Taking Some Burden Off an Explicit CTL Model Checker

Torsten Liebke and Karsten Wolf(✉)

Universität Rostock, Institut für Informatik, Rostock, Germany
{torsten.liebke,karsten.wolf}@uni-rostock.de

Abstract. In the CTL category of recent model checking contests, less problems have been solved than in the Reachability and LTL categories. Hence, improving CTL model checking technology deserves particular attention. We propose to relieve a generic explicit CTL model checker. This is done by designing specialised routines that cover a large set of simple (and frequently occurring) formula types. The CTL model checker is then only applied to formulas that do not fall into any special case. For the simple queries, we may apply simple depth-first search instead of recursive search, we may use much more powerful dialects of the stubborn set reduction, and we may add additional tools for verification, such as the state equation. Our approach covers about half of the CTL category of a recent model checking contest and significantly increases the power of CTL model checking.

Keywords: CTL model checking · Partial order reduction

1 Introduction

In recent years, Computational Tree Logic (CTL, [2]) has been the category where most queries were left unsolved in the yearly Petri net model checking contest (MCC, [9]). Consequently, CTL model checking deserves particular attention with the aim of keeping pace with LTL and reachability checking. At present, leading Petri net CTL model checkers such as TAPAAL [7] and LoLA [29] use explicit model checking algorithms. Their main tool for alleviating state explosion is the stubborn set method [22] or, more general, the class of partial order reduction methods [12,18]. In essence, partial order reduction explores, in any given marking, only a subset of the enabled transitions. CTL preserving partial order reduction [11] has severe restrictions: we either find, in a given marking, a *singleton* set consisting of an invisible transition that satisfies all other conditions for a stubborn set, or we have to fire all transitions enabled in this marking. This condition is necessary for CTL preservation since otherwise the position of visible transitions with respect to branching points may not be preserved which in turn would jeopardise preservation of the branching time logic CTL.

Many CTL queries have a rather simple structure in the sense that they contain only few temporal operators. In the MCC, this might be an artifact of the

S. Donatelli and S. Haar (Eds.): PETRI NETS 2019, LNCS 11522, pp. 321–341, 2019.
https://doi.org/10.1007/978-3-030-21571-2_18

formula generation mechanism. However, we share the same experience with the users of our tool LoLA. Even if complicated CTL formulas occasionally occur, they are subject to several simplification approaches. Firstly, there exist many tautologies in temporal logic. Not all of them are commonly known. This way, an originally complicated formula may automatically be rewritten to a much simpler query [1]. The formula rewriting system of LoLA currently contains more than 100 rewrite rules that are based on CTL* tautologies. For Petri nets, secondly, linear programming techniques employing the Petri net state equation can be applied to the atomic propositions in the formula [1], sometimes proving them to be invariantly true or false. This way, whole subformulas of a query may collapse, enabling further rewriting based on tautology. Boolean combinations of queries can be simplified by checking the subformulas separately (thus having queries with less visible transitions in each run which propels partial order reduction). Thirdly, complicated queries may be replaced by simpler queries through modifications in the system under investigation. A simple example is the verification of relaxed soundness [8] for workflow nets. For every transition t, we have to show that there is a path to a given final place f that includes the occurrence of t. In CTL, this reads as $\mathrm{EF}(t \text{ occurs} \wedge \mathrm{EF}(f \geq 0))$. Inserting a fresh post-place p to t, the query can be simplified to $\mathrm{EF}\ (p \geq 0 \wedge f \geq 0)$. The most systematic approach of this kind is LTL model checking as a whole. The explicit verification of an LTL formula ϕ is done by modifying the system under investigation (we refer to the construction of the product system with the Büchi automaton for $\neg\phi$, [26]). In the modified system, we only need to verify $\neg$ GF *accepting-state* instead of the arbitrarily complicated ϕ.

We conclude that explicit CTL model checking can be substantially improved through a special treatment of as many as possible of the most simple queries. Special treatment means that we apply a specific verification procedure to such queries thus avoiding the application of the generic CTL model checking routines. This approach has two obvious advantages. Firstly, some of the queries may permit the use of completely different verification technology. For example, for properties like $EF\ \phi$ or $AG\ \phi$ (with ϕ assumed not to contain additional temporal operators), we may employ the Petri net state equation for verification [28]. Secondly, a verification technique dedicated to just one class C of simple CTL queries may use a better partial order reduction: we only need to preserve C rather than whole CTL.

In this paper, we focus on the second item. We identify several classes of simple CTL queries for which specific search routines enable the use of partial order reduction methods better than CTL preserving ones. These partial order reduction methods are already known in most cases. So the actual contribution of this paper is to show that the systematic separation of simple queries from general CTL routines can indeed improve CTL model checking. In 2018, almost 70% of the CTL queries in the MCC were transferred to specific routines for simple queries in our tool LoLA. Employing these methods, LoLA could solve more than 50% of the queries that could not be solved with the generic CTL model checking algorithm.

We start with a brief introduction of the terminology of Petri nets and the temporal logic CTL. We then provide the necessary facts on the stubborn set method. In the main part of the paper, we discuss our list of simple CTL queries. We conclude with experimental results.

2 Terminology

Definition 1 (Place/transition net). *A place/transition net consists of a finite set P of places, a finite and disjoint set T of transitions, a set $F \subseteq (P \times T) \cup (T \times P)$ of arcs, a weight function $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ where $[x, y] \notin F$ if and only if $W(x, y) = 0$, and a marking m_0, the initial marking. A marking is a mapping $m : P \to \mathbb{N}$.*

Definition 2 (Behaviour of a place/transition net). *Transition t is enabled in marking m if, for all $p \in P$, $W(p, t) \leq m(p)$. If t is enabled in m, t can fire, producing a new marking m' where, for all $p \in P$, $m'(p) = m(p) - W(p, t) + W(t, p)$. This firing relation is denoted as $m \xrightarrow{t} m'$. It can be extended to firing sequences by the following inductive scheme: $m \xrightarrow{\varepsilon} m$ (for the empty sequence ε), and $m \xrightarrow{w} m' \wedge m' \xrightarrow{t} m'' \implies m \xrightarrow{wt} m''$ (for a sequence w and a transition t). The reachability graph of a place/transition net N has a set of vertices that comprises of all markings that are reachable by any sequence from the initial marking of m. Every element $m \xrightarrow{t} m'$ of the firing relation ($t \in T$) defines an edge from m to m' annotated with t.*

With a matrix C where, for all $p \in P$ and $t \in T$, $C(p, t) = W(t, p) - W(p, t)$, and marking m, equation $m_0 + Cx = m$ is called the *Petri net state equation*. It has a nonnegative integer solution for every m reachable from m_0: fix a firing sequence w from m_0 to m and let, for every t, $x[t]$ be the number of occurrences of t in w. For unreachable m, the state equation may or not have nonnegative integer solutions.

In the sequel, we consider only Petri nets with finite reachability graph (i.e. bounded Petri nets).

Definition 3 (Syntax of CTL). `TRUE`, `FALSE`, `FIREABLE` (t) *(for $t \in T$),* `DEADLOCK`, *and $k_1 p_1 + \cdots + k_n p_n \leq k$ ($k_i, k \in \mathbb{Z}, p_i \in P$) are atomic propositions. $PQ = \{A, E\}$ is called the set of path quantifiers, $UT = \{X, F, G\}$ the set of unary temporal operators, and $BT = \{U, R\}$ the set of binary temporal operators.*

Every atomic proposition is a CTL formula. If ϕ and ψ are CTL formulas, so are $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $QY\phi$ (with $Q \in QP$ and $Y \in UT$), and $Q(\phi B \psi)$ (with $Q \in QP$ and $B \in BT$).

The logic LTL is defined similarly. The only difference is that the path quantifiers are not used in LTL.

Definition 4 (Semantics of CTL). *Marking m satisfies CTL formula ϕ ($m \models \phi$) according to the following inductive scheme:*

- $m \models$ `TRUE`, $m \not\models$ `FALSE`;
- $m \models$ `FIREABLE (`t`)` *if t is enabled in m;*
- $m \models$ `DEADLOCK` *if there is no enabled transition in m;*
- $m \models k_1 p_1 + \cdots + k_n p_n \leq k$ *if* $k_1 m(p_1) + \cdots + k_n m(p_n) \leq k$;
- $m \models \neg\phi$ *if* $m \not\models \phi$;
- $m \models (\phi \wedge \psi)$ *if* $m \models \phi$ *and* $m \models \psi$;
- $m \models EX\phi$ *if there is a t and an m' with* $m \xrightarrow{t} m'$ *and* $m' \models \phi$;
- $m \models E(\phi U \psi)$ *if there is a path* $m_1 m_2 \ldots m_k$ ($m_1 = m, k \geq 1$) *in the reachability graph where* $m_k \models \psi$ *and, for all i with* $1 \leq i < k$, $m_i \models \phi$;
- $m \models A(\phi U \psi)$ *if, for all maximal paths (i.e. infinite or ending in a deadlock)* $m_1 m_2 \ldots$ *in the reachability graph with* $m_1 = m$, *there is a k ($k \geq 1$) where* $m_k \models \psi$ *and, for all i with* $1 \leq i < k$, $m_i \models \phi$.

The semantics of the remaining CTL operators is defined using the tautologies $(\phi \vee \psi) \iff \neg(\neg\phi \wedge \neg\psi)$, $AX\phi \iff \neg EX\neg\phi$, $EF\phi \iff E($`TRUE` $U\phi)$, $AF\phi \iff A($`TRUE` $U\phi)$, $AG\phi \iff \neg EF\neg\phi$, $EG\phi \iff \neg AF\neg\phi$, $E(\phi R\psi) \iff \neg A(\neg\phi U\neg\psi)$, *and* $A(\phi R\psi) \iff \neg E(\neg\phi U\neg\psi)$.

A place/transition net N satisfies a CTL formula if its initial marking m_0 does.

For LTL, all temporal modalities concern the same single path. Moreover, only infinite paths are considered. A maximal finite path is transformed into an infinite path by infinitely repeating the last (deadlock) marking. Otherwise, evaluation accords with CTL. A place/transition net N satisfies an LTL formula if all paths starting from m_0 do.

3 CTL Model Checking

We consider *local* model checking, that is, we want to evaluate a given CTL formula just for the initial marking m_0. Other markings are only considered as far as necessary for determining the value at m_0. In global model checking, one would be interested in the value of the given formula in all reachable markings. As a reference for our work, we use the algorithm of [27]. In the sequel, we briefly sketch this algorithm.

We assume that, attached to every marking, there is a vector that has an entry for every subformula of the given CTL query. The value of a single entry can be true, false, or unknown. Whenever we want to access the value of a subformula ϕ in a marking m, we inspect the corresponding value. If it is unknown, we recursively launch a procedure to evaluate ϕ in m.

If ϕ is an atomic proposition or a Boolean combination of subformulas, evaluation is trivial. For evaluating a formula of shape $EX\ \phi'$ or $AX\ \phi'$, we proceed to the immediate successor states and evaluate ϕ' in those states. If the successor

marking has not been visited yet, we add it to the set of visited markings and initialise its vector of values.

If ϕ has the shape $A(\psi U \chi)$, we launch a depth-first search from m, aiming at the detection of a counterexample. The search proceeds through markings that satisfy ψ, violate χ, and for which $A(\psi U \chi)$ is recorded as unknown. Whenever we leave the space of state satisfying these assumption, there is a reaction that does not require continuation of the search beyond that marking, as follows.

If χ is satisfied, or $A(\psi U \chi)$ is recorded as true in any marking m', we backtrack since there cannot be a counterexample path containing m'. If χ and ψ are violated, or $A(\psi U \chi)$ is recorded as false, we exit the search since the search stack forms a counterexample for $A(\psi U \chi)$ in m. If we hit a marking m' on the search stack, we have found a counterexample, too (a path where ψ and not χ hold forever). The depth-first search assigns a value different from unknown to all states visited during the search: For markings on the search stack (i.e. participating in the counterexample), $A(\psi U \chi)$ is false, while for states that have been visited but already removed from the search stack, $A(\psi U \chi)$ is actually true.

If ϕ has the shape $E(\psi U \chi)$, we launch a similar depth-first search, aiming at the detection of a witness path. This time, we integrate Tarjan's algorithm [21] for detecting the strongly connected components (SCC) during the search. It proceeds through markings that satisfy ψ, violate χ, and for which $E(\psi U \chi)$ is recorded as unknown. If we hit a marking m' where χ is satisfied, or $E(\psi U \chi)$ is true, we have found our witness. In states where ψ and χ are violated, or $E(\psi U \chi)$ is known to be false, we backtrack since there cannot be witness path containing such a marking. Again, we assign a value different from unknown to every marking visited during the search. Markings that are on the search stack as well as markings that are not on the search stack but appear in SCC that have not yet been completely explored, get value true. An SCC is not yet fully explored if it contains elements that are still on the search stack. Then, however, a path to the search stack extended by the remaining portion of the search stack forms a witness. For markings in SCC that have been completely explored, $E(\psi U \chi)$ is false.

We can see that existential and universal until operators are not fully symmetric. This is due to the fact that a cycle of markings that satisfy ψ and violate χ, form a counterexample for universal until but no witness for existential until. Any SCC with more than one member would contain such a cycle. Consequently, universal until will never remove markings from the search stack without closing a whole (singleton) SCC.

The remaining CTL operators can be traced back to the two until operators using tautologies. Since every search assigns values to all visited markings, the overall run time of the algorithm is $O(|\phi||R|)$ where $|\phi|$ is the length of ϕ (the number of subformulas), and $|R|$ is the number of markings reachable from m_0.

4 Partial Order Reduction—the Stubborn Set Method

Given a Petri net N and a property ϕ, the stubborn set method aims at producing a subgraph G' of the reachability graph G of N such that the evaluation of

ϕ using G' yields the same value as the evaluation on G. To this end, a set stubborn(m) of transitions is assigned to every marking m, and only enabled transitions in stubborn(m) are explored for the construction of G'.

Over the years, a consistent systematic has emerged for presenting stubborn set methods. There is a list of *principles* that should govern the selection of stubborn sets. Each principle comes with an *algorithmic approach* for computing a stubborn set that obeys that principle. Finally, there is a list of *results* stating that, if G' is computed using stubborn sets that meet some selection of principles, all properties of a certain class of properties are preserved. In the sequel, we shall list principles and results that we need for our considerations below. For some principles, there exist several variations that push results further to the limit. However, our focus here is not stubborn set theory as such but CTL model checking technology. For this reason, we selected principles such that presentation is as understandable as possible. For stronger results on stubborn sets, the reader is referred to [11,15,23,25]. We will further completely skip the algorithmic approaches as they are not necessary for understanding our argument.

In the sequel, let $N = [P, T, F, W, m_0]$ be an arbitrary fixed place/transition net.

Definition 5 (COM: The commutativity principle). stubborn(m) $\subseteq T$ *satisfies the commutativity principle (COM for short) if, for all* $w \in (T \setminus \text{stubborn}(m))^*$ *and all* $t \in \text{stubborn}(m)$, $m \xrightarrow{wt} m'$ *implies* $m \xrightarrow{tw} m'$.

Definition 6 (KEY: The key transition principle). stubborn(m) $\subseteq T$ *satisfies the key transition principle (KEY for short) if* m *does not enable any transition or it contains a transition* t^* *(a* key transition*) such that, for all* $w \in (T \setminus \text{stubborn}(m))^*$, $m \xrightarrow{w} m'$ *implies that* t^* *is enabled in* m'.

Definition 7 (VIS: The visibility principle). *Transition* t *is invisible w.r.t. an LTL or CTL formula* ϕ *if, for all atomic propositions* ψ *occurring in* ϕ *and all markings* m, m', $m \xrightarrow{t} m'$ *implies that* $\psi(m)$ *holds if and only if* $\psi(m')$ *holds.* stubborn(m) $\subseteq T$ *satisfies the visibility principle for a property* ϕ *(VIS(*ϕ*) for short) if* stubborn(m) *contains only invisible transitions w.r.t.* ϕ*, or all transitions.*

Definition 8 (IGN: The non-ignoring principle). stubborn *satisfies the non-ignoring principle (IGN for short) if every cycle in the reduced reachability graph contains a marking where all enabled transitions are explored.*

Definition 9 (UPS: The up-set principle). *For a marking* m *and a CTL property* ϕ *such that* $m \not\models \phi$, U *is an up-set if every path from* m *to a marking that satisfies* ϕ *contains an element of* U. stubborn(m) *satisfies the up-set principle w.r.t.* ϕ *if* $m \models \phi$ *or* $U \subseteq \text{stubborn}(m)$, *for some up-set* U.

Definition 10 (BRA: The branching principle). stubborn(m) *satisfies the branching principle (BRA for short) if* stubborn(m) *contains a single enabled transition, or all enabled transitions.*

In the following propositions, let G' be a reduced reachability graph using stubborn sets that meet the principles mentioned in the assumption.

Each principle has a specific purpose for proving property preservation. In most cases, we assume that there is a path π in the full reachability graph (e.g. a witness or counterexample for the property under investigation) and show that the reduced system contains a path π that is equally fit w.r.t. the studied property. With COM, π' may execute transitions in another order than π. With KEY (in connection with COM), π' may contain transitions that are not occurring in π. With UPS, the stubborn set at m will always contain a transition of π. With VIS, visible transitions in π' appear in the same order as in π, if they appear in π'. IGN is used for making sure that all transitions of π are eventually occurring in π', and BRA is used for making sure that visible transitions are not swapped with branches in the state space other than branches that are introduced by concurrency. Again, [11,15,23,25] provide more details concerning these issues.

Proposition 1 (Preservation of deadlocks, [22]). *If the principles COM and KEY are satisfied then G' contains all deadlocks and at least one infinite path of the original reachability graph.*

Proposition 2 (Preservation of terminal SCC, [24]). *If the principles COM, KEY, and IGN are satisfied then G' contains at least one marking of every terminal SCC of the original reachability graph.*

Proposition 3 (Preservation of reachability, [15,19]). *Let ϕ be a CTL formula without temporal operators. If the principles COM and UPS(ϕ) are satisfied then $EF\phi$ is preserved.*

Proposition 4 (Preservation of LTL-X, [18,23]). *Let ϕ be an LTL property not using the X operator. If the principles COM, KEY, VIS(ϕ), and IGN are satisfied than ϕ is preserved.*

Proposition 5 (Preservation of CTL-X, [11]). *Let ϕ be a CTL formula not using the X operator. If the principles COM, KEY, VIS(ϕ), IGN, and BRA are satisfied then ϕ is preserved.*

5 Simple CTL Queries

We are now ready to discuss the advantages of separating simple CTL queries. In most cases, one of the advantages shall be the ability of using a more powerful stubborn set method. In all reported cases, we will be able to drop the very limiting BRA principle that enables reduction only in markings where just one (invisible) enabled transition is sufficient to meet all the other principles. In addition, less restrictive conditions (i.e. a smaller set of principles to be met), leads to potentially smaller stubborn sets and thus to better reduction.

The simple problems discussed below appear as pairs of an existentially and a universally quantified formula. These two formulas can be reduced to each

other by negation. Hence, they permit the application of the same verification techniques.

In the sequel, let ϕ and ψ be CTL formulas without temporal operators. Experimental data refers to the tool LoLA 2 [29], applied to the benchmark of the model checking contest (MCC) 2018. We give 300 seconds for every individual query. More details on experiments can be found in Sect. 7.

5.1 AG ϕ, EF ϕ

For the reachability problem $EF\phi$, we may use stubborn set as suggested by Proposition 3 [19], or a relaxed version [15]. Both techniques have specific advantages. The first method works much better if $EF\phi$ is true while the second method has advantages if $EF\phi$ is false. Any of the methods, however, is much more powerful than the CTL-X preserving method.

For reachability, Petri net structure theory can be applied. If the Commoner's theorem [4,13] applies, EF DEADLOCK evaluates to false. The conditions of the theorem can be checked as a satisfiability problem in propositional logic (SAT) [17]. The Petri net state equation, enhanced with the refinement method proposed in [28] provides a powerful tool for verifying other reachability queries. Since the structural methods can be traced back to NP-complete problems (SAT resp. Integer Linear Programming) and therefore use only polynomial space, they can be applied in parallel to state space exploration.

The ability of LoLA to solve far beyond 90% of the queries in the reachability category of the MCC, compared to less than 70% if only a CTL model checker is applied to the CTL category, clearly confirms the conclusion to separate reachability queries from CTL model checking.

5.2 AF ϕ, EG ϕ

The CTL formula $AF\phi$ is equivalent to the LTL formula $F\phi$. The universal path quantifier is implicitly present in LTL, too, since a system satisfies an LTL formula if *all* its paths do. That is, we may apply LTL-X preserving stubborn sets instead of CTL-X preserving ones. Without the BRA principle, LTL-X preserving stubborn sets are more powerful (more than 90% success in the LTL category, compared to less than 70% success if CTL-X preserving stubborn sets are applied to all of the CTL category).

Additionally, we may completely drop the IGN principle for visible transitions. We sketch a proof for $EG\phi$. If there is no witness path (an infinite path where ϕ permanently holds) in the original reachability graph, there cannot be one in the reduced reachability graph which is a subgraph. If there is an (infinite) witness path π, then by COM, KEY, and VIS, there is an infinite path π' in the reduced system such that visible transitions of π' occur in the same order as in π. Invisible transitions in π' do not alter the value of ϕ. That is, π' witnesses $EG\phi$ as well since otherwise there would be a prefix of π where ϕ is violated, contradicting the assumption that π is a witness path.

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

5.3 E (ϕ U ψ), A(ϕ R ψ)

To satisfy E (ϕ U ψ), we need to use stubborn sets that preserve two properties: first, the reachability of ψ, and second, the non-violation of ϕ. Combining the discussion for reachability (EF) and non-violation (EG), we propose the following combination of principles for the stubborn sets to be used: COM, UPS(ψ), and VIS(ϕ). We sketch the arguments for correctness of this setting. Assume the original reachability graph contains a witness path π. By UPS(ψ), this path contains a transition that is in the stubborn set used in the initial marking. By COM, we can shift the first such transition to the front of the path. By VIS(ϕ), this modification does not change the order of transitions visible for ϕ. At least the first transition of the modified path can be replayed in the reduced reachability graph. By induction, a witness path in the reduced system is established.

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

We can employ linear programming for checking a necessary and a sufficient condition for E(ϕ U ψ). A necessary criterion is obviously EF ψ, and the approach in [28] can be used for checking this condition. A sufficient condition is the reachability of ψ using only transitions that are invisible to ϕ, in addition to checking ϕ in the initial marking. This can be checked by removing all transitions visible for ϕ from N and applying the approach of [28] to the resulting net. With the moderate memory footprint of linear programming, the necessary and sufficient conditions can be checked in parallel to the actual depth-first search ("portfolio approach").

5.4 EGEF ϕ, AFAG ϕ

For this pair of formulas, we do not have a dedicated version of stubborn sets, so we apply CTL preserving stubborn sets for state space reduction. However, the check for the pair of temporal operators can be folded into a single depth-first search. We present the approach for EGEF ϕ. The witness path π for the EG operator is a maximal path (i.e. infinite or ending in a deadlock).

If the path ends in a deadlock, the deadlock marking has to satisfy ϕ since this is the only way for ϕ to be reachable from that marking. If the deadlock satisfies ϕ, all markings on the path satisfy EF ϕ automatically, so this case can be easily implemented. An infinite path appears in a model checker as a cycle that is reachable from m_0. For satisfying EGEF ϕ, it is necessary and sufficient that, from one of the markings m on the cycle, a marking m' is reachable that

satisfies ϕ. Necessity follows immediately from the definition of the semantics of CTL. Sufficiency follows from the fact that m is reachable from all markings in π, so m' is reachable as well from all markings in π.

We record, for every marking visited in depth-first search, whether a marking satisfying ϕ can be reached. To this end, every marking that satisfies ϕ itself is marked as "can reach ϕ". In addition, whenever depth-first search backtracks from a marking that can reach ϕ, the predecessor marking is marked as well as "can reach ϕ". For detecting cycles, we use the well-known fact [14] that every cycle in a state space contains an edge from some marking m to a marking m' such that, at some stage of depth-first search, m is the top element of the search stack and m' is on the search stack as well (such an edge is called *backward edge*). During the search, we maintain information whether or not the search stack contains such m'. If this is the case while the marking on top of the stack can reach ϕ, we return *true*. If we reach a deadlock satisfying ϕ, we return *true* as well. If the search is completed without having returned true, we return *false*.

Lemma 1. *The procedure sketched above correctly evaluates EGEF ϕ.*

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

In addition to the combined depth-first search, we can add a check for EF ϕ as a necessary condition and AG ϕ as a sufficient criterion to a portfolio for EGEF ϕ. Again, the state equation approach can be used in order not to take too much memory away from the main search procedure.

5.5 EFEG ϕ, AGAF ϕ

We present the approach for EFEG ϕ. We check the property by nested depth-first search. The approach uses ideas from [5,6,10,14] that are concerned with the similar problem of finding accepting cycles in Büchi automata. Outer search

proceeds through markings that have already proven not to be part of a ϕ-cycle (or a ϕ-deadlock). This includes markings that do not satisfy ϕ and markings where inner search has already been run. Inner search proceeds only through ϕ-markings and tries to find a cycle or a deadlock. By definition, EFEG ϕ holds if and only if a ϕ-cycle or a ϕ-deadlock is reachable from m_0. We start with outer search. Whenever we encounter a fresh ϕ-marking m, we switch to inner search. If inner search terminates without having found a cycle or deadlock, we resume outer search in m.

This procedure is very similar to the general CTL model checking algorithm. However, we may apply dedicated stubborn sets. In outer search, we distinguish markings that satisfy ϕ from markings that do not satisfy ϕ. If m does not satisfy ϕ, we use stubborn sets that satisfy COM and UPS(ϕ). If m satisfies ϕ, we have two correct combinations of principles. We can use stubborn sets that satisfy COM and UPS($\neg\phi$), or stubborn sets that satisfy COM, KEY, and VIS(ϕ). In inner search, we use stubborn sets satisfying COM, KEY, and VIS(ϕ).

Lemma 2. *A reduced reachability graph obeying the principles stated above preserves EFEG ϕ.*

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

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

witnesses EG ϕ and EG ϕ is preserved by stubborn sets with COM, KEY, and VIS (see Subsect. 5.2).

Second, consider the case where m_1 does not satisfy ϕ. Since $m_k = m_i^*$ satisfies ϕ, the path from m_1 to m_k contains a transition of the up-set used in m_1, and, by the UPS principle, elements of the stubborn set used in m_1. Applying COM, we obtain an alternative path where the first transition is in the stubborn set used in m_1. Its successor meets the same properties in m_1 but with a smaller value for k.

It remains to consider the case where m_1 satisfies ϕ and the first case is not applicable. Then, for at least one q ($2 \leq q \leq k$), m_q violates ϕ. If we apply stubborn sets satisfying COM and UPS($\neg\phi$), we argue as in the second case. This yields a continuation for the witness path in the reduced reachability graph. If we obey COM, KEY, and VIS instead, we argue as follows. If a transition of the stubborn set used in m_1 occurs in π, COM yields a continuation of the path in the reduced reachability graph. Otherwise, the stubborn set in m_1 contains only invisible transitions (by VIS). Choose a key transition t^* in the stubborn set for m_1 (available via KEY). By KEY, t^* is never disabled in π. By COM, all transitions in π can still be executed after having fired t^*. The t^*-successor m' of m_1 occurs in the reduced reachability graph. The third case is applicable only a finite number of times since m' satisfies ϕ but there is no ϕ-cycle reachable in inner search from m_1. □

As in previous cases, AG ϕ is a sufficient condition for EFEG ϕ while EFϕ is necessary. The state equation approaches to these properties may be added to the portfolio for EFEG ϕ.

5.6 AGEF ϕ, EFAG ϕ, EFAGEF ϕ, AGEFAG ϕ

These properties are tightly related to terminal SCC of the reachability graph. For AGEF ϕ, every terminal SCC must contain a marking satisfying ϕ. For EFAG ϕ, there must exist a terminal SCC where all markings satisfy ϕ. For EFAGEF ϕ, a terminal SCC must exist where at least one marking satisfies ϕ, and for AGEFAG ϕ, all markings in all terminal SCC must satisfy ϕ.

By Proposition 2, stubborn sets obeying COM, KEY, and IGN preserve access to all terminal SCC of the reachability graph. Adding UPS(ϕ) for AGEF ϕ and EFAGEF ϕ (or UPS($\neg\phi$) for the other two cases) at least inside the terminal SCC preserves the properties under investigation. There are several strategies for implementing UPS in the terminal SCC. We can either require it for all markings (then KEY may be dropped) [20], or enforce a relaxed version of UPS in all markings (see [15] for details), or we may launch a depth first search using stubborn sets with COM and UPS whenever we encounter a terminal SCC in the reduced graph w.r.t. COM, KEY, and IGN.

The proposed procedure has two advantages. First, we proceed in a single depth-first search compared to the recursive approach of a CTL model checker. Second, we can drop the very problematic BRA principle. Being able to drop the VIS principle as well, the stubborn set method can achieve substantial reduction even in cases where ϕ is a property that refers to a large number of places, and causes many transitions to be visible.

For all properties considered in this subsection, AG ϕ is a sufficient condition and EF ϕ is necessary. Using the state equation approach mentioned above, we can add these checks to our portfolio. This way, we have an additional opportunity to answer the query early while using only a moderate amount of additional memory.

5.7 Formulas Starting with EX and AX

This section is concerned with formulas of the shape EXEF ϕ, EXEG ϕ, EXE(ϕ R ψ), EXE(ϕ U ψ), EXEGEF ϕ, EXEFEG ϕ, AXAG ϕ, AXAF ϕ, AXA(ϕ R ψ), AXA(ϕ U ψ), AXAGAF ϕ and AXAFAG ϕ. We explicitly discuss the existentially quantified ones. Verification of these properties can be traced back to the respective formula without the leading EX operator. All we need to do is to explore all enabled transitions of m_0, and not to store m_0. That is, whenever m_0 is visited during the search, it is treated as fresh marking and a stubborn set can be used. Other than this, the same stubborn set approaches as discussed earlier are applicable.

5.8 Single-Path Formulas

In this section, we discuss a larger class of CTL formulas. We aim at applying LTL model checking instead of CTL model checking. This way, the BRA principle may be skipped. Switching to an LTL model checker is actually a good idea, given the better success rate of tools like LoLA in the LTL category of the MCC. According to [3], removing the path quantifiers of a CTL formula yields the only candidate to be an equivalent LTL formula. But this candidate may or may not turn out to be indeed equivalent. The ACTL formulas where equivalence can be achieved can be characterised [16]. We chose to apply the approach to a collection of CTL formulas that can be more easily be recognised by a rewriting system.

LTL is a linear time temporal logic. That is, a counterexample for an LTL formula is always a single maximal path of the system. In contrast, CTL is a branching time temporal logic. This means that the counterexample is a subtree of the computation tree (the unrolling of the reachability graph). For instance, a witness for EGEF ϕ consists of a maximal path where, for each marking a finite path to a state satisfying ϕ branches off. Even with the observations made in Subsect. 5.4, the structure remains more complicated than a single path. However, in several cases, the branching structure collapses into a single path. Consider EFEG ϕ. Here, we only need a finite path to the first state of a ϕ-cycle (or deadlock), extended with the cycle itself. It is precisely a counterexample for the LTL formula GF $\neg\phi$ that is obtained by negating EFEG ϕ to AGAF $\neg\phi$ and then dropping the universal path quantifiers. In the sequel, we shall exhibit a class of CTL formulas where this approach is applicable. We call them *single-path* formulas. They may contain only existential path quantifiers or only universal path quantifiers. In the next definition, let a *state predicate* be a CTL formula without any temporal operator.

Definition 11 (Existential single-path formula). *If ϕ and ψ are existential single-path formulas and ω is a state predicate, then the following formulas are existential single-path formulas:*

- *ω (the base of the inductive definition);*
- *$EG\ \omega$;*
- *$EF\ \phi$;*
- *$E(\omega\ U\ \phi)$;*
- *$E(\phi\ R\ \omega)$;*
- *$\phi \vee \psi$;*
- *$\phi \wedge \omega$;*

Universal single-path formulas are defined accordingly:

Definition 12 (Universal single-path formula). *If ϕ and ψ are universal single-path formulas and ω is a state predicate, then the following formulas are universal single-path formulas:*

- *ω (the base of the inductive definition);*
- *$AF\ \omega$;*
- *$AG\ \phi$;*

- $A(\omega\ R\ \phi)$;
- $A(\phi\ U\ \omega)$;
- $\phi \wedge \psi$;
- $\phi \vee \omega$;

The class of single-path formulas covers several cases discussed earlier in this paper. However, the results above are stronger then the results we shall obtain now, so the separate treatment is indeed justified. It is easy to see that the negation of an existential single-path formula is indeed a universal single-path formula and vice versa. That is, we may restrict subsequent considerations to universal single-path formulas.

For a universal single-path formula ϕ, let LTL(ϕ) be the formula obtained from ϕ by removing all path quantifiers. We claim:

Lemma 3. *Let ϕ be a universal single-path formula and N a Petri net. Then N satisfies ϕ if and only if N satisfies LTL(ϕ).*

Proof. We show that violation of ϕ implies violation of LTL(ϕ) and violation of LTL(ϕ) implies violation of ϕ. We proceed by induction, according to Definition 12.

Case ω (state predicate): In both CTL and LTL, a state predicate is violated if it does not hold in the initial marking.

Case AF ω: In both CTL and LTL, a counterexample is a maximal path where all markings violate ω. Since ω is a state predicate, it directly refers to the markings on the path.

Case $A(\omega\ R\ \phi)$: A counterexample for A(ω R ϕ) is a finite path to a marking where all but the last marking violate ω and the last marking violates ϕ. As ω is a state predicate, the intermediate markings as such violate ω. Hence, the path, extended by a counterexample path for ϕ at the final marking (which exists by induction hypothesis) yields a path that is a counterexample for LTL(A(ω R ϕ)). For the other direction, consider a counterexample for LTL(A(ω R ϕ)). It must have a suffix serving as a counterexample for LTL(ϕ). Hence the first marking of that path violates ϕ (using once more the induction hypothesis). The markings that are not part of the considered suffix violate ω, so the full path is a counterexample for A(ω R ϕ).

Case $A(\phi\ U\ \omega)$: A counterexample can either be a maximal path where ω is violated in every marking (then apply the argument of Case AF ω) or a path where ω is violated until both ω and ϕ are violated (then apply the argument of case A(ω R ϕ)).

Case AG ϕ: This case can be traced back to Case A(ω R ϕ) using the tautology AG $\phi \iff$ A(false R ϕ).

Case $\phi \wedge \psi$: If ϕ is violated, there is a counterexample for ϕ for which the induction hypothesis may be applied. Otherwise, there is a counterexample for ψ for which again the induction hypothesis applies.

Case $\phi \vee \omega$: In this case, ϕ and ω are violated. Since ω is a state predicate, only the initial marking of the path is concerned. Hence, the induction hypothesis applied to ϕ yields the desired result. □

Using Lemma 3 the considered fragment of CTL can be verified using an LTL model checker. As another option, we may use a CTL model checker but apply LTL preserving stubborn sets. Existential single-path formulas can be verified by checking their negation.

5.9 Boolean Combinations

If a CTL formula is a Boolean combination of subformulas, we may check the subformulas individually. Doing that, the subformulas often have a smaller set of visible transitions, so some of the stubborn set principles are stronger for a subformula than for the whole formula. Some subformulas may contain the X operator, so the stubborn set method can be applied at least to the subformulas not containing the X operator. Some subformulas may fall into any of the classes considered above, so their verification may be accelerated.

In a setting with distributed memory, the subformulas can be verified in parallel. With shared memory, a parallel execution is not necessarily recommendable since the individual verification procedures compete for memory which may lead to memory exhaustion in all procedures while verification could have been successful if the whole memory were available for either of the procedures.

To get the most out of our accelerated procedures in a shared memory setting, we rate subformulas according to their simplicity. Then, the simplest formulas are checked first. This way, we get an increased probability that the result of the Boolean combination can already be determined (by a true subformula of a disjunction or a false subformula of a conjunction) before the procedures for the most complicated formulas have been launched.

Our rating works as follows. The simplest category consists of subformulas that do not contain temporal operators. They are true, false, or can be evaluated by just inspecting the initial marking. Second category consists of formulas that contain only X operators. They can be verified by exploring the state space to a very limited depth. Then follow categories for the simple cases studied above. The simplicity of these categories is mainly influenced by our experience concerning their performance in the MCC. Then follow the categories LTL-X, CTL-X, LTL, and CTL (in this order). For the last categories, applicability of stubborn sets is the distinguishing feature.

6 Preprocessing

It has already been recognised [1] that formulas should be carefully preprocessed before running a model checking procedure. Atomic propositions may turn out to be always true or always false, proven by the infeasibility of a linear program that can be derived from the proposition and the Petri net state equation. Once some of the propositions have been identified as true or false, whole subformulas may turn out to be true or false as well. This way, a significant number of formulas can be evaluated without running a model checker at all. For other formulas, the remaining model checking problem is simpler than the original one.

In the remainder of this section, we add a few observations to the findings of [1]. We have two objectives. First, we want to increase the number of situations where one of the special routines discussed in the previous section can be applied. Second, we want to increase the power of the stubborn set method.

Boolean Operators. Some tautologies, such as AG $(\phi \wedge \psi) \iff (\text{AG } \phi \wedge \text{AG } \psi)$ can be applied in both directions. Applying it from right to left decreases the number of temporal operators. However, the operator in general applies to a more complicated subformula, with more transitions being visible. Applying the formula from left to right leads to a formula with more temporal operators that, however, work on a smaller subformula. Stubborn sets potentially work better. Moreover, we increase the likelihood that the Boolean operator then becomes the root of the formula tree and the results of Sect. 5.9 are applicable. Hence, our tool LoLA uses an orientation of tautologies that prefers pushing Boolean operators towards the root of the formula tree.

X Operators. We also try to push X operators towards the root to the formula tree. To this end, we apply tautologies such as EFEX $\phi \iff$ EXEF ϕ from left to right. This way, we increase the likelihood that we finally obtain one of the formulas considered in Sect. 5.7. Moreover, we get larger subformulas that do not contain an X operator. Since CTL preserving stubborn sets work only on X-free formulas, stubborn reduction is not applicable to a formula containing an X operator as a whole. However, when an X-free subformula of a CTL formula is evaluated on some level of recursion in the procedure sketched in Sect. 3, there is no reason not to apply stubborn sets. Hence, the rewriting strategy improves the applicability of stubborn set reduction.

Traps. When investigating atomic propositions, [1] mainly employs the Petri net state equation. In quite some situations where the state equation is not able to prove a proposition to be invariantly true or false, a trap can actually help. A trap is a set Q of places that, once containing a token, always keeps at least one token. This is formally established by requiring that every transition that consumes tokens from any place in Q, also produces a token on some place in Q. Consider an atomic proposition $k_1p_1 + \cdots + k_np_n \geq 1$ with all k_i being positive. If $\{p_1, \ldots, p_n\}$ includes a trap that has at least one token in the initial marking, the proposition is invariantly true. Existence of a trap is easily checked. We start with $\{p_1, \ldots, p_n\}$ and remove places where some transition consumes tokens while not producing tokens on any of the places. This way, we obtain the maximal trap included in $\{p_1, \ldots, p_n\}$.

Embedded Place Invariants. A place invariant i assigns a weight $i(p)$ to every place p such that the weighted sum of tokens remains constant for all reachable markings. Place invariants can be found by solving the system of equations $C^T i = 0$, where C is the incidence matrix of N. With a place invariant i, the equation $i(p_1)m(p_1) + \cdots + i(p_n)m(p_n) = im_0$ holds for all reachable markings in a net with set of places $\{p_1, \ldots, p_n\}$. Sometimes, such invariants can be used for simplifying an atomic proposition. Consider as an example the proposition $p_1 + 2p_2 + p_3 \geq 2$ and assume that there is a place invariant that yields the equation $p_1 + p_2 = 1$. Then the atomic proposition can be simplified to $p_2 + p_3 \geq 1$.

It is not constant but does no longer mention p_1. Consequently, the set of visible transitions may become smaller since the environment of p_1 does no longer need to be considered as visible (unless transitions still appear in the environment of p_2 or p_3). With a smaller set of visible transitions, better stubborn set reduction may be expected (in particular regarding the VIS principle). In general, a helpful invariant can be systematically computed as the solution of a linear program. Consider an atomic proposition of the shape $k_1 p_1 + \cdots + k_m p_m + l_1 q_1 \cdots + l_n q_n \ op \ k$, where all p_j and q_j are places, all k_j are positive integers, all l_j are negative integers, $op \in \{=, \neq, <, >, \leq, \geq\}$, and k is an integer. The linear program looks for the largest possible invariant where the coefficients are between 0 and k_i (resp. l_i): Maximise $i(p_1)+\cdots+i(p_m)-i(q_1)-\cdots-i(q_n)$ where $C^T i = 0$, and $0 \leq i(p_j) \leq k_j$ (for $1 \leq j \leq m$), and $l_j \leq i(q_j) \leq 0$ (for $1 \leq j \leq n$). If the linear program is feasible, subtracting the resulting solution from the atomic proposition may or may not lead to less mentioned places but is guaranteed not to add places to the formal sum of the proposition.

7 Experimental Validation

We implemented the methods discussed in the paper in our tool LoLA (implementation does not yet cover EXEFEG, EXEGEF, and the universal counterparts AXAGAF and AXAFAG). For evaluating the methods, we use the benchmark provided by the MCC 2018 [9]. We used the formulas provided in the CTL category. While the nets of the MCC are contributed by the community, the formulas are actually generated automatically, and are to a certain degree random.

In 2018, a total of 767 place/transition Petri nets were used in the MCC. For every net, 32 CTL formulas are provided. This makes 24544 individual verification problems. For 3704 problems (15.1%), the initial rewriting process yielded a formula that does not contain any temporal operator. Here, sufficiently many atomic propositions have been found to be invariantly true or false. Resulting formulas can be evaluated by just inspecting the initial marking, so no actual run of a model checker is necessary. 13366 problems (54.5%), after rewriting, fall into some of the categories mentioned in Sect. 5. That is, we need to run the generic CTL model checker only for 30.4% of the CTL problems in the MCC!

For the 13366 problems where application of a special routine is possible, we compared the proposed routine with a run of the generic CTL model checking procedure. To this end, we used 300 seconds of execution time and unlimited memory for every problem instance. Experiments were executed on our machine Ebro. This machine has been used for executing parts of the actual MCC in recent years. It has 32 physical cores running at 2.7 GHz and 1 TB of RAM. Memory overflow was no issue within the 300 seconds given to each instance.

Table 1 lists the results of our experiments. It shows that specialised routines in total are more successful than the CTL model checking procedure. For the formulas where specialised routines have been found, we increased the success rate from 69.2% to 85.7%. In other words, specialised routines are able to solve more

Table 1. Comparison between CTL model checking procedure (CTL) and special routine (as proposed in this paper).

Formula type	Number	CTL		Special		Improvement	
	(#)	#	%	#	%	#	%
EF ϕ, AG ϕ	2471	1438	58.2	2300	93.1	862	34.9
EG ϕ, AF ϕ	1767	1625	92.0	1670	94.5	45	2.5
E(ϕ R ψ), A(ϕ U ψ)	168	157	93.5	160	95.2	3	1.8
E(ϕ U ψ), A(ϕ R ψ)	318	187	58.8	198	62.3	11	3.5
EFEG ϕ, AGAF ϕ	515	340	66.0	431	83.7	91	17.7
EGEF ϕ, AFAG ϕ	385	276	71.7	277	71.9	1	0.3
EXEF ϕ, AXAG ϕ	353	193	54.7	319	90.4	126	35.7
EXEG ϕ, AXAF ϕ	197	177	89.8	178	90.4	1	0.5
EXE(ϕ R ψ), AXA(ϕ U ψ)	19	17	89.5	18	94.7	1	5.3
EXE(ϕ U ψ), AXA(ϕ R ψ)	33	20	60.6	24	72.7	4	12.1
EFAG ϕ, AGEF ϕ	884	286	32.4	343	38.8	57	6.4
EFAGEF ϕ, AGEFAG ϕ	13	3	23.1	6	46.2	3	23.1
Single-Path	421	275	65.3	295	70.1	20	4.8
Boolean	5822	4250	73.0	5239	90.0	989	17.0
All	13366	9244	69.2	11458	85.7	2214	16.5

than half of the cases where a generic CTL model checker was not successful. This means that the proposed approach proved to be effective.

The table also shows that success is very unevenly distributed over the various formula types. The big success of reachability (EF ϕ) is of course to be expected and can be quoted to the large portfolio that included search with very powerful stubborn sets, the state equation approach, and the use of the siphon/trap property.

On the other edge of the spectrum, the little success for EGEF ϕ is well explained by the fact that we still need to apply the CTL-X preserving stubborn set method, so we have a more efficient exploration of the state space but the state space as such remains the same. In case of EXEG ϕ, the CTL model checker left only 20 problems open. That is, there is not much room for improvement. Problems in the MCC can be separated into the categories "easy enough for everybody","too hard for everybody", and "battleground". The first category refers to nets with rather small state space. Here, every approach is able to get a result in time. In the second category, we have nets with very large state spaces and dense dependencies between transitions. At least explicit model checkers that depend on the reduction power of the stubborn set method, have no chance to verify such systems. This means that progress in model checking mainly refers to the battleground category. We should aim at covering the problems in this category as much as possible. Returning to the EXEG ϕ category,

the little success may very well be due to the fact that only one of the 20 formulas left open by the CTL model checker actually fell into the battleground category. Consequently, we do *not* conclude that the special routine for EXEG ϕ is ineffective as such. Given the fact that we may apply more powerful stubborn sets, we have reason to believe that the procedure would be more effective on a different benchmark, with more EXEG formulas in the battleground category. In consequence, the large bandwidth of success rates in the different formula types does not jeopardise the general conclusion in favour of using specialised routines.

The checks for EF ϕ as a necessary condition and AG ϕ as a sufficient condition, which we run in parallel to the depth-first search, provided solutions to 305 problems (1.2% of the whole CTL category).

8 Conclusion

We proposed to relieve the CTL model checker by providing specialised support for a large set of simple CTL queries. Special treatment permits the use of much more powerful stubborn set dialects. In addition, the Petri net state equation may be employed for solving the problem, or for checking necessary or sufficient conditions in a portfolio approach. In the MCC, specialised routines are applicable to more than half of the problems. In the introduction, we argued that a significant percentage of simple queries has to be expected in practice, too.

With our approach we increased the success rate for simple formulas by 16.5% in the MCC benchmark. This is a remarkable achievement since none of the additionally solved problems falls into the "easy enough for everybody" category. Over half of the simple problems left unsolved by the CTL model checker can now be solved. The performance demonstrated here with the MCC benchmark (that uses randomly generated formulas) can be repeated in other situations with meaningful formulas. Unfortunately, there is not enough space to report details.

Offering the new methods, LoLA unfortunately does not yet reach the performance of TAPAAL [7], the 2018 winner of the MCC CTL category. TAPAAL offers some techniques that have not (yet) been implemented in LoLA. For instance, TAPAAL uses sophisticated net reduction as another form of preprocessing [7].

Future work could include finding more formula types that permit any improvement in verification. In addition, some of the ideas of this paper could be integrated into a CTL model checker itself. For instance, treating AGEF, or even AGEFAG, in a single depth-first search should be possible even if that pair of operators occurs in the middle of a more complex CTL formula. In addition, the proposed stubborn set dialects may not necessarily be the optimal ones for the respective formula type. Finding alternative stubborn set methods for larger classes of formulas, we may ultimately be able to have a dedicated dialect of stubborn sets for every subformula of a CTL query. Finally, the initially proposed idea of modifying the net for the sake of simplifying a CTL query has not been systematically explored yet.

References

1. Bønneland, F., Dyhr, J., Jensen, P.G., Johannsen, M., Srba, J.: Simplification of CTL formulae for efficient model checking of petri nets. In: Khomenko, V., Roux, O.H. (eds.) PETRI NETS 2018. LNCS, vol. 10877, pp. 143–163. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91268-4_8
2. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982). https://doi.org/10.1007/BFb0025774
3. Clarke, E.M., Draghicescu, I.A.: Expressibility results for linear-time and branching-time logics. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) REX 1988. LNCS, vol. 354, pp. 428–437. Springer, Heidelberg (1989). https://doi.org/10.1007/BFb0013029
4. Commoner, F.: Deadlocks in Petri Nets. Applied Data Research Inc., Wakefield, Massachusetts, Report CA-7206-2311 (1972)
5. Courcoubetis, C., Vardi, M.Y., Wolper, P., Yannakakis, M.: Memory-efficient algorithms for the verification of temporal properties. Formal Methods Syst. Des. **1**(2/3), 275–288 (1992)
6. Couvreur, J.-M., Duret-Lutz, A., Poitrenaud, D.: On-the-fly emptiness checks for generalized Büchi automata. In: Godefroid, P. (ed.) SPIN 2005. LNCS, vol. 3639, pp. 169–184. Springer, Heidelberg (2005). https://doi.org/10.1007/11537328_15
7. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K.Y., Møller, M.H., Srba, J.: TAPAAL 2.0: integrated development environment for timed-arc petri nets. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 492–497. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_36
8. Dehnert, J., Rittgen, P.: Relaxed soundness of business processes. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 157–170. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45341-5_11
9. Kordon, F., et al.: Homepage of the Model Checking Contest, June 2018. http://mcc.lip6.fr/
10. Geldenhuys, J., Valmari, A.: More efficient on-the-fly LTL verification with tarjan's algorithm. Theor. Comput. Sci. **345**(1), 60–82 (2005)
11. Gerth, R., Kuiper, R., Peled, D.A., Penczek, W.: A partial order approach to branching time logic model checking. Inf. Comput. **150**(2), 132–152 (1999)
12. Godefroid, P., Wolper, P.: A partial approach to model checking. Inf. Comput. **110**(2), 305–326 (1994)
13. Hack, M.H.T.: Analysis of Production Schemata by Petri Nets. Master's thesis, MIT, Dept. Electrical Engineering, Cambridge (1972)
14. Holzmann, G.J., Peled, D.A., Yannakakis, M.: On nested depth first search. In: Proceedings 2nd SPIN Workshop, pp. 23–32 (1996)
15. Kristensen, L.M., Schmidt, K., Valmari, A.: Question-guided stubborn set methods for state properties. Formal Methods Syst. Des. **29**(3), 215–251 (2006)
16. Maidl, M.: The common fragment of CTL and LTL. In: Proceedings of FOCS, pp. 643–652. IEEE Computer Society (2000)
17. Oanea, O., Wimmel, H., Wolf, K.: New algorithms for deciding the siphon-trap property. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 267–286. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13675-7_16

18. Peled, D.: All from one, one for all: on model checking using representatives. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 409–423. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-56922-7_34
19. Schmidt, K.: Stubborn sets for standard properties. In: Donatelli, S., Kleijn, J. (eds.) ICATPN 1999. LNCS, vol. 1639, pp. 46–65. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48745-X_4
20. Schmidt, K.: Stubborn sets for model checking the EF/AG fragment of CTL. Fundam. Inform. **43**(1–4), 331–341 (2000)
21. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)
22. Valmari, A.: Stubborn sets for reduced state space generation. In: Rozenberg, G. (ed.) ICATPN 1989. LNCS, vol. 483, pp. 491–515. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-53863-1_36
23. Valmari, A.: The state explosion problem. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 429–528. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_21
24. Valmari, A.: Stubborn set methods for process algebras. In: Proceedings of DIMACS Workshop on Partial Order Methods in Verification, vol. 29, pp. 213–231 (1997)
25. Valmari, A., Hansen, H.: Stubborn set intuition explained. In: Koutny, M., Kleijn, J., Penczek, W. (eds.) Transactions on Petri Nets and Other Models of Concurrency XII. LNCS, vol. 10470, pp. 140–165. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-55862-1_7
26. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency. LNCS, vol. 1043, pp. 238–266. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-60915-6_6
27. Vergauwen, B., Lewi, J.: A linear local model checking algorithm for CTL. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 447–461. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57208-2_31
28. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. Logical Methods Comput. Sci. **8**(3) (2012)
29. Wolf, K.: Petri net model checking with LoLA 2. In: Khomenko, V., Roux, O.H. (eds.) PETRI NETS 2018. LNCS, vol. 10877, pp. 351–362. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91268-4_18