *Let $N = (P, T, W, I)$ be a Petri net and let $b \in \mathcal{B}(AP)$ be a proposition. For a marking $M \in \mathcal{M}(N)$ the set $A_M(b) \subseteq T$ of* interesting transitions of b *is defined inductively as $A_M(b) = \emptyset$ if*

$M \models b$, and otherwise as follows.

$$
\begin{aligned}
A_M(tt) = A_M(ff) &= \emptyset \\
A_M(e_1 < e_2) = A_M(e_1 \leq e_2) &= \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2) \\
A_M(e_1 > e_2) = A_M(e_1 \geq e_2) &= \mathrm{incr}(e_1) \cup \mathrm{decr}(e_2) \\
A_M(e_1 = e_2) &= \begin{cases} \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2) & \text{if } \mathrm{eval}_M(e_1) > \mathrm{eval}_M(e_2) \\ \mathrm{incr}(e_1) \cup \mathrm{decr}(e_2) & \text{if } \mathrm{eval}_M(e_1) < \mathrm{eval}_M(e_2) \end{cases} \\
A_M(e_1 \neq e_2) &= \mathrm{incr}(e_1) \cup \mathrm{decr}(e_2) \cup \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2) \\
A_M(b_1 \wedge b_2) &= A_M(b_i) \text{ for some } i \text{ where } M \not\models b_i \\
A_M(b_1 \vee b_2) &= A_M(b_1) \cup A_M(b_2)
\end{aligned}
$$

$$
\begin{aligned}
A_M(\neg e_1 < e_2) &= A_M(e_1 \geq e_2) & A_M(\neg e_1 \leq e_2) &= A_M(e_1 > e_2) \\
A_M(\neg e_1 > e_2) &= A_M(e_1 \leq e_2) & A_M(\neg e_1 \geq e_2) &= A_M(e_1 < e_2) \\
A_M(\neg e_1 = e_2) &= A_M(e_1 \neq e_2) & A_M(\neg e_1 \neq e_2) &= A_M(e_1 = e_2) \\
A_M(\neg(b_1 \wedge b_2)) &= A_M(\neg b_1 \vee \neg b_2) & A_M(\neg(b_1 \vee b_2)) &= A_M(\neg b_1 \wedge \neg b_2)
\end{aligned}
$$

The purpose of A_M is to be an efficiently computable syntactic over-approximation which ensures the rule **R**, i.e. to reach a state satisfying b some transition from $A_M(b)$ must be fired. These properties are formalised in the following two lemmas.

Lemma 3.23. [7] *Let $N = (P, T, W, I)$ be a Petri net, let $M \in \mathcal{M}(N)$ be a marking, and let $b \in \mathcal{B}(AP)$ be a proposition. If $M \not\models b$ and $M \xrightarrow{w} M'$ for some $w \in \overline{A_M(b)}^*$, then $M' \not\models b$.*

Lemma 3.23 reflects the rule **R**, hence by including $A_M(b)$ in a stubborn set $St(M, q)$ for all progressing propositions $b \in \mathrm{Prog}(q)$ we guarantee **R**. A_M also leads to a suitable definition of unsafe actions as follows.

Lemma 3.24. *Let $N = (P, T, W, I)$ be a Petri net and $b \in \mathcal{B}(AP)$ be a proposition. Then for any marking $M \in \mathcal{M}(N)$ where $M \not\models b$, the set $T \setminus A_M(b)$ is safe wrt. b, i.e. for $t \notin A_M(b)$ and $w \in \overline{A_M(b)}^*$, if $M \xrightarrow{w} M'$, $M \xrightarrow{tw} M''$, and $M' \not\models b$, then $M'' \not\models b$.*

Proof. We proceed by structural induction in the form of b. Let $t \notin A_M(b)$ and $w \in (T \setminus A_M(b))^*$, and assume $M \not\models b$, $M \xrightarrow{w} M'$, $M' \not\models b$, and $M \xrightarrow{tw} M''$. The inductive hypothesis is that for any subproposition b' of b, $M' \not\models b'$ implies $M'' \not\models b'$, i.e. t is safe wrt. any subproposition b' of b.

$b = e_1 < e_2$ Since $M \not\models b$ then $\mathrm{eval}_M(e_1) \geq \mathrm{eval}_M(e_2)$. By the definition of A_M we have $\mathrm{decr}(e_1) \cup \mathrm{incr}(e_2) \subseteq A_M(b)$, hence $t \notin \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2)$. Thus when $M \xrightarrow{t} M_t$, by Lemma 3.18 $t \notin \mathrm{decr}(e_1)$ implies $\mathrm{eval}_M(e_1) \leq \mathrm{eval}_{M_t}(e_1)$, and $t \notin \mathrm{incr}(e_2)$ implies $\mathrm{eval}_M(e_2) \geq \mathrm{eval}_{M_t}(e_2)$. Hence $\mathrm{eval}_{M'}(e_1) \leq \mathrm{eval}_{M''}(e_1)$ and $\mathrm{eval}_{M'}(e_2) \geq \mathrm{eval}_{M''}(e_2)$, so since $M' \not\models b$ we also have $M'' \not\models b$.

$b = e_1 \leq e_2$ Equivalent to $b = e_1 < e_2 + 1$
$b = e_1 > e_2$ Equivalent to $b = e_2 < e_1$.
$b = e_1 \geq e_2$ Equivalent to $b = e_2 < e_1 + 1$
$b = e_1 = e_2$ If $\text{eval}_M(e_1) > \text{eval}_M(e_2)$ then the argument proceeds as the case $e_1 < e_2$. If $\text{eval}_M(e_1) < \text{eval}_M(e_2)$ then the argument proceeds as the case $e_2 < e_1$.
$b = e_1 \neq e_2$ Since $\text{incr}(e_1) \cup \text{decr}(e_1) \cup \text{incr}(e_2) \cup \text{decr}(e_2) \subseteq A_M(b)$, if $M \xrightarrow{t} M_t$ then $\text{eval}_{M_t}(e_1) = \text{eval}_M(e_1)$ and $\text{eval}_{M_t}(e_2) = \text{eval}_M(e_2)$. Therefore since $M' \not\models b$ we also have $M'' \not\models b$.
$b = b_1 \wedge b_2$ Since $M \not\models b$, for some $i \in \{1, 2\}$ we must have $M \not\models b_i$ and $A_M(b_i) \subseteq A_M(b)$. By Lemma 3.23, since $t_i \notin A_M(b_i)$ we have $M' \not\models b_i$, so by the inductive hypothesis $M'' \not\models b$.
$b = b_1 \vee b_2$ Since $M' \not\models b$ then $M' \not\models b_1$ and $M' \not\models b_2$. By the inductive hypothesis we get $M'' \not\models b_1$ and $M'' \not\models b_2$ and therefore $M'' \not\models b$.

We have thus demonstrated that if $M \not\models b$, any transition $t \in T \setminus A_M(b)$ is safe wrt. b. □

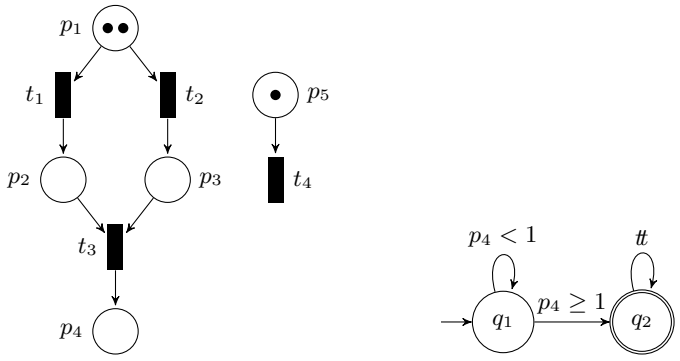
We now give a syntax-driven implementation of the reachability-based stubborn set method for LTL queries such that the rules **COM**, **R**, and **SAFE** are satisfied.

Theorem 3.25. *Let $N = (P, T, W, I)$ be a Petri net, $\mathcal{A} = (Q, \delta, Q_0, F)$ be an NBA, and $St : \mathcal{M}(N) \times Q \to 2^T$ be a product reduction such that, for all markings $M \in \mathcal{M}(N)$ and states $q \in Q$, St satisfies Condition 1 and 2, then St satisfies the rules **COM**, **R**, and **SAFE**.*

1. *Either $St(M, q) = T$, or $q \in \text{Reach}(\mathcal{A})$ and for all $b \in \text{Prog}(q)$ we have $\text{en}(M) \cap St(M, q) \subseteq T \setminus A_M(b)$.*
2. *Otherwise we have*
   (a) $\bigcup_{b \in \text{Prog}(q)} A_M(b) \subseteq St(M, q)$ and
   (b) $St(M, q)$ is a COM-saturated set in M.

Proof. By Lemma 3.24, Condition 1 ensures the rule **SAFE**. By Lemma 3.23, Condition 2a ensures **R**, and by Lemma 3.16 Condition 2b ensures **COM**. Hence by Theorem 3.12 a reduction satisfying the above conditions is an LTL-preserving reachability-based stubborn set. □

In summary, the stubborn set $St(M, q)$ is generated as follows. First, if $q \in F$ or some progressing proposition $b \in \text{Prog}(q)$ is satisfied by M, the set of all transitions is returned. Otherwise, the closure algorithm is run on the set of interesting transitions to obtain a stubborn set satisfying **COM** and **R**. The resulting stubborn set is checked for whether there is any overlap with enabled interesting transitions, in which case the set of all transitions is returned, otherwise the computed stubborn set is returned.



(a) Petri net (b) NBA

Fig. 5: Example of reachability stubborn set applied to Petri nets.

Example 3.26. We will now provide an example of using the reachability stubborn set method to compute a subset of successors of a Petri net marking. Consider the product system of the Petri net shown in Figure 5a and the NBA in Figure 5b. In the initial marking M_0 the enabled transitions are

$$\mathrm{en}(M_0) = \{t_1, t_2, t_4\} \ . \tag{1}$$

The goal of the stubborn set is then to determine a subset of these enabled transitions to explore which are sufficient to ensure the property being verified. We first need to determine if the method applies in q_1. Since q_1 is not an accepting state and $p_4 \geq 1 \vee p_4 < 1 = \#t$, it is a reachability state. The only progressing proposition $p_4 \geq 1$ is not satisfied by M_0 so we do not return all transitions immediately. We now determine the set of interesting transitions

$$A_{M_0}(p_4 \geq 1) = \mathrm{incr}(p_4) \cup \mathrm{decr}(1) = \{t_3\} \cup \emptyset = \{t_3\} \ .$$

Next we determine a set of transitions that contains t_3 and satisfies Lemma 3.16. Such a set is

$$St(M_0, q_1) = \{t_1, t_2, t_3\} \ . \tag{2}$$

We now ensure that none of the enabled transitions in this set are interesting. We have $\mathrm{en}(M_0) \cap St(M_0, q_1) = \{t_1, t_2\}$ which is disjoint from $A_{M_0}(p_4 \geq 1) = \{t_3\}$, so we have $\mathrm{en}(M_0) \cap St(M_0, q_1) \subseteq T \setminus A_{M_0}(p_4 \geq 1)$, and $St(M_0, q_1)$ is therefore a valid stubborn set. From Equation 1 we see that the transition t_4 is enabled in M_0, but it is not part of the stubborn set in Equation 2. As such the interleaving which fires t_4 first and then either t_1 or t_2 will not be explored by state space exploration, and the size of the state space is therefore reduced.

Mixed Stubborn Set Method The implementation of the mixed stubborn set method for Petri nets follows straightforwardly from the existing implementations of the classic stubborn set method and the reachability stubborn set

method and from Definition 3.13. In state $\langle M, q \rangle$, we check whether $q \in \text{Reach}(\mathcal{A})$ and whether no $b \in \text{Prog}(q)$ is satisfied by M. If this is the case, we apply the reachability stubborn set method, otherwise we fall back to the classic stubborn set method.

3.5 Evaluation

In this section we introduce the dataset and the setup used for the evaluation, and the results from the stubborn set methods[1]. For evaluating the proposed methods we use the dataset from the 2020 edition of the Model Checking Contest (MCC) [26]. The dataset consists of 1016 place/transition nets modelling academic and industrial use cases. In addition to these 1016 place/transition models, the MCC also features a set of coloured Petri nets, which we do not include in our evaluation. For each model 32 randomly generated LTL formulae are provided, split evenly between cardinality formulae and fireability formulae. This gives us a total of 32 512 model checking queries to evaluate our approaches against. For evaluating each configuration we run each query for 15 min with 16 GiB of memory on one core from an AMD Opteron 6376.

The techniques described in this paper are implemented as an extension to the LTL model checker presented in [42], which is a part of the open source verification engine `verifypn` [25] for the model checker TAPAAL [12]. The LTL model checker uses version 2.9.6 of the Spot library [13] for translating LTL formulae into NBAs, and a derivative of Tarjan's algorithm [20,38] for searching for accepting cycles. To speed up verification we also employ the query simplifications from [6] and most of the structural reductions from [7]. This configuration serves as the baseline against which we evaluate the techniques. The query simplifications have the potential to solve queries outright, before any state exploration is performed and thus before any of the techniques presented in this thesis have a chance to affect performance. To make the contributions of the present techniques as clear as possible we will omit all trivially solved queries from the dataset. These comprise two cases, namely those which are solved outright by query simplifications, and those which has no valid initial NBA state in the product state space. The resulting dataset after removing these queries contains 19 274 queries of which 7583 are cardinality queries and 11 691 are fireability queries.

For a visual comparison between different configurations we use cactus plots inspired by [8]. For each configuration, the time t_i taken to solve query q_i is measured, and the times t_i are sorted in increasing order. These sorted times are then plotted as $(1, t_1), (2, t_2) \ldots$, where $t_1 \leq t_2 \leq \cdots$ for each configuration using a logarithmic time axis. By doing this, different queries may have been used to evaluate different configurations, so direct comparisons on individual queries are not possible. However, these plots allow us to get a broad overview of how different configurations handle easier or harder queries. For example, one method might be really good at easy queries but scale poorly to harder

[1] Reproducibility package: `https://github.com/simwir/Aut-LTL-Thesis`

Table 1: Results from evaluation of the stubborn set methods by category. The suffix + denotes a positive answer, while − denotes a negative answer. C denotes cardinality queries and F denotes fireability queries.

(a) Number of answers by category.

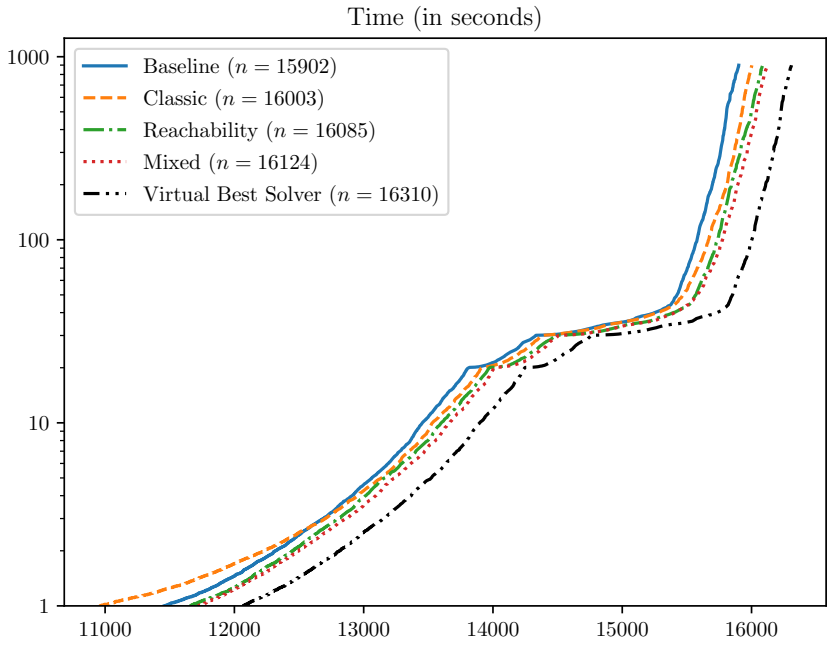
	LTLC+	LTLC−	LTLF+	LTLF−	Total	
Baseline	555	5708	703	8936	15 902	82.5 %
Classic	559	5735	714	8995	16 003	83.0 %
Reachability	565	5763	713	9044	16 085	83.5 %
Mixed	566	5775	725	9058	16 124	83.7 %

(b) Percentage gain in the number of answered queries compared to the baseline by category.

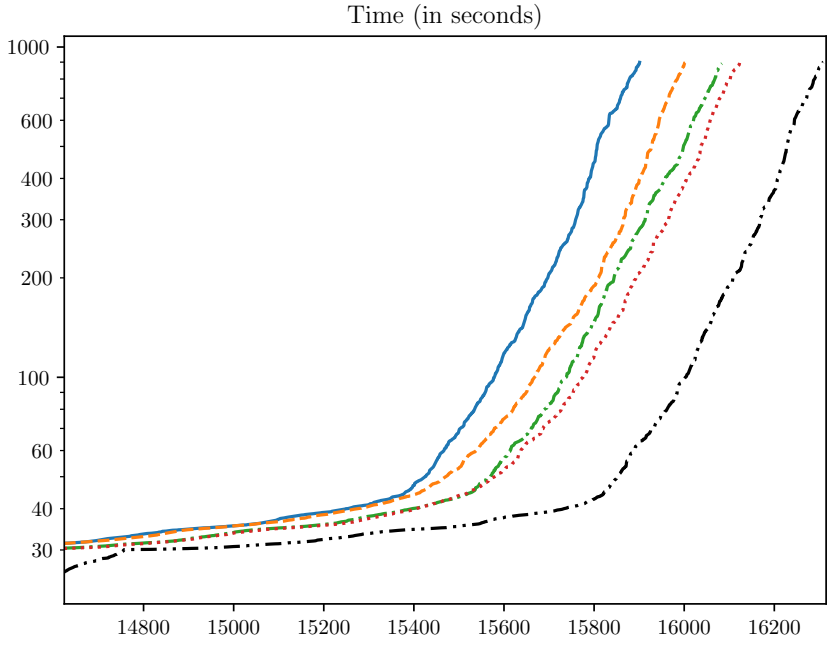
	LTLC+	LTLC−	LTLF+	LTLF−	Total
Classic	0.7 %	0.5 %	1.6 %	0.7 %	0.6 %
Reachability	1.8 %	1.0 %	1.4 %	1.2 %	1.2 %
Mixed	2.0 %	1.2 %	3.1 %	1.4 %	1.4 %

queries, whereas another may require a lot of overhead on easy queries but be capable of dealing with harder queries. We also include a Virtual Best Solver (VBS) [8] series, created by taking the minimum time to solve query q_i across all configurations. For example, if query q was solved in 1 s, 5 s, and 0.1 s by different configurations then the VBS time is 0.1 s. In addition to the cactus plots we include a breakdown of the number of positive and negative answers in each category. The categories are LTLC+, LTLC−, LTLF+, and LTLF−, where + refers to positive answers, i.e. no counterexample exists, − refers to negative answers, i.e. counterexample was found, and C and F refer to the cardinality and fireability categories.

Table 1a shows the number of answers obtained for each category using the different configurations, and Table 1b shows the percentage gains for each stubborn set method compared to the baseline. While in absolute numbers the additional answers are primarily due to negative answers, the percentage increase is largest for positive answers. In general, positive answers are expected to be harder to obtain than negative answers, as they require disproving the existence of any counterexample, which means the full state space has to be explored. While stubborn sets address the size of the state space, which does appear to help, in many cases the stubborn set method is not able to remove any states. For example, among the positive queries answered by both the baseline and the mixed stubborn set method, only 312 queries allowed for any reduction in the size of the state space. In the other 939 cases, the stubborn set method is pure



(a) All problems slower than 1 s.



(b) 1500 hardest problems, not counting VBS.

Fig. 6: Cactus plots over time taken by different stubborn set methods versus the baseline.

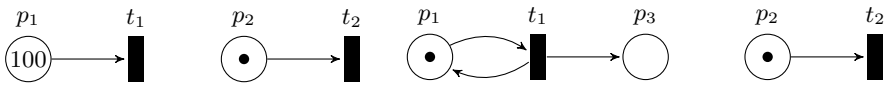
overhead, meaning that answers take longer to obtain. As for negative answers, reducing the size of the state space may also affect the search order, such that a path leading to a counterexample is more likely to be found. We also note that there are many more negative instances than positive instances in the dataset—of the 19 274 queries that are not trivially solved, we determine using the mixed method that 14 833 (76.96 %) are negative queries and only 1291 (6.70 %) are positive queries.

Figure 6 shows cactus plots from the experiments with stubborn sets. In Figure 6a all data points slower than 1 s are plotted, and in Figure 6b the 1500 hardest points for the best configuration (excluding VBS) are plotted. From Figure 6a we see that the classic method is slower on easier queries. A hypothesis is that the stubborn set extensions caused by $\mathbf{IGN}(\varphi)$ happen frequently in smaller models which results in a lot of overhead. The mixed method does not have this characteristic, perhaps because reachability states are common, and $\mathbf{IGN}(\varphi)$ does not apply in those. Otherwise, the main takeaway is that the configurations have very similar profiles, indicating minimal reason to not choose the best configuration. In Figure 6b we see that the different configurations behave very similarly until a point where queries start to become very difficult for the various configurations. Perhaps most interesting is that the reachability method and mixed method have near identical profiles until a slight shift around 15 500, indicating that this is where the mixed method pays off. The VBS has a similar performance characteristic to the actual methods but reaches further right than the other configurations, hinting that the methods are complementary.

Based on the evaluation we conclude that the mixed method has the best performance, although the reachability method on its own is also has a significant performance gain, both compared to not using any stubborn sets and to using the classical stubborn set method.

4 Guided search

When doing explicit state model checking using depth-first search algorithms, such as the on-the-fly variant of Tarjan's algorithm [38,20] which we use, how quickly a target state is found is highly dependent on the order in which successors to the current state are explored. By default, we use some arbitrary static ordering of the transitions in the Petri net to determine which transition to fire next. Worst case, the algorithm might spend a long time exploring an irrelevant area of the state space when a different choice of successor may lead to the answer faster. A way of alleviating this issue is by using heuristics to guide the search toward successors that are more likely to result in an answer. In the following we describe some heuristics suitable for LTL verification of Petri nets. In marking M, the heuristics assign a non-negative number to each M' where $M \to M'$ such that the markings with smaller numbers should be explored first. For heuristic function h, we accomplish this by sorting the set of successors of marking M, that is $\{M' \in \mathcal{M}(N) \mid M \to M'\}$, in ascending order based on the value of the heuristic $h(M')$.



(a) Example where heuristics can find a counter example faster.

(b) Example where heuristics can be applied to avoid the infinite loop.

Fig. 7: Example Petri nets where the search order matters heavily when verifying $\mathsf{G}p_2 \neq 0$.

Example 4.1. Consider the Petri net in Figure 7a, with the query $\mathsf{G}p_2 \neq 0$, that is p_2 is never 0. If the static ordering of the transitions puts t_1 before t_2 we end up firing t_1 100 times before t_2 is fired and the counterexample to $\mathsf{G}p_2 \neq 0$ is found. Firing t_2 before t_1 therefore saves 100 states, a significant speed-up. Using a heuristic to guide the search lets us accomplish this while not being dependent on being lucky with the arbitrary ordering of transitions. In the Petri net in Figure 7b, t_1 can be fired infinitely often, meaning that using the same transition ordering as before t_1 will be fired indefinitely, preventing us from finding the counterexample to $\mathsf{G}p_2 \neq 0$. In this case, a heuristic that at some point fires t_2 will result in us finding a counterexample.

This illustrates the two advantages we may gain from heuristics. Some problems which we are not able to answer without it suddenly becomes feasible and in other cases there may still be performance to be gained.

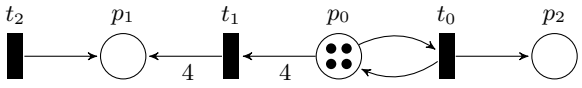
Distance-Based Heuristic The distance-based heuristic is roughly based on the difference in the number of tokens between a marking and the requirements described by the atomic propositions of the formula. This method is an adaptation of the heuristic search for reachability analysis of Petri nets presented in [25]. We view this difference as a *distance* between the current marking and a satisfying marking. The distance is calculated using the recursive formula dist defined in Figure 8. For a Petri net N, an LTL formula φ, and a marking $M \in \mathcal{M}(N)$ we calculate the heuristic as $\text{dist}(M, \varphi, \not\Vdash)$.

Example 4.2. Consider the Petri net N in Figure 9a and the LTL formula $\varphi = \neg\mathsf{F}(p_0 > 3 \wedge \mathsf{XF}p_1 > 3)$. We want to determine whether $N \models \varphi$. Assume that the static transition ordering is $t_0 < t_1 < t_2$. With this static ordering we see that p_2 is going to be ever increasing and we will never find a counterexample. In order to use the heuristics we first calculate the distance function for each successor. We let M_i denote the marking after firing t_i once. We then get $\text{dist}(M_0, \varphi, \not\Vdash) = 4$, $\text{dist}(M_1, \varphi, \not\Vdash) = 4$, and $\text{dist}(M_2, \varphi, \not\Vdash) = 3$. The heuristic prioritises the transition t_2, leading us one step closer to violating $\mathsf{F}p_1 > 3$. Repeating the procedure, after 4 firings of t_2, we end up in a marking where $M(p_1) = 4$, and considering the NBA in Figure 9b we end up in q_2, which is accepting and has a tautology as the retarding proposition. As such it is clear to see that any continuation of this run will visit the accepting NBA state infinitely often, resulting in a counterexample to the formula.

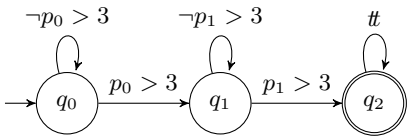
$$
\begin{aligned}
\mathrm{dist}(M, \mathsf{Q}\varphi, \mathit{negated}) &= \mathrm{dist}(M, \varphi, \mathit{negated}), \text{ if } \mathsf{Q} \in \{\mathsf{A}, \mathsf{F}, \mathsf{X}\} \\
\mathrm{dist}(M, \mathsf{G}\varphi, \mathit{negated}) &= \mathrm{dist}(M, \varphi, \neg\mathit{negated}) \\
\mathrm{dist}(M, \varphi_1 \,\mathsf{U}\, \varphi_2, \mathit{negated}) &= \mathrm{dist}(M, \varphi_2, \mathit{negated}) \\
\mathrm{dist}(M, \neg\varphi, \mathit{negated}) &= \mathrm{dist}(M, \varphi, \neg\mathit{negated}) \\
\mathrm{dist}(M, \varphi_1 \wedge \varphi_2, \mathit{ff}) &= \mathrm{dist}(M, \varphi_1, \mathit{ff}) + \mathrm{dist}(M, \varphi_2, \mathit{ff}) \\
\mathrm{dist}(M, \varphi_1 \vee \varphi_2, \mathit{ff}) &= \min(\mathrm{dist}(M, \varphi_1, \mathit{ff}), \mathrm{dist}(M, \varphi_2, \mathit{ff})) \\
\mathrm{dist}(M, \varphi_1 \wedge \varphi_2, \mathit{tt}) &= \min(\mathrm{dist}(M, \varphi_1, \mathit{tt}), \mathrm{dist}(M, \varphi_2, \mathit{tt})) \\
\mathrm{dist}(M, \varphi_1 \vee \varphi_2, \mathit{tt}) &= \mathrm{dist}(M, \varphi_1, \mathit{tt}) + \mathrm{dist}(M, \varphi_2, \mathit{tt}) \\
\mathrm{dist}(M, e_1 \bowtie e_2, \mathit{negated}) &= \Delta(\bowtie, \mathrm{eval}_M(e_1), \mathrm{eval}_M(e_2), \mathit{negated}) \\
&\qquad \text{for } \bowtie \in \{<, \leq, \neq, =, >, \geq\}
\end{aligned}
$$

$$
\begin{aligned}
\Delta(=, v_1, v_2, \mathit{ff}) &= |v_1 - v_2| \\
\Delta(\neq, v_1, v_2, \mathit{ff}) &= \begin{cases} 1, & \text{if } v_1 = v_2 \\ 0, & \text{otherwise} \end{cases} \\
\Delta(<, v_1, v_2, \mathit{ff}) &= \max(v_1 - v_2 + 1, 0) \\
\Delta(\leq, v_1, v_2, \mathit{ff}) &= \max(v_1 - v_2, 0) \\
\Delta(>, v_1, v_2, \mathit{ff}) &= \Delta(<, v_2, v_1, \mathit{ff}) \\
\Delta(\geq, v_1, v_2, \mathit{ff}) &= \Delta(\leq, v_2, v_1, \mathit{ff})
\end{aligned}
\qquad
\begin{aligned}
\Delta(=, v_1, v_2, \mathit{tt}) &= \Delta(\neq, v_1, v_2, \mathit{ff}) \\
\Delta(\neq, v_1, v_2, \mathit{tt}) &= \Delta(=, v_1, v_2, \mathit{ff}) \\
\Delta(<, v_1, v_2, \mathit{tt}) &= \Delta(\geq, v_1, v_2, \mathit{ff}) \\
\Delta(>, v_1, v_2, \mathit{tt}) &= \Delta(\leq, v_1, v_2, \mathit{ff}) \\
\Delta(\leq, v_1, v_2, \mathit{tt}) &= \Delta(>, v_1, v_2, \mathit{ff}) \\
\Delta(\geq, v_1, v_2, \mathit{tt}) &= \Delta(<, v_1, v_2, \mathit{ff})
\end{aligned}
$$

Fig. 8: Definition of the distance between a marking and a LTL formula. Based on [25] and extended to LTL.



(a) Petri net.



(b) NBA corresponding to the LTL formula $\mathsf{F}(p_0 > 3 \wedge \mathsf{XF}p_1 > 3)$.

Fig. 9: Example system where heuristics are advantageous when considering the LTL formula $\varphi = \neg\mathsf{F}(p_0 > 3 \wedge \mathsf{XF}p_1 > 3)$.

Automaton-Based Heuristic We now present a refinement of the previous method where instead of looking at the entire LTL formula, we look at the progressing formulae of the current state in the NBA. The main idea of this approach is that if we are not in an accepting state then we would like to leave the current state as quickly as possible and move closer to an accepting state. As such we prioritise transitions based on how quickly they can enable a progressing formula and how far the resulting NBA state is from some accepting state. Let N be a Petri net, $\mathcal{T} = (\mathcal{M}(N), T, \to, L, M_0)$ be an LTS, $\mathcal{A} = (Q, \delta, Q_0, F)$ be an NBA, and for $q \in Q$ let $\mathrm{BFS}(q)$ be the shortest path distance from q to any $q' \in F$ (if $q \in F$ then $\mathrm{BFS}(q) = 0$). Then given a state $\langle M, q \rangle$ in $\mathcal{T} \otimes \mathcal{A}$ where $q \notin F$, we calculate the heuristic for each successor $M' \in \mathrm{suc}(M)$ as the minimum of $(1 + \mathrm{BFS}(q')) \cdot \mathrm{dist}(M', b, \mathit{ff})$ over all $q' \in Q$ where $q \xrightarrow{b} q'$. For improved performance we precompute $\mathrm{BFS}(q)$ for all $q \in Q$.

Example 4.3. Let us again consider the Petri net in Figure 9a, and the NBA corresponding to $\neg\varphi$, presented in Figure 9b. First we notice in Definition 2.6 that when creating the initial states Q'_0 of the product state space we evaluate the outgoing formulae of the initial NBA states once. In this case the result is that we have to evaluate the outgoing propositions of q_0 to determine the initial product states. Here we see that the initial marking satisfies the progressing proposition $p_0 > 3$ but not the retarding proposition $\neg p_0 > 3$. As such the only inital NBA state in the product state space is q_1. Next we calculate the heuristic where, as before, M_i is the marking resulting from firing t_i. As there is only one progressing proposition we can simplify the heuristic calculation to $(1+\mathrm{BFS}(q_1))+\mathrm{dist}(M_i, p_1 > 3, \mathit{ff})$, yielding the values $1 \cdot \mathrm{dist}(M_0, p_1 > 3, \mathit{ff}) = 4$, $1 \cdot \mathrm{dist}(M_1, p_1 > 3, \mathit{ff}) = 0$, and $1 \cdot \mathrm{dist}(M_2, p_1 > 3, \mathit{ff}) = 3$. The transition with the highest priority is t_1 which immediately leads to a marking where $p_1 > 3$, so we can move to the accepting state. This illustrates a key advantage that the NBA based heuristic has over using the distance-based heuristic on the entire LTL formula, namely that is can disregard parts of the formula that are not relevant at the moment.

Fire-Count Heuristic Sometimes when exploring the state space the same transitions are fired many times compared to the other transitions. This may be a problem since only firing the same transitions may prevent different behaviours of the net from being explored. We therefore propose a heuristic that tries to balance the number of times each transition is fired by penalising transitions that are fired many times. Generally firing a transition a few more times than some other transition poses no problem, so we introduce a threshold such that each firing above this threshold incurs a penalty that is logarithmic in the number of firings over the threshold. Let $t \in T$ be a transition from a Petri net, let f_t denote the number of firings of t in the current search path, and let ℓ be the penalty threshold. Then the heuristic is calculated as

$$\mathrm{FC}(t) = \begin{cases} 0 & \text{if } f_t < \ell \\ \lfloor \log_2(f_t - \ell) \rfloor & \text{otherwise} \end{cases} .$$

Table 2: Results from the evaluation of heuristics.

(a) Number of answers by category.

	LTLC+	LTLC−	LTLF+	LTLF−	Total	
Baseline	555	5708	703	8936	15 902	82.5 %
Fire-Count	555	5878	699	9165	16 297	84.6 %
Distance	553	5907	693	9187	16 340	84.8 %
Automaton	555	5967	693	9236	16 451	85.4 %
Automaton + Fire-Count	555	5973	690	9244	16 462	85.4 %

(b) Percentage gain in the number of answered queries compared to the baseline by category.

	LTLC+	LTLC−	LTLF+	LTLF−	Total
Fire-Count	0.0 %	3.0 %	−0.6 %	2.6 %	2.5 %
Distance	−0.4 %	3.5 %	−1.4 %	2.8 %	2.8 %
Automaton	0.0 %	4.5 %	−1.4 %	3.4 %	3.5 %
Automaton + Fire-Count	0.0 %	4.6 %	−1.8 %	3.4 %	3.5 %

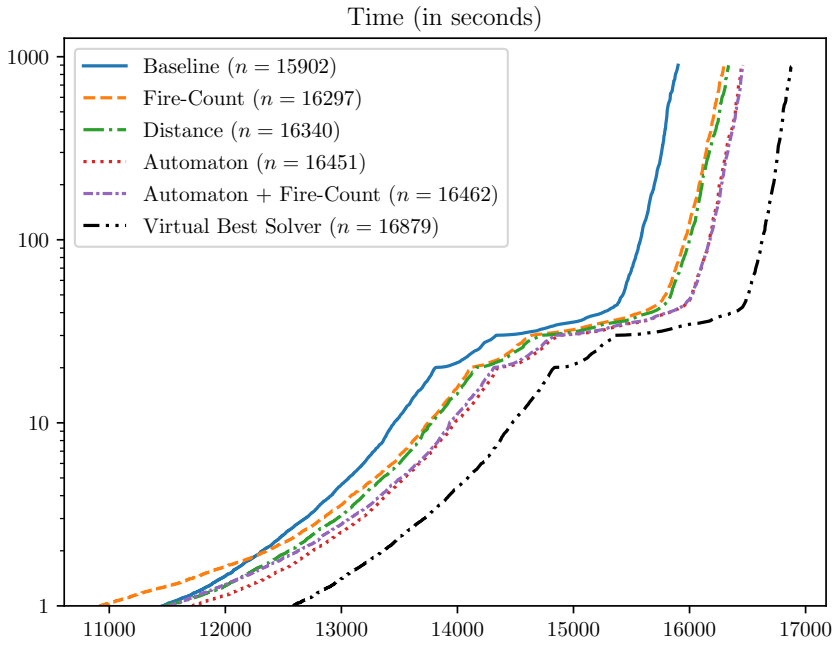
An alternative approach to this issue is presented in [29]. They explore an under-approximation technique where the token counts in the initial marking are reduced if individual places contain many tokens.

Combining Heuristics To determine whether different heuristics are complementary we also consider combining heuristics together by taking the sum of two individual heuristics. We see the automaton-based heuristic as an extension of the distance-based heuristic, so we only consider the combination of the automaton heuristic and the fire-count heuristic.

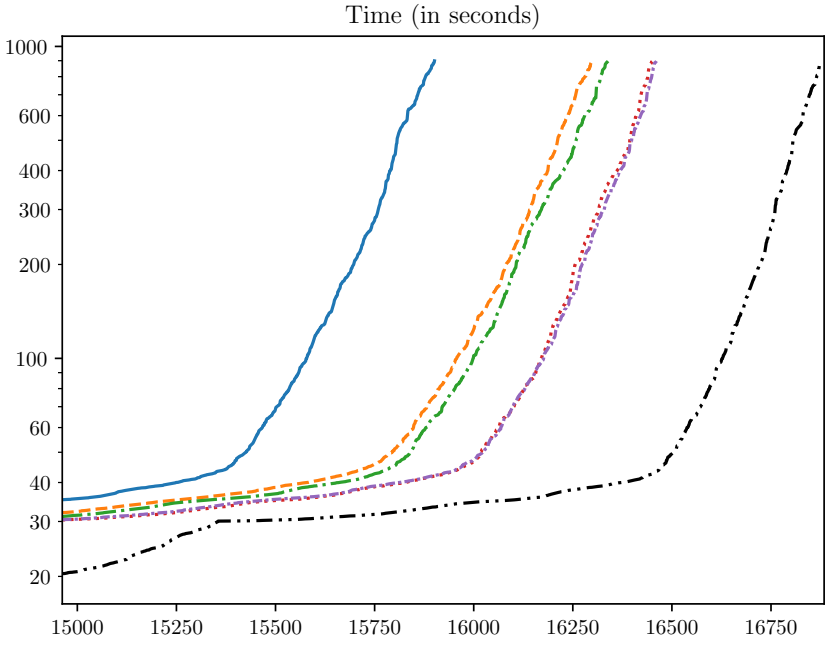
4.1 Evaluation

In this section we present experimental evaluation of the heuristics. The experimental setup is as before: 15 min and 16 GiB of RAM per query. By parameter search we determined that for the fire-count heuristic alone a threshold of $\ell = 500$ worked best, and for combining the fire-count heuristic with the automaton heuristic a threshold of $\ell = 5000$ worked best. These values are used for the configurations involving the fire-count heuristic.

Figure 10 shows cactus plots for the different configurations of heuristics. In Figure 10a we note that the fire-count heuristic takes a comparatively long time to solve simpler queries. This may be because the heuristic starts by zero-initialising an array of size $|T|$, the number of transitions in the net, and because the heuristic has no effect before the threshold is met. For harder queries all the



(a) All answers slower than 1 s.



(b) Top 1500 answers.

Fig. 10: Cactus plots for heuristics.

heuristics achieve gains, as can be seen clearly in Figure 10b. At the hardest 1500 queries, we see a similar trend as with the stubborn set methods where the improvements mainly enable solving more problems before they become too difficult. The distance between the VBS and the actual techniques strongly suggest that running different processes in parallel has great potential.

Table 2a shows the number of answers by category. There are no gains in the positive cases which is expected since the heuristics only affect the search order and in positive cases we have to explore the entire state space. Interestingly, in the LTLC+ category nearly no answers are lost while in LTLF+ up to 13 answers are lost in the worst case. One explanation of this is the explosion in the number of atomic propositions when expanding fireability atomic propositions into cardinality atomic propositions. In the worst case each fireability atomic proposition t could be unfolded into a disjunction of size $2 \cdot |P|$ if for all places $p \in P$ we have $W(p,t) > 0$ and $I(p,t) < \infty$. In that case computing the distance $\text{dist}(M, \varphi)$ may then become a significant task. The gains are similar between LTLC− and LTLF−.

The single best configuration based on these experiments is the automaton heuristic composed with the fire-count heuristic.

5 Performance Evaluation

In the following, we present experimental performance evaluation of the full model checker, including stubborn sets and heuristics, as well as a comparison against the state of the art LTL model checker ITS-LoLA, under the MCC'20 setup.

5.1 Combined Stubborn Sets and Heuristics

In this section we present experimental evaluation of combining the heuristics and the stubborn sets. As previously, the experimental setup is 15 min and 16 GiB of RAM per query. Configurations are denoted using "[heuristic] + [stubborn set]". In these configurations, the state space is reduced using stubborn sets, and the resulting set of transitions is ordered based on the heuristic. For example, the configuration Automaton + Reachability describes a search guided by the automaton heuristic in a state space reduced by the reachability stubborn set method.

Table 3 lists results from evaluating combinations of the automaton heuristic with the different stubborn set methods. The combination of the automaton heuristic and the fire-count heuristic, which was previously determined to be the single best heuristic, combined with mixed stubborn sets achieved a total of 16 635 total answers, less than the automaton heuristic combined with mixed stubborn sets which achieved 16 658 answers. We therefore omit this configuration from further analysis. As expected based on the evaluation of stubborn sets, the mixed stubborn set method answers the most queries, followed by the reachability stubborn set method and the classic stubborn set method. Figure 11b

Table 3: Comparison of the performance gains from combining the automaton heuristic with each of the stubborn set methods.

(a) Number of answers by category for combinations of automaton heuristic and its variants with reachability and mixed stubborn set methods.

	LTLC+	LTLC−	LTLF+	LTLF−	Total	
Baseline	555	5708	703	8936	15 902	82.5 %
Automaton + Classic	559	5994	705	9260	16 518	85.7 %
Automaton + Reach	565	6022	705	9315	16 607	86.2 %
Automaton + Mixed	566	6039	716	9337	16 658	86.4 %

(b) Percentage gain in the number of answered queries compared to the baseline by category.

	LTLC+	LTLC−	LTLF+	LTLF−	Total
Automaton + Classic	0.7 %	5.0 %	0.3 %	3.6 %	3.9 %
Automaton + Reach	1.8 %	5.5 %	0.3 %	4.2 %	4.4 %
Automaton + Mixed	2.0 %	5.8 %	1.8 %	4.5 %	4.8 %

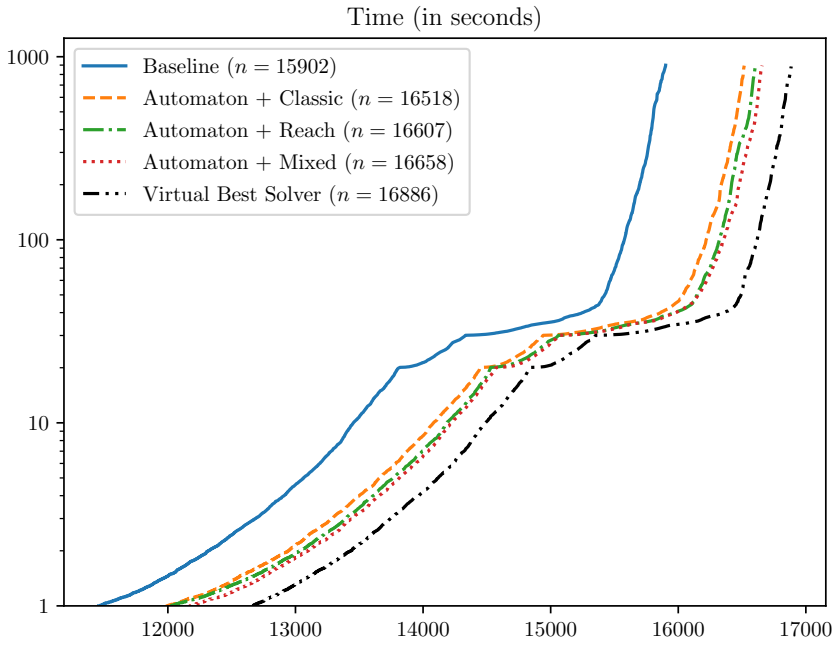
Table 4: Comparison of the individual performance gains from the mixed stubborn set method and automaton heuristic with the performance gains from combining the techniques.

(a) Number of answers by category.

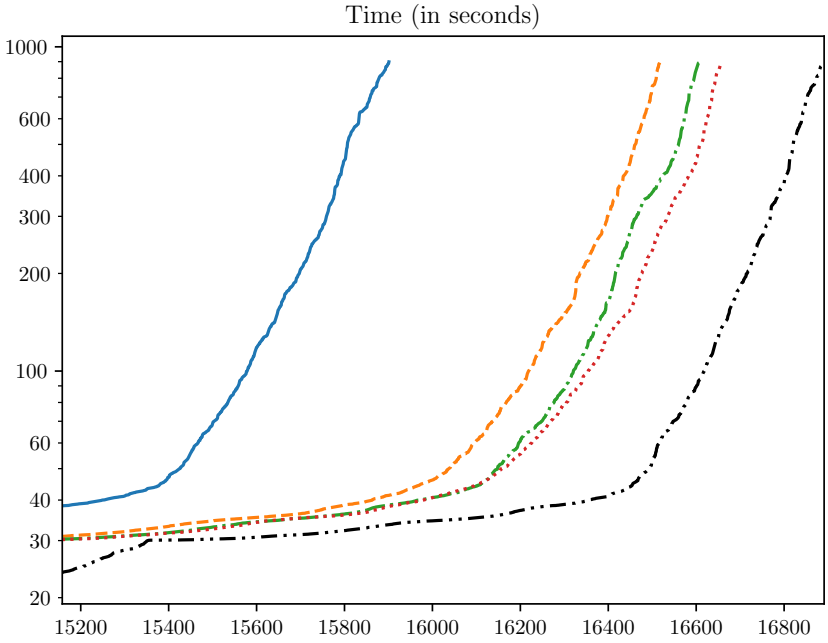
	LTLC+	LTLC−	LTLF+	LTLF−	Total	
Baseline	555	5708	703	8936	15 902	82.5 %
Mixed Stubborn	566	5775	725	9058	16 124	83.7 %
Automaton Heuristic	555	5967	693	9236	16 451	85.4 %
Automaton + Mixed	566	6039	716	9337	16 658	86.4 %

(b) Percentage gain in the number of answered queries compared to the baseline by category.

	LTLC+	LTLC−	LTLF+	LTLF−	Total
Mixed Stubborn	2.0 %	1.2 %	3.1 %	1.4 %	1.4 %
Automaton Heuristic	0.0 %	4.5 %	−1.4 %	3.4 %	3.5 %
Automaton + Mixed	2.0 %	5.8 %	1.8 %	4.5 %	4.8 %



(a) All answers slower than 1 s.



(b) Top 1500 answers.

Fig. 11: Cactus plots for combined heuristics and stubborn sets.

shows a similar trend as Figure 6 where the reachability method and mixed method have similar profiles up to a point where they diverge at around 16 100.

Table 4 recounts the results from the mixed stubborn set method, the automaton heuristic and the combination Automaton + Mixed. The number of answers gained by the combined technique over the baseline is approximately equal to the sum of answers gained by the individual techniques—the mixed stubborn set method gains 222 answers over the baseline, and the automaton heuristic gains 549 answers over the baseline, while the combined method gains 756 answers, slightly less than $222 + 549 = 771$.

In conclusion, the single best configuration is the combination of the mixed stubborn set method with the automaton heuristic.

5.2 Evaluation Against State of the Art

To evaluate the performance of our LTL model checker, we compare our implementation against ITS-LoLA [39,48], the winner of the LTL category of the 2020 edition of the Model Checking Contest (MCC) [26]. The evaluation is performed following the setup of the MCC, and against the version of ITS-LoLA submitted for the competition.[2] Unlike in the previous evaluations we do not remove trivially solved queries, so for each of the 1016 model instances there are 16 cardinality formulae and 16 fireability formulae. For each evaluation the tool is allotted 16 GiB of memory and 4 CPU cores. The tool must then answer as many of the 16 queries as possible within 1 h. We evaluate each tool on the total number of answers.

We now describe in detail how we use the available resources. We perform the verification of the 16 queries in four phases.

Phase 1: Parallel Simplification In the first phase we perform parallel simplification. Here we run the query simplification as described in [6,42] on the 16 queries in parallel, after which we perform structural reductions based on the methods described in [42,7]. We run the parallel simplification for at most 17 min. The query simplification may solve some queries outright, and if so we remove them from the to-do list for the remaining phases.

Phase 2: Parallel Verification The second phase is the parallel verification phase. Here we run through each query in the to-do list and try four different strategies in parallel. We use this phase to try different heuristics in the hope that one of them quickly finds a counterexample. As we saw in Figure 10 the VBS is significantly better than the individual heuristics. In this phase we hope to win some of that performance by running them in parallel. This phase is run for up to 5 min for each of the remaining queries. The four strategies run in parallel are:

– Distance heuristic,

[2] Available at: `https://mcc.lip6.fr/2020/results.php`

Table 5: Number of answers in the MCC setup.

	LTLC+	LTLC−	LTLF+	LTLF−	Total	
Baseline	5276	10 039	2855	11 818	29 988	92.3 %
Tapaal	5292	10 124	2878	11 931	30 225	93.0 %
ITS-LoLA	5204	9278	2664	10 133	27 277	84.0 %

– Automaton heuristic,
– Composed heuristic of automaton heuristic and log fire-count with a threshold of 5000, and
– No heuristic but nested depth first search instead of Tarjan's algorithm.

For the three strategies that use Tarjan's algorithm we use the mixed stubborn set method.

Phase 3: Sequential Verification After the parallel verification phase, we enter the sequential verification phase, where a single verification job is run at a time, but for a longer time. This is due to the 16 GiB memory restrictions which we might exceed if we run multiple processes in parallel for a long time. This verification job is run using our overall best configuration, namely the mixed stubborn set method using the automaton based heuristic. The idea in this phase is that in order to determine that there is no counterexample we have to explore the entire state space. This likely requires a large amount of time and memory, especially since these are formulae that were not solved in previous, shorter phases. In this phase we use a dynamic timeout based on the remaining time available and queries still to be solved. We run each query for $\max\left(8\,\text{min}, \frac{\text{Remaining time}}{\text{Remaining queries}}\right)$, evaluated at the start of each query. As such if some queries are solved quickly, the remaining are allotted a bit more time.

Phase 4: Random Verification If there happens to be any more time left after the sequential verification phase, and there still are some queries which have not been answered the final phase is the random verification phase. In this phase we run 4 random searches in parallel on each of the remaining queries in turn for $\frac{\text{Remaining time}}{\text{Remaining queries}}$ each. We repeat this until either all queries have been answered or we run out of time.

The baseline configuration runs the same scheduling algorithm but using different configurations. The sequential step is done without any stubborn set reductions or heuristics, and instead of various heuristics the parallel verification phase is done using a mix of random and non-random Tarjan's algorithm [20] and NDFS [11].

Results Table 5 shows the number of answers in the MCC setup. Tapaal and ITS-LoLA disagree on 25 queries. There is no official ground truth for the MCC dataset, but Thierry-Mieg maintains a repository of trusted answers from the

competition, which is updated in case an answer is found to have been wrong or uncertain [41]. This repository only contains answers for queries for which there were determined a trusted answer [26], so not all queries have answers. Out of the 25 disagreements the trusted answers assigns 4 errors to us in the fireability category and assigns to ITS-LoLA 8 errors in the fireability category and 5 errors in the cardinality category. Since both tools disagree with the trusted answers neither are infallible so we remove these 25 queries from the dataset. There is a disagreement between the baseline and ITS-LoLA on 1 cardinality query and 1 fireability query. These have also been removed from the dataset. There are no disagreements between baseline and Tapaal. This leaves us with a dataset of 32 485 queries.

Tapaal outperforms ITS-LoLA with a margin of around 3000 queries, a percentage increase of 10.8 %, a significant improvement compared to state of the art. This margin corresponds to solving 56.8 % of queries not solved by ITS-LoLA. Secondly we see that while Tapaal performs better in all categories, most of our performance improvement stems from queries containing a counterexample. This is in line with the results we have seen above where the techniques introduced in this thesis mainly affected the number of counterexamples found, and only to a lesser degree the number of positive answers. Considering that heuristics only affect negative queries this is expected. Furthermore, ITS-LoLA employs multiple techniques that assist in the positive cases [48,40]. They use the same classic stubborn set method as we do [45] and they also use structural reductions [40], but unlike us they use symmetry reductions [35,36], allowing for further state space reductions.

Looking at baseline versus Tapaal we note that the improvement of Tapaal is smaller than noted in previous, single-query experiments. One hypothesis is that the random searches and use of NDFS in the parallel step of the baseline help make up much of the difference otherwise gained via heuristics. This would be consistent with smaller gains in negative answers than previously and similar gains in positive answers, which are likely due to the partial order reductions. Still, for practical applications the performance of a single setup is important, and we are therefore of the opinion that the presented techniques display a significant improvement.

6 Conclusion

We presented a novel automaton-based stubborn set technique and automaton-based heuristics for guided search that are suitable for Linear Temporal Logic (LTL) model checking of Petri nets. The stubborn set method is a variant of reachability-preserving stubborn set methods where we consider outgoing edges in the Nondeterministic Büchi Automaton (NBA) as a reachability problem under certain conditions. It improves on the classic stubborn set method [44] by allowing arbitrary LTL formulae rather than only next-free formulae. We also presented several heuristics for guided search of model checking LTL on Petri nets. The automaton-based heuristic is based on an adaptation of the heuristics

for reachability analysis in [25] to LTL, with the modification that local information in the NBA is used. We also considered the adapted heuristic without using local information. Lastly we presented a heuristic based on punishing firing the same transition too many times to encourage exploring different aspects of the model.

We implemented these heuristics and the stubborn set method along with an implementation of the classic stubborn set method [44] as an extension to the open source model checking backend `verifypn` [25] for TAPAAL [12]. Based on this implementation we experimentally evaluated the techniques using the dataset from the 2020 edition of the Model Checking Contest (MCC) [26], which after removing trivially solved queries contains 19 274 queries. We found that the reachability stubborn set method solved 16 085 (83.5 %) queries while the classic stubborn set method solved 16 003 (83.0 %). The baseline solved 15 902 (82.5 %) queries, meaning that the reachability stubborn set method gains nearly twice as many answers compared to the baseline as the classic stubborn set method. The adapted heuristic without local information solved 16 340 (84.8 %) queries while the automaton-based heuristic solved 16 451 (85.4 %), demonstrating an advantage to using local information. The heuristic punishing firing the same transition too many times answered 16 297 (84.6 %) queries, less than the distance-based heuristic.

To find a single best configuration using these techniques, we evaluated combinations of the stubborn set methods and the NBA based heuristic. When combined with the automaton heuristic, the classic stubborn set method answered 16 518 (85.7 %) of queries, the reachability stubborn set method answered 16 607 (86.2 %) of queries, and the mixed stubborn set method answered 16 658 (86.4 %) of queries. We found that the combined automaton and fire-count heuristic with the mixed stubborn set answered 16 635 (86.3 %) queries, less than the same configuration without the fire-count heuristic. We also found that the individual gains from stubborn sets and heuristics are approximately additive. The mixed stubborn set method gains 222 answers over the baseline, and the automaton heuristic gains 549 answers over the baseline, while the combination gains 756 answers, slightly less than $222 + 549 = 771$. This is a percentage increase of 4.8 % over the baseline in number of answers, illustrating that the novel techniques we have introduced improves the performance of the LTL model checker in `verifypn`.

To evaluate the real world performance of the techniques presented in this thesis we compare against ITS-LoLA [39,48], the winner of the 2020 edition of the MCC [26]. As dataset we used the full MCC'20 dataset but discarded 27 queries where TAPAAL and ITS-LoLA disagreed on the answer. On the resulting dataset of 32 485 queries, the new model checker solved 30 225 (93.0 %) queries while ITS-LoLA solved 27 277 (84.0 %) queries, a improvement of 10.8 % by TAPAAL, and a reduction of unsolved queries by 56.8 % compared to ITS-LoLA.

We thus conclude that the techniques presented in this thesis shows significant performance gain, and with these improvements the LTL-engine of TAPAAL is among state of the art.

Future Work A line of inquiry into the reachability stubborn set method is to expand the set Reach($\mathcal{A}$) of reachability states beyond those whose edges form a tautology or perhaps even into accepting states, which would make the method applicable in a wider range of typical LTL properties. There may also be better ideas for how to handle unsafe actions, either generating them in a different way or taking less extreme recovery actions than setting $St(s, q)$ to the set of all actions. Stubborn set methods that target other structural properties of NBAs, such as weak NBAs or other properties related to connected components, are prospective directions for more specialised automaton-based methods. For heuristics, interesting questions include investigating other fire-count penalty functions than the logarithm, and heuristics which also apply in the accepting states of the NBA. Other ways of prioritising the NBA than BFS may also be of interest. Lastly, the algorithms used for generating NBAs may exhibit properties or patterns that are exploitable for smarter techniques.

References

1. Babiak, T., Křetínskỳ, M., Řehák, V., Strejček, J.: LTL to büchi automata translation: Fast and more deterministic. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 95–109. Springer (2012)
2. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT press (2008)
3. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: International conference on tools and algorithms for the construction and analysis of systems. pp. 193–207. Springer (1999)
4. Bønneland, F., Jensen, P., Larsen, K., Muniz, M., Srba, J.: Start pruning when time gets urgent: Partial order reduction for timed systems. In: Proceedings of the 30th International Conference on Computer Aided Verification (CAV'18). LNCS, vol. 10981, pp. 527–546. Springer-Verlag (2018). https://doi.org/10.1007/978-3-319-96145-3_28
5. Bønneland, F., Jensen, P., Larsen, K., Muniz, M., Srba, J.: Stubborn set reduction for two-player reachability games. Logical Methods in Computer Science **17**(1), 1–26 (2021)
6. Bønneland, F., Dyhr, J., Jensen, P.G., Johannsen, M., Srba, J.: Simplification of CTL formulae for efficient model checking of Petri nets. In: International Conference on Applications and Theory of Petri Nets and Concurrency. pp. 143–163. Springer (2018)
7. Bønneland, F.M., Dyhr, J., Jensen, P.G., Johannsen, M., Srba, J.: Stubborn versus structural reductions for petri nets. Journal of Logical and Algebraic Methods in Programming **102**, 46–63 (Jan 2019). https://doi.org/10.1016/j.jlamp.2018.09.002, `https://doi.org/10.1016/j.jlamp.2018.09.002`
8. Brain, M., Davenport, J.H., Griggio, A.: Benchmarking solvers, SAT-style. In: SC^2@ ISSAC (2017)
9. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Information and computation **98**(2), 142–170 (1992)

10. Clarke, E.M., Emerson, E.A., Jha, S., Sistla, A.P.: Symmetry reductions in model checking. In: Computer Aided Verification, pp. 147–158. Springer Berlin Heidelberg (1998). https://doi.org/10.1007/bfb0028741, https://doi.org/10.1007/bfb0028741
11. Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M.: Memory-efficient algorithms for the verification of temporal properties. Formal methods in system design **1**(2-3), 275–288 (1992)
12. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K., Møller, M., Srba, J.: TAPAAL 2.0: integrated development environment for timed-arc Petri nets. In: Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12). LNCS, vol. 7214, p. 492–497. Springer-Verlag (2012)
13. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 — a framework for LTL and ω-automata manipulation. In: Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16). Lecture Notes in Computer Science, vol. 9938, pp. 122–129. Springer (oct 2016). https://doi.org/10.1007/978-3-319-46520-3_8
14. Edelkamp, S., Jabbar, S.: Large-scale directed model checking LTL. In: International SPIN Workshop on Model Checking of Software. pp. 1–18. Springer (2006)
15. Edelkamp, S., Lafuente, A.L., Leue, S.: Directed explicit model checking with HSF-SPIN. In: International SPIN Workshop on Model Checking of Software. pp. 57–79. Springer (2001)
16. Edelkamp, S., Schuppan, V., Bošnački, D., Wijs, A., Fehnker, A., Aljazzar, H.: Survey on directed model checking. In: International Workshop on Model Checking and Artificial Intelligence. pp. 65–89. Springer (2008)
17. Esparza, J., Heljanko, K.: A new unfolding approach to LTL model checking. In: International Colloquium on Automata, Languages, and Programming. pp. 475–486. Springer (2000)
18. Esparza, J., Křetínskỳ, J., Sickert, S.: One theorem to rule them all: A unified translation of LTL into ω-automata. In: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science. pp. 384–393 (2018)
19. Esparza, J., Schröter, C.: Net reductions for LTL model-checking. In: Lecture Notes in Computer Science, pp. 310–324. Springer Berlin Heidelberg (2001). https://doi.org/10.1007/3-540-44798-9_25, https://doi.org/10.1007/3-540-44798-9_25
20. Geldenhuys, J., Valmari, A.: More efficient on-the-fly LTL verification with Tarjan's algorithm. Theoretical Computer Science **345**(1), 60–82 (2005)
21. Godefroid, P.: Using partial orders to improve automatic verification method. In: International Conference on Computer Aided Verification. pp. 176–185. Springer (1990). https://doi.org/10.1007/BFb0023731
22. Hansen, H., Lin, S.W., Liu, Y., Nguyen, T.K., Sun, J.: Diamonds are a girl's best friend: Partial order reduction for timed automata with abstractions. In: International Conference on Computer Aided Verification. pp. 391–406. Springer (2014)
23. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley (2003)
24. Holzmann, G.J.: The model checker SPIN. IEEE Transactions on software engineering **23**(5), 279–295 (1997)
25. Jensen, J., Nielsen, T., Østergaard, L., Srba, J.: TAPAAL and reachability analysis of P/T nets. LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) **9930**, 307–318 (2016). https://doi.org/10.1007/978-3-662-53401-4_16

26. Kordon, F., Garavel, H., Hillah, L.M., Hulin-Hubard, F., Amparore, E., Berthomieu, B., Biswal, S., Donatelli, D., Galla, F., Ciardo, G., Dal Zilio, S., Jensen, P., He, C., Le Botlan, D., Li, S., Miner, A., Srba, J., Thierry-Mieg, Y.: Complete Results for the 2020 Edition of the Model Checking Contest. http://mcc.lip6.fr/2020/results.php (June 2020)
27. Lehmann, A., Lohmann, N., Wolf, K.: Stubborn sets for simple linear time properties. In: International Conference on Application and Theory of Petri Nets and Concurrency. pp. 228–247. Springer (2012)
28. Liebke, T.: Büchi-automata guided partial order reduction for LTL. In: PNSE@ Petri Nets. pp. 147–166 (2020)
29. Liebke, T., Wolf, K.: Verification of token-scaling models using an under-approximation (2020)
30. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: International Conference on Computer Aided Verification. pp. 164–177. Springer (1992)
31. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (1989)
32. Peled, D.: All from one, one for all: on model checking using representatives. In: Computer Aided Verification, pp. 409–423. Springer Berlin Heidelberg (1993). https://doi.org/10.1007/3-540-56922-7_34, https://doi.org/10.1007/3-540-56922-7_34
33. Petri, C.A.: Communication with automata. Ph.D. thesis, Universität Hamburg (1966)
34. Schmidt, K.: Stubborn sets for standard properties. In: Lecture Notes in Computer Science, pp. 46–65. Springer Berlin Heidelberg (1999). https://doi.org/10.1007/3-540-48745-x_4, https://doi.org/10.1007/3-540-48745-x_4
35. Schmidt, K.: How to calculate symmetries of Petri nets. Acta Informatica **36**(7), 545–590 (2000)
36. Schmidt, K.: Integrating low level symmetries into reachability analysis. In: Graf, S., Schwartzbach, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 315–330. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
37. Schmidt, K.: Narrowing Petri net state spaces using the state equation. Fundamenta Informaticae **47**(3-4), 325–335 (2001)
38. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM Journal on Computing **1**(2), 146–160 (Jun 1972). https://doi.org/10.1137/0201010, https://doi.org/10.1137/0201010
39. Thierry-Mieg, Y.: Symbolic model-checking using ITS-tools. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 231–237. Springer Berlin Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_20, https://doi.org/10.1007/978-3-662-46681-0_20
40. Thierry-Mieg, Y.: Structural reductions revisited (2020)
41. Thierry-Mieg, Y.: pnmcc-models-2020. https://github.com/yanntm/pnmcc-models-2020 (2021), accessed: 2021-05-26
42. Ulrik, N.J., Virenfeldt, S.M.: Extending TAPAAL with LTL model checking. Tech. rep., Aalborg University (2020)
43. Valmari, A.: Stubborn sets for reduced state space generation. In: International Conference on Application and Theory of Petri Nets. pp. 491–515. Springer (1989)
44. Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design **1**(4), 297–322 (1992)

45. Valmari, A.: The state explosion problem. In: Lectures on Petri Nets I: Basic Models, pp. 429–528. Springer Berlin Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_21, https://doi.org/10.1007/3-540-65306-6_21
46. Valmari, A., Vogler, W.: Fair testing and stubborn sets. In: Model Checking Software, pp. 225–243. Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-32582-8_16, https://doi.org/10.1007/978-3-319-32582-8_16
47. Vardi, M.Y.: Automata-theoretic model checking revisited. In: Lecture Notes in Computer Science, pp. 137–150. Springer Berlin Heidelberg (2007). https://doi.org/10.1007/978-3-540-69738-1_10, https://doi.org/10.1007/978-3-540-69738-1_10
48. Wolf, K.: Petri net model checking with LoLA 2. In: Khomenko, V., Roux, O.H. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 351–362. Springer International Publishing, Cham (2018)