Automata-Driven Partial Order Reduction and Guided Search for LTL Model Checking

Peter Gjøl Jensen, Jiří Srba, Nikolaj Jensen Ulrik, and Simon Mejlby Virenfeldt

Department of Computer Science
Aalborg University, Denmark

Abstract. In LTL model checking, a system model is synchronized using the product construction with Büchi automaton representing all runs that invalidate a given LTL formula. An existence of a run with infinitely many occurrences of an accepting state in the product automaton then provides a counter-example to the validity of the LTL formula. Classical partial order reduction methods for LTL model checking allow to considerably prune the searchable state space, however, the majority of published approaches do not use the information about the current Büchi state in the product automaton. We demonstrate that this additional information can be used to significantly improve the performance of existing techniques. In particular, we present a novel partial order method based on stubborn sets and a heuristically guided search, both driven by the information of the current state in the Büchi automaton. We implement these techniques in the model checker TAPAAL and an extensive benchmarking on the dataset of Petri net models and LTL formulae from the 2021 Model Checking Contest documents that the combination of the automata-driven stubborn set reduction and heuristic search improves the state-of-the-art techniques by a significant margin.

1 Introduction

The state space explosion problem is one of the main barriers to model checking of large systems as the number of reachable states can be exponentially larger than the size of a high-level system description in a formalism like e.g. a Petri net [32]. Addressing this problem has been the subject of much research, with directions including partial order reductions [20,30,39], symbolic model checking [3,8], guided searches using heuristics [14,15], and symmetry reductions [9,35]. Some system description languages afford specialized techniques in addition to the above. For example, state space explosion of Petri nets can be addressed with structural reductions [4,17,29].

We focus on partial order reductions, a family of techniques designed to prune the state space search that arises from interleaving executions of concurrently running system components. An important category of partial order reduction techniques are the ample set [30], persistent set [20], and in particular the stubborn set methods [40] which are the main focus of the paper. The goal of the techniques is, given a specific state, to determine a subset of actions to explore such that all representative executions are preserved with respect to the desired property. Partial order reduction techniques are supported in several well-established tools, e.g. Tapaal [11], LoLA 2 [44], and Spin [22], and have proven to be useful in practice [4,23,26].

The main approach to Linear Temporal Logic (LTL) model checking [33] is based on a translation of the negation of an LTL formula into a Nondeterministic Büchi Automaton (NBA) and then synchronizing it with the system being verified. The goal is then to find a reachable accepting cycle in the synchronized product. While much research has been done on optimizing the construction of NBAs [1,16,43], and on the state space reductions described above, only few state space techniques take the Büchi automaton into account. For example, the classical next-free LTL preserving partial order method by Valmari [40] is based only on the syntax of the formula and is completely agnostic to the choice of verification algorithm and the Büchi state in the product automaton [41]. Some of the work done within the field of stubborn sets includes a specialized, automata-driven approach for a subclass of LTL formulae called simple LTL formulae [26], and more recently Liebke [27] introduced an automaton-based stubborn set approach for the full LTL logic. While his method is theoretically interesting, no implementation and experimental evaluation is available yet.

During the state-space exploration, the choice of which successor state to be explored first, has a large impact on the performance of depth-first algorithms for LTL model checking such as Nested Depth First Search (NDFS) [10] and Tarjan's algorithm [18]. A poor choice of successor can cause a lot of time to be wasted by exploring executions where accepting cycles do not exist. A way of addressing this problem is by using heuristics to guide the search in a direction that is more likely to be relevant for the given property. Previous work in this direction includes [13,14] in which A^* is used as a search algorithm with heuristics based on finite state machine representations, and [24] presents a best-first search algorithm using a syntax-driven heuristic, both focusing on reachability properties. To the best of our knowledge, heuristic search techniques for LTL and in particular based on the information of the current Büchi state, have not yet been systematically explored.

We contribute with a novel automata-driven stubborn set partial order method and automata-driven heuristics for guided search for model checking of LTL formulae on Petri nets. The stubborn set method is a nontrivial extension of the stubborn set technique for reachability analysis presented in [4]. This new method looks at the local structure of the NBA and considers as stubborn all actions that can cause the change of NBA state. The guided search is based on the heuristics of [24] describing the distance between a state (marking) and the satisfaction of a formula. We extend this method such that in nonaccepting NBA states we estimate the distance to possible accepting states where we can progress. Common to our techniques is the desire to leave nonaccepting NBA states as quickly as possible in order to find an accepting state earlier than otherwise.

We provide an implementation of these techniques as an extension of the open-source engine `verifypn` [24] used in the model checker Tapaal [11]. We evaluate its performance using the LTL dataset of the 2021 edition of the Model Checking Contest (MCC) [25] and compare it to the baseline LTL model checker implementing the Tarjan's algorithm [18], as well as the classical stubborn set method of Valmari [40,41] and the most recent automata-driven partial order technique of Liebke [27]. We implemented all these approaches in the Tapaal framework and conclude that while the Valmari's as well as Liebke's method considerably improve the performance of the baseline Tarjan's algorithm (and Liebke's approach is performing in general better than the classical reduction), our automata-driven approach improves the performance a degree further, in particular when combined with the heuristic search. Finally, we compare our implementation with the ITS-Tools model checker [38] that scored second after Tapaal at the 2021 Model Checking Contest [25]. We conclude that while ITS-Tools solves 87.8% of all LTL queries in the benchmark, our tool with automata-driven partial order reduction and heuristic search answers 94% of all queries.

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 [6,21] and more recently for timed games [7]. Regarding LTL model checking, the classical approaches for partial order reduction by Valmari [40,41] do not consider the Büchi state that is a part of the product system where we search for an accepting cycle. The initial work by Peled, Valmari and Kokkarinen [31] on automata-driven reduction received only little attention but it was recently revived by Liebke [27] for the use in LTL model checking, based on the insight from [26]. Liebke's idea is to design a stubborn set reduction so that sequences of non-stubborn actions cannot change the current Büchi state, allowing him to weaken and drop some requirements used in the classical partial order approach for LTL. Even though theoretically promising, the approach has not yet been implemented and experimentally evaluated. While our method relies on similar ideas as [27], the approaches differ in how we handle the looping formula of Büchi states: Liebke's method introduces more stubborn actions related to the looping formula whereas our method only adds stubborn actions for the formulae that change Büchi state (and possibly for the implicit formula leading to a sink state). We moreover implement both the classical and Liebke's techniques and compare them to our approach on a large benchmark of LTL formulae for Petri net model.

In [14] 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 our work that guides the search towards any form of state change in the NBA. The work in [14] assumes that individual (fixed number of) processes are given as finite state machines, an approach that is less general than Petri nets. Another approach to guided search is presented in [36] where state equations are used to guide the search, but it has not yet been extended to LTL model checking and it is computationally more demanding. In contrast, we emphasize simple heuristics that are faster to compute and efficient on a large number of models.

2 Preliminaries

We now define basic concepts of LTL model checking and recall the Petri net model. Let $\mathbb{N}^0$ denote the natural numbers including zero and let ∞ be such that $x < \infty$ for all $x \in \mathbb{N}^0$. By *tt* and *ff* we denote true and false, respectively.

2.1 Labelled Transition Systems

Let AP be a fixed set of *atomic propositions*. A Labelled Transition System (LTS) with propositions is a tuple $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ where

- S is a 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 $s'' \xrightarrow{w} s'$ where $\alpha \in \Sigma$ and $w \in \Sigma^*$. 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 an 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 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) [33] is given by

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

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 $\sigma \models \varphi$, according to the following inductive definition:

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

Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ 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 1. Figure 1a illustrates an LTS $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ with the set of actions $\Sigma = \{\alpha, \beta\}$ and the set of atomic propositions $AP = \{a, b\}$. The initial state s_0 satisfies the formula $\mathsf{FG}(\neg a \vee b)$ as every infinite run either loops between s_0 and s_1 (and then satisfies $\mathsf{G}\neg a$ already from the initial state) or it loops in s_3 (and then it satisfies $\mathsf{FG}b$).

2.3 Nondeterministic Büchi Automata

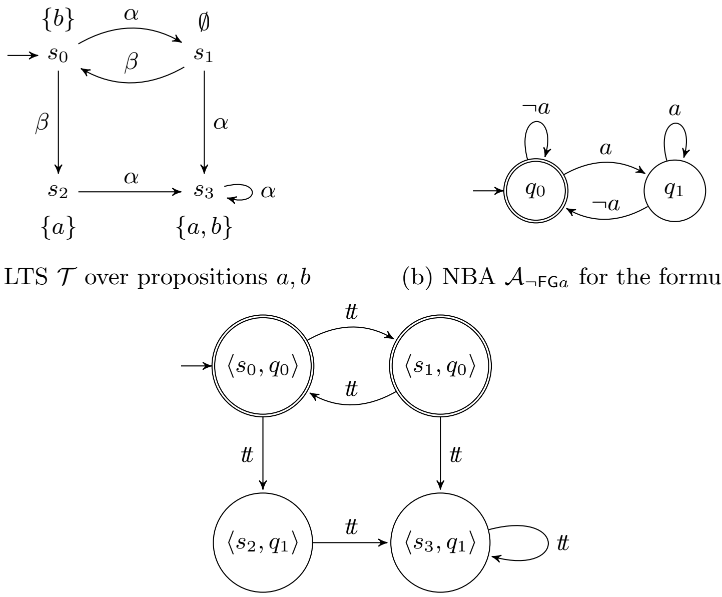
The standard approach for verifying whether $s \models \varphi$ for some state s and LTL formula φ seeks to find a counterexample to φ in the system synchronized with a Nondeterministic Büchi Automaton (NBA) equivalent to $\neg\varphi$ (see e.g. [2]). Before we define NBA, we introduce a logics for the propositions we may find as guards in the NBA. We let $\mathcal{B}(AP)$ denote the set of propositions over the set of atomic propositions AP, given by the grammar

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

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:

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

For a proposition $b \in \mathcal{B}(AP)$ and an 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 $[\![b]\!] = \{A \in 2^{AP} \mid A \models b\}$. We also write $b_1 = b_2$ iff $[\![b_1]\!] = [\![b_2]\!]$.



(a) LTS $\mathcal{T}$ over propositions a, b

(b) NBA $\mathcal{A}_{\neg\mathsf{FG}a}$ for the formula $\neg\mathsf{FG}a$

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

Fig. 1: Example LTS $\mathcal{T}$ and NBA $\mathcal{A}_{\neg\mathsf{FG}a}$; $\mathcal{T} \not\models \mathsf{FG}a$ due to the accepting cycle $(\langle s_0, q_0\rangle\langle s_1, q_0\rangle)^\omega$ in $\mathcal{T} \otimes \mathcal{A}_{\neg\mathsf{FG}a}$.

A *Nondeterministic Büchi Automaton* (NBA) is a tuple $\mathcal{A} = (Q, \delta, Q_0, F)$ where

- Q is a set of *states*,
- $\delta \subseteq Q \times \mathcal{B}(AP) \times Q$ is a *transition relation* such that for each $q \in Q$, there exist 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 consider only NBAs in a normal form so that 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 a single transition $q \xrightarrow{b \vee b'} q'$. For a state $q \in Q$ we define the set of *progressing propositions* as $\mathrm{Prog}(q) = \{b \in \mathcal{B}(AP) \mid q \xrightarrow{b} q' \text{ for some } q' \in Q \setminus \{q\}\}$, and the *retarding proposition* as $\mathrm{Ret}(q) = b \in \mathcal{B}(AP)$ such that $q \xrightarrow{b} q$ or $\mathrm{Ret}(q) = \mathit{ff}$ if no such b exists.

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

- $q_0 \in Q_0$,

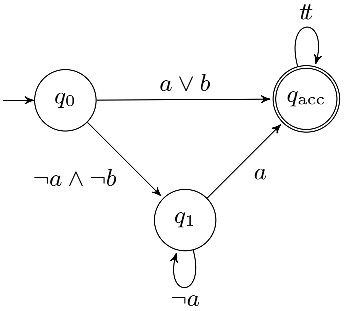


Fig. 2: NBA $\mathcal{A}_\varphi$ where $\varphi = ((\mathsf{G}a)\,\mathsf{U}\,(\mathsf{F}a)) \vee b$ with complex edge propositions

- $q_i \xrightarrow{b_i} q_{i+1}$ and $A_i \models b_i$ for all $i \geq 0$, and
- $q_i \in F$ for infinitely many $i \geq 0$.

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 well-known result.

Theorem 1 [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. 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 the retarding proposition ff.The set of progressing propositions of q_1 is the singleton set $\text{Prog}(q_1) = \{a\}$, and the retarding proposition is $\text{Ret}(q_1) = \neg a$.

From Theorem 1 we know that any infinite word σ that satisfies φ must be accepted by $\mathcal{A}_\varphi$ and vice versa. Recall that an LTS $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ 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 synchronize $\mathcal{T}$ with $\mathcal{A}_{\neg\varphi}$ and look for counterexamples.

Definition 1 (Product). *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS and let $\mathcal{A} = (Q, \delta, Q_0, F)$ be an NBA. Then the* product *$\mathcal{T} \otimes \mathcal{A} = (Q', \delta', Q'_0, F')$ is an NBA such that*

- *$Q' = S \times Q$,*
- *$\langle s, q\rangle \xrightarrow{tt} \langle s', q'\rangle$ if either $s \rightarrow s'$ or s is a deadlock and $s = s'$, and $q \xrightarrow{b} q'$ for some $b \in \mathcal{B}(AP)$ s.t. $s' \models b$,*
- *$Q'_0 = \{\langle s_0, q\rangle \in Q' \mid \exists q_0 \in Q_0 \,.\, q_0 \xrightarrow{b} q$ for some $b \in \mathcal{B}(AP)$ s.t. $s_0 \models b\}$, 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 [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. We use a specialized variant of Tarjan's connected component algorithm described in [18] for checking the emptiness of the product automaton.

Example 3. The LTS $\mathcal{T}$ depicted in Figure 1a does not satisfy the LTL formula $\mathsf{FG}a$. In order to show this, Figure 1b depicts 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 (with inhibitor arcs) 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 the *arc weight* function, and
- $I : (P \times T) \to \mathbb{N} \cup \{\infty\}$ is the *inhibitor arc weight* function.

A *marking* is a function $M : P \to \mathbb{N}^0$ assigning to each place a number of *tokens*. We write $\mathcal{M}(N)$ to denote the set of all markings of Petri net N. The semantics of a Petri net $N = (P, T, W, I)$ is given by the transition relation between markings such that $M \xrightarrow{t} M'$ if for all $p \in P$ we have $M(p) \geq W(p,t)$, $M(p) < I(p,t)$, and $M'(p) = M(p) - W(p,t) + W(t,p)$.

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 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 and the set AP of atomic propositions is formed by 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 $\oplus \in \{\cdot, +, -\}$. 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

$$M \models t \text{ iff } M \xrightarrow{t}$$
$$M \models e_1 \bowtie e_2 \text{ iff } \text{eval}_M(e_1) \bowtie \text{eval}_M(e_2)$$

and where $\text{eval}_M(p) = M(p)$, $\text{eval}_M(c) = c$ and $\text{eval}_M(e_1 \oplus e_2) = \text{eval}_M(e_1) \oplus \text{eval}_M(e_2)$.

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.

3 Automata-Guided Partial Order Reduction

Partial order reductions are techniques that address the state space explosion problem by reducing the number of interleavings of concurrent actions and exploring only their representative permutations; this can result in exponential reductions in the size of the state space (see e.g. [40,42]). We shall now present our approach improving the classical stubborn set partial order technique [40,41] for LTL without the next operator. We adapt and extend the ideas of the reachability-preserving stubborn set construction from [34,4,7] to automata-driven technique for the full LTL logic. First, we prove the formal correctness of the method on the low level formalism of labelled transition systems and later on we specialize it to Petri nets.

3.1 Automata-Driven Stubborn Set Method for LTL

The basic idea of our approach is to apply the reachability-preserving stubborn set method from [34,4,7], where the reachability problem is the proposition $\bigvee_{b \in \text{Prog}(q)} b$ for Büchi state q. In order to make this work for the full LTL logic, we have to do further considerations.

In the rest of this section, let $\text{Sink}(q) = \neg(\bigvee_{b \in \text{Prog}(q)} b \vee \text{Ret}(q))$ be the *sink state proposition*. We note that $(\vee_{b \in \text{Prog}(q)} b) \vee \text{Ret}(q) \vee \text{Sink}(q) = \mathit{tt}$ for any Büchi state q. In order to preserve correctness of the method for LTL, we require that our stubborn sets do not contain unsafe actions, which are actions that can cause some progressing proposition to become satisfied.

Definition 2 (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 all $w \in (\Sigma \setminus \{\alpha\})^*$, 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* $\text{Safe}(s, b) \subseteq \text{Safe}(s, q)$ *for all propositions $b \in \text{Prog}(q) \cup \{\text{Sink}(q)\}$. Actions from the set* $\text{Safe}(s, q)$ *are called* safe *in the product state $\langle s, q \rangle$.*

The property of a safe action α is that if in a state s of an LTS we execute a sequence of actions w after which we do not satisfy b then executing α first followed by w does not satisfy b either. In particular, when w is empty, if $s \not\models b$ and $s \xrightarrow{\alpha} s'$, then $s' \not\models b$. The idea of safe actions is inspired by a stubborn set technique for games [7] but adapted to our LTL model checking problem.

The main characteristics of our automata-driven method is that the partial order reduction no longer only depends on the current LTS state, but we also consider the NBA state we are in at the moment. For this reason, we formally define a reduction on the product state space.

Definition 3 (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. A* product reduction *is a function $St : S \times Q \rightarrow 2^{\Sigma}$. Let $\mathcal{T} \otimes_{St} \mathcal{A}$ be the* reduced product *of the product $\mathcal{T} \otimes \mathcal{A}$ restricted by St such that $\langle s, q \rangle \rightarrow_{St} \langle s', q' \rangle$ in $\mathcal{T} \otimes_{St} \mathcal{A}$ if and only if $\langle s, q \rangle \rightarrow \langle s', q' \rangle$ in $\mathcal{T} \otimes \mathcal{A}$ and $s \xrightarrow{\alpha} s'$ for some $\alpha \in St(s, q)$.*

We can now present the list of axioms required by our stubborn set method for LTL model checking.

Definition 4 (Axioms on product reduction). *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 following four axioms are defined as follows (universally quantified for all $s \in S$ and all $q \in Q$):*

COM *If $\alpha \in St(s, q)$ and $\alpha_1, \alpha_2, \ldots, \alpha_n \in \overline{St(s, q)}^*$ and $s \xrightarrow{\alpha_1 \ldots \alpha_n \alpha} s'$ 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 $s' \not\models b$ for all $b \in \mathrm{Prog}(q)$.*

SAFE *Either $\mathrm{en}(s) \cap St(s, q) \subseteq \mathrm{Safe}(s, q)$ and $s \not\models b$ for all propositions $b \in \mathrm{Prog}(q) \cup \{\mathrm{Sink}(q)\}$, or $St(s, q) = \Sigma$.*

KEY *If $\mathrm{en}(s) \neq \emptyset$ and $q \in F$, then there is some* key action *$\alpha_{\mathrm{key}} \in St(s, q)$ such that whenever $s \xrightarrow{\alpha_1 \ldots \alpha_n} s_n$ for $\alpha_1, \ldots, \alpha_n \in \overline{St(s, q)}^*$ then $s_n \xrightarrow{\alpha_{\mathrm{key}}}$.*

Axioms **COM** and **R** are adapted from the standard reachability-preserving stubborn set methods, see e.g. [4,34], and made sensitive to preserve at least one execution (under the stubborn actions from the set $St(s, q)$) to each configuration where some of the progressing formulae becomes enabled. The axiom **SAFE** ensures that we do not prune any outgoing transition ($St(s, q) = \Sigma$) if some unsafe stubborn action is enabled or if some progressing proposition is already satisfied. Note that while the sink state proposition is important for the axiom **SAFE**, it is not important for **R**. Finally, the axiom **KEY** asserts that there is a key stubborn action in accepting Büchi states, ensuring that we preserve at least one infinite accepting run.

We are now ready to prove the main correctness theorem for our stubborn set method for LTL model checking.

Theorem 3. *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS, $\mathcal{A} = (Q, \delta, Q_0, F)$ be an NBA, $St : S \times Q \rightarrow 2^{\Sigma}$ be a product reduction satisfying* **COM**, **R**, **SAFE**, and **KEY***, 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.*

3.2 Stubborn Sets for LTL Model Checking on Petri Nets

We now present a syntax-driven method for efficiently computing stubborn sets for markings in a Petri net. We start by defining a COM-saturated set of Petri net transitions, using the increasing presets and decreasing postsets of transitions (see also [4]).

Definition 5 (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 $p^{\circ} \subseteq T'$, and*
2. *for all $t \in T'$, if $M \overset{t}{\not\rightarrow}$ 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 $p^{-} \subseteq T'$.*

Intuitively, Condition 1 requires that 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' and must be a part of the set 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 and must be a part of T'. Condition 2 states that a transition t' that can cause a disabled transition t to become enabled cannot be commuted with t and must be added to 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. The lemma moreover shows that an enabled stubborn transition cannot be disabled by firing any sequence of nonstubborn transitions.

Lemma 1. *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. For all $t \in T'$ and all $t_1, \ldots, t_n \in T \setminus T'$*

a) if $M \xrightarrow{t_1 \ldots t_n t} M'$ then $M \xrightarrow{t t_1 \ldots t_n} M'$, and
b) if $M \xrightarrow{t_1 \ldots t_n} M'$ and $M \xrightarrow{t}$ then $M' \xrightarrow{t}$.

The conditions in Definition 5 give rise to a straightforward closure algorithm that starting from some set of transitions T' iteratively includes additional transitions as required by Conditions 1 and 2 until the set of transitions gets saturated, however, due to the choice of the place p in Condition 2, it is not guaranteed that we always get the same COM-saturated set.

The next definition of increasing and decreasing transitions of an arithmetic expression is needed for constructing safe stubborn sets and for axiom **R**.

Definition 6 (Increasing/decreasing transitions). *Let $N = (P, T, W, I)$ be a Petri net and let $e \in E$ be an arithmetic expression. The sets of* increasing transitions $\mathrm{incr}(e)$ *and* decreasing transitions $\mathrm{decr}(e)$ *are recursively defined by:* $\mathrm{incr}(p) = {}^{+}p$, $\mathrm{decr}(p) = p^{-}$, $\mathrm{incr}(c) = \mathrm{decr}(c) = \emptyset$, $\mathrm{incr}(e_1 + e_2) = \mathrm{incr}(e_1) \cup \mathrm{incr}(e_2)$, $\mathrm{decr}(e_1 + e_2) = \mathrm{decr}(e_1) \cup \mathrm{decr}(e_2)$, $\mathrm{incr}(e_1 - e_2) = \mathrm{incr}(e_1) \cup \mathrm{decr}(e_2)$, $\mathrm{decr}(e_1 - e_2) = \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2)$, $\mathrm{decr}(e_1 \cdot e_2) = \mathrm{incr}(e_1 \cdot e_2) = \mathrm{incr}(e_1) \cup \mathrm{incr}(e_2) \cup \mathrm{decr}(e_1) \cup \mathrm{decr}(e_2)$.

The sets $\mathrm{incr}(e)$ and $\mathrm{decr}(e)$ contain all transitions that can possibly increase, resp. decrease the value of the expression $e \in E$; this is formalized as follows.

Lemma 2 [4]. *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 $M \xrightarrow{t_1 \ldots t_n} M'$ for $t_1, \ldots, t_n \in T$. If $\mathrm{eval}_M(e) < \mathrm{eval}_{M'}(e)$ then there is i such that $t_i \in \mathrm{incr}(e)$, and if $\mathrm{eval}_M(e) > \mathrm{eval}_{M'}(e)$ then there is i such that $t_i \in \mathrm{decr}(e)$.*

In order to preserve the axiom **SAFE**, we shall define the notion of *strictly interesting transitions*, i.e. those transitions that have the potential to change a value of a given Boolean combination of atomic propositions. The purpose of the set of strictly interesting transitions A^{+}_{M} given in the following definition is to efficiently compute syntactic over-approximations of all unsafe transitions in a marking M.

Definition 7 (Strictly interesting transitions). *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* strictly interesting transitions of b *is defined as*

$$
\begin{aligned}
A^{+}_{M}(tt) &= A^{+}_{M}(ff) = \emptyset \\
A^{+}_{M}(e_1 < e_2) &= A^{+}_{M}(e_1 \leq e_2) = \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2) \\
A^{+}_{M}(e_1 > e_2) &= A^{+}_{M}(e_1 \geq e_2) = \mathrm{incr}(e_1) \cup \mathrm{decr}(e_2) \\
A^{+}_{M}(e_1 = e_2) &= \begin{cases} \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2) & \text{if } \mathrm{eval}_M(e_1) > \mathrm{eval}_M(e_2) \\ \mathrm{incr}(e_1) \cup \mathrm{decr}(e_2) & \text{if } \mathrm{eval}_M(e_1) < \mathrm{eval}_M(e_2) \end{cases} \\
A^{+}_{M}(e_1 \neq e_2) &= \mathrm{incr}(e_1) \cup \mathrm{decr}(e_2) \cup \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2) \\
A^{+}_{M}(b_1 \vee b_2) &= A^{+}(b_1 \wedge b_2) = A^{+}_{M}(b_1) \cup A^{+}_{M}(b_2)
\end{aligned}
$$

$$
\begin{aligned}
A^{+}_{M}(\neg(e_1 < e_2)) &= A^{+}_{M}(e_1 \geq e_2) & \qquad A^{+}_{M}(\neg(e_1 \leq e_2)) &= A^{+}_{M}(e_1 > e_2) \\
A^{+}_{M}(\neg(e_1 > e_2)) &= A^{+}_{M}(e_1 \leq e_2) & \qquad A^{+}_{M}(\neg(e_1 \geq e_2)) &= A^{+}_{M}(e_1 < e_2) \\
A^{+}_{M}(\neg(e_1 = e_2)) &= A^{+}_{M}(e_1 \neq e_2) & \qquad A^{+}_{M}(\neg(e_1 \neq e_2)) &= A^{+}_{M}(e_1 = e_2) \\
A^{+}_{M}(\neg(b_1 \wedge b_2)) &= A^{+}_{M}(\neg b_1 \vee \neg b_2) & \qquad A^{+}_{M}(\neg(b_1 \vee b_2)) &= A^{+}_{M}(\neg b_1 \wedge \neg b_2)
\end{aligned}
$$

Lemma 3. *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 any $t \notin A^+_M(b)$ and any $w \in (T \setminus \{t\})^*$, if $M \xrightarrow{w} M'$, $M \xrightarrow{tw} M''$, and $M' \not\models b$, then $M'' \not\models b$.*

In order to satisfy axiom **R**, we can define a weaker notion of interesting transitions as used in [4].

Definition 8 (Interesting transitions). *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*

$$A_M(b) = \begin{cases} A_M(b_i) & \text{for some } i \text{ where } M \not\models b_i \text{ if } b = b_1 \land b_2 \\ A^+_M(b) & \text{otherwise.} \end{cases}$$

Lemma 4 [4]. *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 $M' \not\models b$.*

We now state our main theorem that allows for a syntax-driven implementation of automata-driven stubborn set reduction for full LTL on Petri nets.

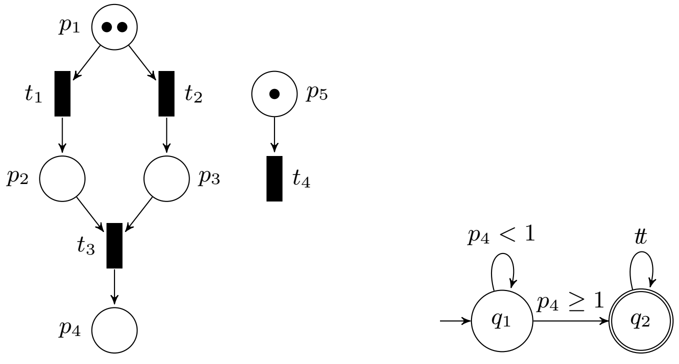
Theorem 4. *Let $N = (P, T, W, I)$ be a Petri net, $\mathcal{A} = (Q, \delta, Q_0, F)$ be an NBA, and $St : \mathcal{M}(N) \times Q \to 2^T$ be a product reduction that for all markings $M \in \mathcal{M}(N)$ and states $q \in Q$ satisfies*

1. *$St(M, q)$ is a COM-saturated set in M, and*
2. *$\bigcup_{b \in \mathrm{Prog}(q)} A_M(b) \subseteq St(M, q)$, and*
3. *either $\mathrm{en}(M) \cap St(M, q) \subseteq T \setminus A^+_M(b)$ and $M \not\models b$ for all $b \in \mathrm{Prog}(q) \cup \{\mathrm{Sink}(q)\}$, or $St(M, q) = T$, and*
4. *if $\mathrm{en}(M) \neq \emptyset$ and $q \in F$ then $\mathrm{en}(M) \cap St(M, q) \neq \emptyset$.*

Then St satisfies the axioms **COM**, **R**, **SAFE** *and* **KEY**.

Proof. By Lemma 3, Condition 3 ensures axiom **SAFE**. By Lemma 4, Condition 2 ensures **R**, and by Lemma 1 part a) our Condition 1 ensures **COM**. Condition 4 ensures **KEY** by Lemma 1 part b) as $St(M, q)$ is COM-saturated. □

Hence by Theorem 3, any reduction satisfying the conditions of Theorem 4 is LTL-preserving stubborn set reduction. The theorem also provides an algorithmic way to generate the LTL-preserving stubborn set $St(M, q)$. First, if some progressing proposition $b \in \mathrm{Prog}(q) \cup \{\mathrm{Sink}(q)\}$ is satisfied by M, then the set of all transitions is returned. Otherwise, the COM-saturation algorithm is run on $A_M(b)$ for $b \in \mathrm{Prog}(q)$ to obtain a stubborn set satisfying **COM** and **R**. To ensure **SAFE** is satisfied, the resulting stubborn set is checked for whether there is any overlap with enabled strictly 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. 3: Example of our stubborn set method applied to Petri nets

If $q \in F$ and $\text{en}(M) \cap St(M,q) = \emptyset$, an arbitrary enabled transition is added to $St(M,q)$ to ensure **KEY** is not violated, and the previous checks for **COM** and **SAFE** are repeated.

Example 4. We shall now give an example of the computation of a stubborn set for the Petri net shown in Figure 3a (here we use the classical graphical notation for Petri nets where circles represent places and rectangles transitions; the default weight of all arcs is 1) and the NBA in Figure 3b. In the initial marking M_0, the enabled transitions are $\text{en}(M_0) = \{t_1, t_2, t_4\}$. When computing the stubborn set $St(M_0, q_1)$ we note that the progressing formula $p_4 \geq 1$ is not satisfied, and the sink formula is ff, so a reduction is possible. First, we determine the set of interesting transitions

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

Next, we determine a COM-saturated set that contains t_3 which turns out to be $St(M_0, q_1) = \{t_1, t_2, t_3\}$. We now ensure that none of the enabled transitions in this set are strictly interesting. Indeed, the only interesting transition t_3 is not enabled, thus $\text{en}(M_0) \cap St(M_0, q_1) \subseteq T \setminus A^+_{M_0}(p_4 \geq 1)$ and therefore **SAFE** is satisfied. We can so conclude that $St(M_0, q_1) = \{t_1, t_2, t_3\}$ is a valid stubborn set. Since the enabled transition t_4 is not in the stubborn set, we avoid exploring the interleavings with the transition t_4, reducing the size of the state space that we search.

4 Automata-Driven Guided Search

When performing explicit state model checking using depth-first search algorithms, such as the on-the-fly variant of Tarjan's algorithm [18,37] used for LTL model checking, the order in which we explore the successors may significantly influence how fast we can find an accepting cycle and possibly avoid exploring

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

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

Fig. 4: Heuristic distance function between a marking and a LTL formula

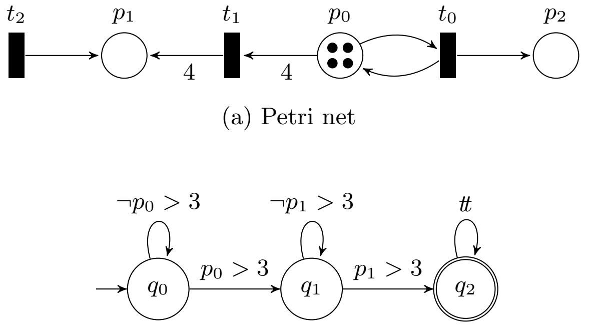
parts of the state space where such a cycle is not present. We shall now design an automata-driven heuristic approach that aims to guide the search to the parts of the state space where a cycle is more likely to be present.

In a marking M, the heuristic function assigns a nonnegative number to each M' where $M \to M'$ such that the markings with smaller numbers are explored first as they are believed to be more likely to lead us to an accepting cycle.

We first extend the distance-based heuristic for reachability [24] to the full LTL logic. The idea of this heuristic is to provide a distance from one marking to another by counting how many tokens must be added/removed in order to make the two markings equal—this idea is then extended to the atomic propositions. Our distance measure is calculated using the recursive function dist given in Figure 4. For a Petri net N, an LTL formula φ, and a marking $M \in \mathcal{M}(N)$ our heuristic function $\mathrm{dist}(M, \varphi, \mathit{tt})$ returns the distance of the marking M to satisfying the LTL formula φ.

The following example shows that the distance-based heuristic can be already useful by itself for guiding the state space search, even without considering the current state in the Büchi automaton.

Example 5. Consider the Petri net N in Figure 5a and the LTL formula $\varphi = \neg\mathsf{F}(p_0 > 3 \wedge \mathsf{X}\mathsf{F} p_1 > 3)$. We want to determine whether $N \models \varphi$. We let M_i denote the marking we reach after firing the transition t_i. Then $\mathrm{dist}(M_0, \varphi, \mathit{tt}) = 4$,



(a) Petri net

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

Fig. 5: Example system where heuristics are advantageous when considering the LTL formula $\varphi = \neg\mathsf{F}(p_0 > 3 \land \mathsf{XF} p_1 > 3)$.

$\mathrm{dist}(M_1, \varphi, \mathit{tt}) = 4$, and $\mathrm{dist}(M_2, \varphi, \mathit{tt}) = 3$. The heuristic prioritises to first follow the transition t_2, leading us one step closer to satisfying $\mathsf{F} p_1 > 3$. Repeating the procedure, after three additional firings of t_2, we end up in a marking with $M(p_1) = 4$ where we satisfy the LTL formula.

As a next step, we use the distance metrics to design a more efficient automata-driven heuristic technique that takes the current Büchi state into consideration. Instead of looking at the entire LTL formula, we consider 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 try to leave the current state as fast as possible in order to move closer to an accepting Büchi state. As such, we prioritise transitions that are more likely to enable progressing formulae, including the consideration how far is the resulting NBA state 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 some $q' \in F$ (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 marking M' of M as the minimum of $(1 + \mathrm{BFS}(q')) \cdot \mathrm{dist}(M', b, \mathit{ff})$ over all $q' \in Q$ where $q \xrightarrow{b} q'$.

Example 6. Let us again consider the Petri net in Figure 5a, and the NBA corresponding to $\neg\varphi$, presented in Figure 5b. In the product construction given in Definition 1, we create the initial Büchi states of the product state space; as the initial marking satisfies the progressing proposition $p_0 > 3$ but not the retarding proposition $\neg p_0 > 3$, there is only one initial product state (where the Büchi automaton is in the state q_1). Now we calculate the heuristic value where, as before, M_i is the marking resulting from firing the transition t_i. There is only one progressing proposition, so the heuristic value is given by $(1+\mathrm{BFS}(q_1)) \cdot \mathrm{dist}(M_i, p_1 > 3, \mathit{ff})$. This gives the values $2 \cdot \mathrm{dist}(M_0, p_1 > 3, \mathit{ff}) = 8$, $2 \cdot \mathrm{dist}(M_1, p_1 > 3, \mathit{ff}) = 0$, and $2 \cdot \mathrm{dist}(M_2, p_1 > 3, \mathit{ff}) = 6$ for the transitions t_0, t_1 and t_2, respectively. The transition with the highest priority is t_1 which immediately leads to a marking satisfying $p_1 > 3$ and we move to the accepting state. This illustrates the advantage of automata-driven heuristics over the distance-based one relying on the whole LTL formula, namely that it can disregard parts of the formula that are not relevant at the moment.

5 Experimental Evaluation

We shall now evaluate the performance of our automata-driven techniques for partial order reduction and guided search on the benchmark of Petri net models and LTL formulae from the 2021 edition of the Model Checking Contest (MCC) [25]. The benchmark consists of 1181 P/T nets modelling academic and industrial use cases, each with 32 LTL formulae split evenly between cardinality formulae and fireability formulae. This gives a total of 37 792 queries for our evaluation, each executed with 15 min timeout and 16 GiB of available memory on one core of an AMD Opteron 6376 processor.

We implemented our automata-driven techniques described in this paper as an extension of the verification engine `verifypn` [24] that is a part of the Tapaal model checker [11]. Our LTL engine uses version 2.9.6 of the Spot library [12] for translating LTL formulae into NBAs, and a derivative of Tarjan's algorithm [18,37] for searching for accepting cycles. To speed up the verification, we also employ the query simplifications from [5] and most of the structural reductions from [4]. We moreover implemented within the `verifypn` engine the classical partial order reduction of Valmari [40,41] (referred to as Classic POR) as well as the automata-based reduction of Liebke [27] (referred to as Liebke POR) that has been theoretically studied but so far without any implementation nor experimental evaluation. In our experiments, we benchmark the baseline implementation (without any partial order reduction nor heuristic search) and our stubborn set reduction (referred to as automata-driven POR) against Classic POR and Liebke POR, both using the standard depth-first search as well as our heuristic search technique (referred to as HPOR). We also provide a full reproducibility package [19].

According to [28], the MCC benchmark contains a large number of trivial instances that all model checkers can solve without much computation effort, as well as instances that are too difficult for any model checker to solve. In our first experiment, we thus selected a subset of interesting/nontrivial instances such that our baseline implementation needed at least 30 seconds to solve them and at least one of the methods provided an answer within 15 minutes. This selection resulted in 3508 queries on which we evaluate the techniques.

Table 1a shows the number of answers obtained for each method without employing the heuristic search and Table 1b with heuristic search (we report here on the automata-driven heuristics only as it provides 233 additional answers compared to the distance-based one). The first observation is that our heuristic search technique gives for all of the partial order methods about 20% improvement in the number of answered queries. Second, while both classic and Liebke's

Table 1: Number of answered positive and negative queries, total number of queries and percentage compared to number of solved queries by at least one method (3508 in total)

(a) Partial order reductions without heuristic search

	Positive	Negative	Total	Solved
Baseline (no POR)	501	1708	2209	61.5 %
Classic POR	527	1846	2373	66.1 %
Liebke POR	551	1868	2419	67.3 %
Automata-driven POR	564	2004	2568	71.5 %

(b) Partial order reductions with heuristic search

	Positive	Negative	Total	Solved
Baseline (heuristic)	496	2463	2959	82.4 %
Classic HPOR	523	2530	3053	85.0 %
Liebke HPOR	555	2512	3067	85.4 %
Automata-driven HPOR	565	2640	3205	89.2 %

partial order reduction techniques (that are essentially comparable when using heuristic search and without it Liebke solves 1.2% more queries) provide a significant 3–6% improvement in the number of answered queries over the baseline (both with and without the heuristic), our method achieves up to 10% improvement.

While in absolute numbers the additional points are primarily due to negative answers (where an accepting cycle exists), we can see also a similar trend in the increased number of positively answered queries. In general, positive answers are expected to be harder to obtain than negative answers, as they require disproving the existence of any counter example and hence full state space search. This is also the reason why adding a heuristic search on top of the partial order techniques can have a negative effect on the number of answered positive queries; here the search order does not matter but the heuristic search method has an overhead for computing the distance functions in every discovered marking.

Overall, while the baseline method solved only 61.5% of queries, our partial order technique in combination with the automata-driven heuristic search now answers 89.2% of queries, which is a considerable improvement and shows that the two techniques can be applied in combination in order to increase the verification performance.

In Figure 6 we focus for each method on the most difficult 1500 queries from the benchmark. For each method, we independently sort the running times (plotted on the y-axis, note the logarithmic scale) in increasing order for all

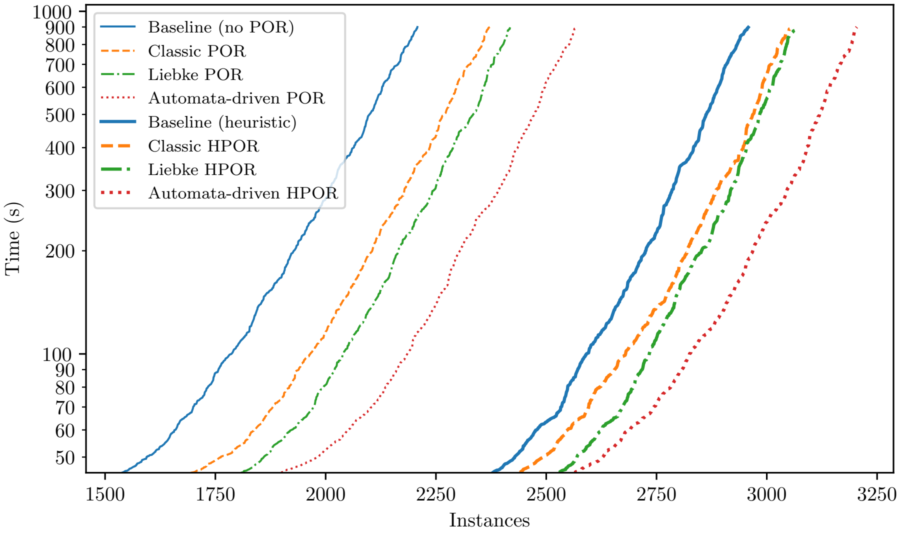


Fig. 6: Comparison of the different methods versus the baseline; on x-axis all instances sorted by the increasing running time (independently per method); on y-axis the running time (in seconds and logarithmic scaling)

Table 2: Number of answers in the MCC setup.

	Positive	Negative	Total	Solved
TAPAAL	9415	26 219	35 629	94.3 %
TAPAAL (no POR, no heuristic)	9345	25 865	35 210	93.2 %
ITS-Tools	8395	24 775	33 170	87.8 %

the query instances (plotted on the x-axes). Hence the plot does not provide a running time comparison per instance (in fact there are even a few queries that the baseline answers but not our heuristic POR method), however, it shows the overall performance trends on the whole dataset. The plot confirms with the general observation we made on the number of answered queries and moreover shows that without the heuristic search (thinner lines in the left part of the plot) Liebke's method is in general performing faster than the classic method. The addition of the heuristic search to the partial order reduction makes a significant improvement, as shown by the thick curves in the right part of the plot. Here the classic and Liebke's have more similar performance, whereas our automata-driven method most significantly profits from the addition of heuristic search.

Finally, in Table 2 we provide the comparison with the model checker ITS-Tools [38] that was second after TAPAAL in the 2021 edition of the Model Checking Contest [25]. In the MCC, 16 queries are verified in parallel with a 1 hour time out, 16 GiB memory limit and 4 available cores. The scripts that execute the verification are taken from the available virtual machines (for the details of the setup consult the MCC webpage[1]) and executed on the total of 37792 queries in the batches of 16 queries. While ITS-tools can solve 87.8% of all queries, Tapaal (the winner in 2021 contest) without partial order reduction and heuristic search answers 93.2% of all queries. The addition of our automata-driven techniques improves the score to 94.3% of answered queries, which is a very satisfactory improvement given that the MCC benchmark contains a significant percentage of models and queries that are beyond the reach of the current model checkers.

6 Conclusion

We presented two automata-driven techniques, stubborn set partial order reduction and a heuristic search method, for improving the performance of LTL model checking. The common element in these methods is that we exploit the fact that states in the product system (where we search for an accepting cycle) contain also the information about the current state of Büchi automaton. Recent work by Liebke [27] suggests a similar approach trying to weaken the classical LTL axioms for partial order reduction; we instead extend the reachability-preserving axioms to the full LTL logic. Our approach is presented first in a general way and then specialized to the Petri net model.

We implemented both the baseline Tarjan's algorithm for LTL model checking, the classical and Liebke's partial order reductions as well as our automata-driven methods and compare them on a large benchmark of LTL models from the 2021 Model Checking Contest. The conclusion is that while both the classical and Liebke's methods provide a significant performance improvement over the baseline algorithm, our automata-driven partial order technique improves the state-of-the-art techniques by another degree. Moreover, our heuristic search is clearly beneficial in combination with all partial order methods and our current best implementation in the tool Tapaal beats the second best tool in the yearly Model Checking Contest by the margin of 6.5%.

In the future work we plan to further improve the performance of our method for example for the subclass of weak Büchi automata and extend the ideas to other logics like CTL.

Acknowledgments. We thank to Yann Thierry-Mieg for creating the oracle database of correct answers for queries from the model checking contest that we used extensively for testing our implementation.

References

1. Babiak, T., Křetínský, M., Řehák, V., Strejček, J.: Ltl to büchi automata translation: Fast and more deterministic. In: Flanagan, C., König, B. (eds.) Tools and

[1] `https://mcc.lip6.fr/`

Algorithms for the Construction and Analysis of Systems. TACAS 2012. LNCS, vol. 7214, pp. 95–109. Springer, Berlin, Heidelberg (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: Tools and Algorithms for the Construction and Analysis of Systems. TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Berlin, Heidelberg (1999)
4. Boenneland, 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., Dyhr, J., Jensen, P., Johannsen, M., Srba, J.: Simplification of CTL formulae for efficient model checking of Petri nets. In: Proceedings of the 39th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets'18). LNCS, vol. 10877, pp. 143–163. Springer-Verlag (2018)
6. 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
7. 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)
8. 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)
9. Clarke, E.M., Emerson, E.A., Jha, S., Sistla, A.P.: Symmetry reductions in model checking. In: Hu, A.J., Vardi, M.Y. (eds.) Computer Aided Verification, LNCS, vol. 1427, pp. 147–158. Springer, Berlin, Heidelberg (1998). https://doi.org/10.1007/bfb0028741
10. 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). https://doi.org/10.1007/BF00121128
11. 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)
12. 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). LNCS, vol. 9938, pp. 122–129. Springer (oct 2016). https://doi.org/10.1007/978-3-319-46520-3_8
13. Edelkamp, S., Jabbar, S.: Large-scale directed model checking LTL. In: Valmari, A. (ed.) International SPIN Workshop on Model Checking of Software. LNCS, vol. 3925, pp. 1–18. Springer (2006). https://doi.org/10.1007/11691617_1
14. Edelkamp, S., Lafuente, A.L., Leue, S.: Directed explicit model checking with HSF-SPIN. In: Dwyer, M. (ed.) International SPIN Workshop on Model Checking of Software. LNCS, vol. 2057, pp. 57–79. Springer, Springer (2001)

15. Edelkamp, S., Schuppan, V., Bošnački, D., Wijs, A., Fehnker, A., Aljazzar, H.: Survey on directed model checking. In: Peled, D.A., Wooldridge, M.J. (eds.) Model Checking and Artificial Intelligence (MoChArt 2008). LNCS, vol. 5348, pp. 65–89. Springer (2008)
16. 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. LICS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3209108.3209161
17. Esparza, J., Schröter, C.: Net reductions for LTL model-checking. In: Margaria, T., Melham, T. (eds.) Correct Hardware Design and Verification Methods, LNCS, vol. 2144, pp. 310–324. Springer Berlin Heidelberg, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-44798-9_25
18. Geldenhuys, J., Valmari, A.: More efficient on-the-fly LTL verification with Tarjan's algorithm. Theoretical Computer Science **345**(1), 60–82 (2005). https://doi.org/10.1016/j.tcs.2005.07.004
19. Gjøl Jensen, P., Srba, J., Jensen Ulrik, N., Mejlby Virenfeldt, S.: Reproducibility Package: Automata-Driven Partial Order Reduction and Guided Search for LTL (Nov 2021). https://doi.org/10.5281/zenodo.5704172
20. Godefroid, P.: Using partial orders to improve automatic verification method. In: Clarke, E.M., Kurshan, R.P. (eds.) International Conference on Computer Aided Verification (CAV '90). LNCS, vol. 531, pp. 176–185. Springer (1991). https://doi.org/10.1007/BFb0023731
21. 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: Biere, A., Bloem, R. (eds.) International Conference on Computer Aided Verification (CAV 2014). pp. 391–406. LNCS, Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_26
22. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley (2003)
23. Holzmann, G.J.: The model checker SPIN. IEEE Transactions on software engineering **23**(5), 279–295 (1997). https://doi.org/10.1109/32.588521
24. 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
25. Kordon, F., Bouvier, P., Garavel, H., Hillah, L.M., Hulin-Hubard, F., Amat., N., Amparore, E., Berthomieu, B., Biswal, S., Donatelli, D., Galla, F., , Dal Zilio, S., Jensen, P., He, C., Le Botlan, D., Li, S., , Srba, J., Thierry-Mieg, ., Walner, A., Wolf, K.: Complete Results for the 2020 Edition of the Model Checking Contest. http://mcc.lip6.fr/2021/results.php (June 2021)
26. Lehmann, A., Lohmann, N., Wolf, K.: Stubborn sets for simple linear time properties. In: Haddad, S., Pomello, L. (eds.) International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS '12). LNCS, vol. 7347, pp. 228–247. Springer, Berlin, Heidelberg (2012)
27. Liebke, T.: Büchi-automata guided partial order reduction for LTL. In: PNSE@ Petri Nets. pp. 147–166 (2020)
28. Liebke, T., Wolf, K.: Taking some burden off an explicit CTL model checker. In: Donatelli, S., Haar, S. (eds.) Application and Theory of Petri Nets and Concurrency. LNCS, vol. 11522, pp. 321–341. Springer, Cham (2019)
29. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (1989). https://doi.org/10.1109/5.24143

30. Peled, D.: All from one, one for all: on model checking using representatives. In: Courcoubetis, C. (ed.) Computer Aided Verification (CAV 1993), LNCS, vol. 697, pp. 409–423. Springer Berlin Heidelberg (1993). https://doi.org/10.1007/3-540-56922-7_34
31. Peled, D.A., Valmari, A., Kokkarinen, I.: Relaxed visibility enhances partial order reduction. Formal Methods in Systems Design **19**(3), 275–289 (2001). https://doi.org/10.1023/A:1011202615884
32. Petri, C.A.: Communication with automata. Ph.D. thesis, Universität Hamburg (1966)
33. Pnueli, A.: The temporal semantics of concurrent programs. Theoretical Computer Science **13**(1), 45–60 (1981). https://doi.org/10.1016/0304-3975(81)90110-9
34. Schmidt, K.: Stubborn sets for standard properties. In: Donatelli, S., Kleijn, J. (eds.) Application and Theory of Petri Nets 1999. LNCS, vol. 1639, pp. 46–65. Springer Berlin Heidelberg (1999). 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). https://doi.org/10.1007/s002360050002
36. Schmidt, K.: Narrowing Petri net state spaces using the state equation. Fundamenta Informaticae **47**(3-4), 325–335 (2001)
37. 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`
38. Thierry-Mieg, Y.: Symbolic model-checking using ITS-tools. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS '15). LNCS, vol. 9035, pp. 231–237. Springer Berlin Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_20
39. Valmari, A.: Stubborn sets for reduced state space generation. In: Rozenberg, G. (ed.) International Conference on Application and Theory of Petri Nets. LNCS, vol. 483, pp. 491–515. Springer, Berlin, Heidelberg (1989). https://doi.org/10.1007/3-540-53863-1_36
40. Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design **1**(4), 297–322 (1992)
41. Valmari, A.: The state explosion problem. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models. LNCS, vol. 1491, pp. 429–528. Springer Berlin Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_21
42. Valmari, A., Vogler, W.: Fair testing and stubborn sets. In: Bošnački, D., Wijs, A. (eds.) Model Checking Software, LNCS, vol. 9641, pp. 225–243. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-32582-8_16
43. Vardi, M.Y.: Automata-theoretic model checking revisited. In: Cook, B., Podelski, A. (eds.) Verification, Model Checking, and Abstract Interpretation (VMCAI 2007). LNCS, vol. 4349, pp. 137–150. Springer Berlin Heidelberg (2007). https://doi.org/10.1007/978-3-540-69738-1_10
44. Wolf, K.: Petri net model checking with LoLA 2. In: Khomenko, V., Roux, O.H. (eds.) Application and Theory of Petri Nets and Concurrency (PETRI NETS 2018). LNCS, vol. 10877, pp. 351–362. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-91268-4_18

A Proofs for Section 3 (Automata-Guided Partial Order Reduction)

Theorem 3. *Let $\mathcal{T} = (S, \Sigma, \rightarrow, L, s_0)$ be an LTS, $\mathcal{A} = (Q, \delta, Q_0, F)$ be an NBA, $St : S \times Q \rightarrow 2^{\Sigma}$ be a product reduction satisfying* **COM**, **R**, **SAFE**, *and* **KEY***, 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 $\pi = \langle s_0, q_0 \rangle \langle s_1, q_1 \rangle \ldots$ be an accepting run in the reduced state space $\mathcal{T} \otimes_{St} \mathcal{A}$. If π contains no state $\langle s_i, q_i \rangle$ where s_i is a deadlock state, then π is also an accepting run in the full state space since the reduction St can only remove some actions but does not add new ones. If π contains a state $\langle s_i, q_i \rangle$ where s_i is a deadlock state in the reduced state space then since the run is accepting, either (i) q_i is accepting or (ii) some progressing proposition $b \in \mathrm{Prog}(q_i)$ is satisfied by s_i. In case (i) we know by axiom **KEY** that s_i is also a deadlock in the full state space. In case (ii) we get by axiom **SAFE** that $St(s_i, q_i)$ contains all actions and hence s_i is also a deadlock in the full state space. This implies that π is an accepting run in $\mathcal{T} \otimes \mathcal{A}$.

" $\Longrightarrow$ ": Let us assume that the full state space $\mathcal{T} \otimes \mathcal{A}$ contains an accepting run $\pi = \langle s_0, q_0 \rangle \langle s_1, q_1 \rangle \ldots$ and we shall construct an accepting run in the reduced state space $\mathcal{T} \otimes_{St} \mathcal{A}$. If the first step in the run π is executable also in $\mathcal{T} \otimes_{St} \mathcal{A}$, then we execute this step until we arrive to the first step that is not executable. Let us so w.l.o.g. assume that the first step in π from $\langle s_0, q_0 \rangle$ is not executable in $\mathcal{T} \otimes_{St} \mathcal{A}$, meaning that $St(s_0, q_0) \neq \Sigma$. There are two cases, either $q_0 \notin F$ or $q_0 \in F$.

- In case $q_0 \notin F$, the run π must change the NBA state at some point and no progressing proposition of q_0 is satisfied in s_0 (otherwise by axiom **SAFE** necessarily $St(s_0, q_0) = \Sigma$ which we assume is not the case). Let $w = \alpha_1 \alpha_2 \ldots \alpha_i \in \Sigma^*$ be a sequence of actions such that $\langle s_0, q_0 \rangle \xrightarrow{\alpha_1} \langle s_1, q_1 \rangle \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_i} \langle s_i, q_i \rangle$ is an execution on a prefix of π such that $q_i \neq q_0$ and $q_j = q_0$ for all $0 \leq j < i$. Since $q_i \neq q_0$, some s_j must satisfy a progressing proposition b where $q_0 \xrightarrow{b}$, so by axiom **R** there must be a stubborn action $\alpha \in St(s_0, q_0)$ in w. Let $w = u \alpha v$ such that $u \in \overline{St(s_0, q_0)}^*$ and $\alpha \in St(s_0, q_0)$. By axiom **COM**, we know that the fact $s_0 \xrightarrow{u\alpha} s' \xrightarrow{v} s_i$ implies that $s_0 \xrightarrow{\alpha} s'_1 \xrightarrow{u} s' \xrightarrow{v} s_i$. Additionally, since $St(s_0, q_0) \neq \Sigma$ and α is enabled in s_0, by axiom **SAFE** we know that α is a safe action, so by Definition 2 we get that s'_1 satisfies neither $\mathrm{Sink}(q_0)$ nor any progressing proposition of q_0. Since u does not contain any stubborn actions, α was a safe action, and s'_1 does not satisfy any progressing proposition, by axioms **R** and **SAFE** we know that no intermediate states along the run $s'_1 \xrightarrow{u} s'$ satisfy any progressing proposition or sink state proposition either. By repeating the argument, we conclude that $\langle s_0, q_0 \rangle \rightarrow_{St} \langle s'_1, q_0 \rangle \rightarrow^*_{St} \langle s', q_0 \rangle$ and we are able to extend the prefix of run π to be executable also in the reduced state space $\mathcal{T} \otimes_{St} \mathcal{A}$; now from $\langle s', q_0 \rangle$ we can repeat the arguments in this proof on the suffix of π.

- In case $q_0 \in F$, there are two situations. Either the NBA state changes again during the run π (and in this situation the previous case applies and we can extended the run as above) or it does not. Let us so assume that $\pi = \langle s_0, q_0\rangle\langle s_1, q_0\rangle\langle s_2, q_0\rangle \ldots$ that can be executed by the sequence of actions $\langle s_0, q_0\rangle \xrightarrow{\alpha_1} \langle s_1, q_0\rangle \xrightarrow{\alpha_2} \langle s_2, q_0\rangle \xrightarrow{\alpha_3} \cdots$ and in case there exists an index i such that $\alpha_i \in St(s_0, q_0)$, we can extend the run in the reduced state space as shown in the previous case. Let us assume so that $\alpha_i \notin St(s_0, q_0)$ for all i. By our assumption that $St(s_0, q_0) \neq \Sigma$ and by axiom **SAFE**, we know that all stubborn actions in $\langle s_0, q_0\rangle$ are safe and by axiom **KEY** we know that for an arbitrarily long prefix $\alpha_1\alpha_2 \ldots \alpha_n$ we can execute some stubborn key actions from s_n. Let α be a key action that can be executed after infinitely many such prefixes (there must be one as there are only finitely many actions). We claim that the run obtained by executing the actions $\alpha\alpha_1\alpha_2 \ldots$ is also an accepting run, implying that by repeating this process, we can obtain an infinite accepting run also in the reduced state space $\mathcal{T} \otimes_{St} \mathcal{A}$. In order to argue that the sequence of actions induces an accepting run, we need to argue that (i) the actions $\alpha_1\alpha_2 \ldots$ can be executed after first performing α and that (ii) the sequence $\alpha\alpha_1\alpha_2 \ldots$ can be executed without leaving the accepting Büchi state q_0. The claim (ii) follows from the earlier observation that all stubborn actions in $\langle s_0, q_0\rangle$ (including the key action α) are safe. For the sake of contradiction, assume that claim (i) does not hold. Let the $\alpha\alpha_1\alpha_2 \ldots \alpha_n$ be the longest sequence that can be executed, after which α_{n+1} cannot be executed. However, as $\alpha_1\alpha_2 \ldots$ is executable there must be an index $m > n$ such that from s_m the key action α is enabled (recall that α is enabled infinitely many times along this sequence). However, by axiom **COM** we can now commute the action α to the beginning of the sequence and conclude that the sequence $\alpha\alpha_1\alpha_2 \ldots \alpha_m$ is executable. This contradicts our assumption that $\alpha\alpha_1\alpha_2 \ldots \alpha_n$ was the longest such sequence (note that $n < m$). We have now showed that also in this case we are able to extend the run in the reduced state space.

We are hence in any situation able to extend the given run π to a run in the reduced state space, which implies to existence of an accepting run in $\mathcal{T} \otimes_{St} \mathcal{A}$. □

Lemma 1. *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. For all $t \in T'$ and all $t_1, \ldots, t_n \in T \setminus T'$*

a) if $M \xrightarrow{t_1 \ldots t_n t} M'$ then $M \xrightarrow{t t_1 \ldots t_n} M'$, and
b) if $M \xrightarrow{t_1 \ldots t_n} M'$ and $M \xrightarrow{t}$ then $M' \xrightarrow{t}$.

Proof. Let us first argue for the part a) of the claim. 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 the sake of contradiction that $M \overset{t}{\not\to}$. 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. This is 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. Again, this is a contradiction.

We can so conclude that $M \xrightarrow{t}$. Now let $M \xrightarrow{t} M_t$. We want to show that $M_t \xrightarrow{t_1 \ldots t_n} M'$. Assume for the 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 we get $p^{\bullet} \subseteq T'$, so since $t_i \notin T'$ we have $t_i \notin p^{\bullet}$, and in the latter case by Condition 1 we get $p^{\circ} \subseteq T'$, so since $t_i \notin T'$ we have $t_i \notin p^{\circ}$. In both cases t cannot disable t_i, so t_i can be fired, which is a contradiction. In conclusion, we argued that $M \xrightarrow{t} M_t$ and $M_t \xrightarrow{t_1 \ldots t_n} M'$.

For the part b) of the claim, we assume that $M \xrightarrow{t_1 \ldots t_n} M'$ and that t is enabled in M. Clearly, as $t_1, \ldots, t_n \in T \setminus T'$, the firing of any of the t_i transitions cannot disable the enabledness of t as all transitions that can possibly disable t are added to the set T' by Condition 1. □

Lemma 3. *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 any $t \notin A^{+}_{M}(b)$ and any $w \in (T \setminus \{t\})^{*}$, 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 on the proposition b. Let $t \notin A^{+}_{M}(b)$ and $w \in (T \setminus \{t\})^{*}$, and assume $M \not\models b$, $M \xrightarrow{w} M'$, $M' \not\models b$, and $M \xrightarrow{tw} M''$. By induction hypothesis we assume that for any subproposition b' of b, the fact $M' \not\models b'$ implies that $M'' \not\models b'$, 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 2 we know that $t \notin \mathrm{decr}(e_1)$ 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 $M'' \not\models b$.

$b = e_1 \leq e_2$ Analogous to $b = e_1 < e_2 + 1$.

$b = e_1 > e_2$ Analogous to $b = e_2 < e_1$.

$b = e_1 \geq e_2$ Analogous to $b = e_2 < e_1 + 1$.

$b = e_1 = e_2$ If $\mathrm{eval}_M(e_1) > \mathrm{eval}_M(e_2)$ then the argument proceeds as for $e_1 < e_2$. If $\mathrm{eval}_M(e_1) < \mathrm{eval}_M(e_2)$ then the argument proceeds as for $e_2 < e_1$.

$b = e_1 \neq e_2$ Since $\mathrm{incr}(e_1) \cup \mathrm{decr}(e_1) \cup \mathrm{incr}(e_2) \cup \mathrm{decr}(e_2) \subseteq A^{+}_{M}(b)$, if $M \xrightarrow{t} M_t$ then $\mathrm{eval}_{M_t}(e_1) = \mathrm{eval}_M(e_1)$ and $\mathrm{eval}_{M_t}(e_2) = \mathrm{eval}_M(e_2)$. Therefore $M' \not\models b$ implies that $M'' \not\models b$.

$b = b_1 \wedge b_2$ Since $M' \not\models b$ we have $M' \not\models b_1$ or $M' \not\models b_2$. By definition of A_M^+, the transition t is not strictly interesting wrt. either of b_1 or b_2. Hence, if $M' \not\models b_1$, by the inductive hypothesis $M'' \not\models b_1$, and if $M' \not\models b_2$ then $M'' \not\models b_2$. In both cases we get $M'' \not\models b_1 \wedge b_2$.

$b = b_1 \vee b_2$ Since $M' \not\models b$ then $M' \not\models b_1$ and $M' \not\models b_2$. By the induction hypothesis on b_1 and b_2, we get that $M'' \not\models b_1$ and $M'' \not\models b_2$, implying $M'' \not\models b_1 \vee b_2$.

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