Stubborn Sets for Simple Linear Time Properties

Andreas Lehmann, Niels Lohmann, and Karsten Wolf

Universität Rostock, Institut für Informatik

Abstract. We call a linear time property *simple* if counterexamples are accepted by a Büchi automaton that has only singleton strongly connected components. This class contains interesting properties such as LTL formulas $\mathbf{G}(\varphi \implies \mathbf{F}\,\psi)$ or $\varphi\,\mathbf{U}\,\psi$ which have not yet received support beyond general LTL preserving approaches.
We contribute a stubborn set approach to simple properties with the following ingredients. First, we decompose the verification problem into finitely many simpler problems that can be independently executed. Second, we propose a stubborn set method for the resulting problems that does neither require cycle detection, nor stuttering invariance, nor existence of transitions that are invisible to *all* atomic propositions. This means that our approach is applicable in cases where traditional approaches fail. Third, we show that sufficient potential is left in existing implementations of the proposed conditions by exploiting all the available nondeterminism in these procedures. We employ a translation to integer linear programming (ILP) for supporting this claim.

1 Introduction

There are two main approaches to the verification of temporal properties, both concerned with the alleviation of the state explosion. In symbolic model checking [1,2], sophisticated data structures represent a set S of states or paths such that the memory consumption does not strongly correlate to the number of elements contained in S. In explicit model checking, the original transition system is replaced by a smaller one that, by construction, is equivalent with respect to the investigated property. This paper is concerned with explicit model checking. Here, the partial order reduction [16,7,11] appears to be the most powerful reduction technique.

There is a broad spectrum of approaches to partial order reduction. On one end of the spectrum, there are approaches that preserve properties of a full temporal logic such as LTL [11,17] or CTL* [6] or a process algebraic semantics [18]. On the other end of the spectrum, singular properties or distinguished classes of properties are preserved such as deadlocks [16], reachability [10] or other standard properties [13]. Between these extremal techniques, there are approaches to smaller but nontrivial fragments of temporal logic [14].

In this paper, we generalize the technique used in [13] for singular properties to a class of properties that can be defined in terms of the structure of a Büchi

automaton. Büchi automata are an established tool for specifying linear time temporal properties and are closely related to the temporal logic LTL. We study Büchi automata where all loops are self loops; that is, the automaton has no strongly connected component with more than one element. For such properties, we decompose the verification problem into a finite number of even simpler verification problems. We present a new stubborn set method that preserves the resulting class of properties. The resulting technique complements existing approaches. In difference to traditional LTL preserving methods, it does not require the detection of cycles (at least not for stubborn set calculation), it tolerates a limited amount of stuttering in the investigated property, and it requires fewer transitions to be invisible with respect to the property.

For increasing the power of the required stubborn set calculation, we further study the impact of nondeterministic choices in a particular procedure for stubborn set calculation. To this end, we give a translation of the stubborn set calculation problem to an integer linear programming (ILP) problem that reflects all possible choices. This way, we import the mature heuristics that ILP offers for resolving nondeterminism. Experiments show that this translation pays off, even when the costly ILP procedure is involved.

The paper is organized as follows. We first introduce Petri nets (Sect. 2) and Büchi automata (Sect. 3). Then we introduce our notion of simple properties and discuss consequences (Sect. 4). We continue with a presentation of traditional stubborn set reduction for liner time properties (Sect. 5) and our new approach (Sect. 6). In Sect. 7, we present a translation of stubborn set calculation to integer linear programming. We conclude with a discussion of related work.

2 Petri Nets

For a set M, denote 2^M its power set. For sets M and N, let M^N be the set of functions $f : N \to M$. We use the following notation for Petri nets.

Definition 1 (Petri net). *A Petri net $N = [P, T, F, W, m_0]$ consists of two finite, nonempty, and disjoint sets P (of places) and T (of transitions), a flow relation $F \subseteq (P \times T) \cup (T \times P)$, a multiplicity assignment $W : F \to \mathbb{N} \setminus \{0\}$, and an initial marking $m_0 \in \mathbb{N}^P$.*

Places and transitions are collectively called nodes. For a node x, let $\bullet x = \{y \mid [y, x] \in F\}$ denote its pre-nodes while $x\bullet = \{y \mid [x, y] \in F\}$ denotes its post-nodes.

Definition 2 (Behavior). *Transition $t \in T$ is enabled in marking $m \in \mathbb{N}^P$ iff, for all $p \in \bullet t$, $m(p) \geq W(p, t)$. We denote this by $m \xrightarrow{t}$. If t is enabled, t can fire in m, yielding marking m' where, for all $p \in P$, $m'(p) = m(p) - W(p, t) + W(t, p)$ under the assumption that $W(x, y)$ is set to 0 for $[x, y] \notin F$. This relation is denoted by $m \xrightarrow{t} m'$.*

We investigate properties that are evaluated in marking sequences of N.

Definition 3 (Marking sequence). *A finite or infinite sequence of markings $m_0 m_1 m_2 \ldots$ ($m_i \in \mathbb{N}^P$) is a marking sequence of the Petri net N iff m_0 is the initial marking of N and, for all i, either there are no enabled transitions in m_i and $m_{i+1} = m_i$, or there is a transition t_i such that $m_i \xrightarrow{t_i} m_{i+1}$. We say that the corresponding transition sequence $t_0 t_1 \ldots$ produces the marking sequence $m_0 m_1 m_2 \ldots$. We denote executability of a transition sequence starting at m by $m \xrightarrow{t_i t_{i+1} \ldots}$ or, if the reached marking is relevant, by $m \xrightarrow{t_i t_{i+1} \ldots} m'$.*

3 Model Checking with Büchi Automata

In this paper, we consider *linear time* properties. A linear time property is one that can be evaluated in each single (typically infinite) run of the system in isolation. Such a property is true of the given system if all runs of the system satisfy it. A popular way to verify linear time properties is to construct a Büchi automaton that accepts all those infinite sequences which violate the property. Using standard product automaton construction, a new Büchi automaton is constructed that accepts all runs which are possible in the system *and* violate the property. If the given property is satisfied in the system, the resulting automaton accepts the empty language, otherwise any accepted run yields a counterexample.

Properties are built upon *propositions*. We postulate a countable set of *atomic* propositions which can combined using Boolean operators.

Definition 4 (Proposition). *Let $\mathcal{A}$ be some countable set, elements of which are called* atomic proposition*. Atomic propositions are* propositions*. If φ and ψ are propositions, so are $(\varphi \wedge \psi)$ and $(\varphi \vee \psi)$. Let $\mathcal{P}_\mathcal{A}$ the set of all propositions over $\mathcal{A}$.*

Subsequently, we omit parentheses relying on the usual precedence of the $\wedge$ and $\vee$ operators. We assume that $\mathcal{A}$ is arbitrary but fixed throughout the paper. In examples, we shall use atomic propositions of the form $p \geq k$ and $p < k$ where p is a place of the investigated Petri net and k is some natural number. Propositions are evaluated in markings. For the atomic propositions, we assume that their interpretation is somehow given by a relation $\models \subseteq \mathbb{N}^P \times \mathcal{A}$. In our example set of atomic propositions, let $m \models p \geq k$ iff $m(p) \geq k$ and $m \models p < k$ iff $m(p) < k$. Relation $\models$ is canonically lifted to arbitrary propositions by stating $m \models (\varphi \wedge \psi)$ ($m \models (\varphi \vee \psi)$) iff $m \models \varphi$ and (or) $m \models \psi$.

We call $\mathcal{A}$ *closed under negation* if, for every $a \in \mathcal{A}$, there is a $b \in \mathcal{A}$ such that, for all m, $m \not\models a$ if and only if $m \models b$. Throughout the paper, we assume $\mathcal{A}$ to be closed under negation which justifies the absense of negation in Def. 4: negation can easily be removed using de Morgan's laws. Our example set of atomic propositions is obviously closed under negation. We furthermore assume that there is a proposition *tt* that is true of all markings and a proposition *ff* that is false of all markings. In our examples, $tt := p \geq 0$ and $ff := p < 0$ satisfy these requirements.

We define Büchi automata such that they are directly controlled by propositions. For defining accepting runs of the automaton, we need to observe that

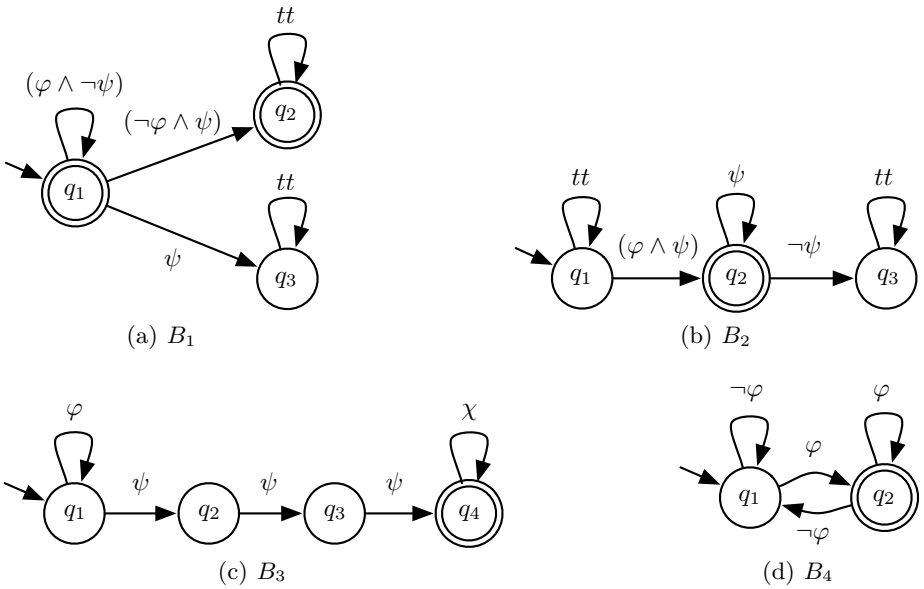


Fig. 1. Examples of Büchi automata

the Büchi automaton attaches propositions to transitions while, in the system, propositions are related to markings.

Definition 5 (Büchi automaton). *A Büchi automaton* $B = [Q, q_0, \delta, \lambda, F]$ *consists of*

- *a finite set* Q *of states;*
- *an initial state* $q_0 \in Q$*;*
- *a transition relation* $\delta \subseteq Q \times Q$*;*
- *a labeling function* $\lambda : \delta \to \mathcal{P}_{\mathcal{A}}$*;*
- *a set* $F \subseteq Q$ *of finite states.*

B accepts an infinite sequence of markings $m_1 m_2 m_3 \ldots$ *if and only if there is an infinite sequence* $q_0 q_1 q_2 q_3 \ldots$ *such that*

- *for all* i, $[q_i, q_{i+1}] \in \delta$*;*
- *for all* $i > 0$, $m_i \models \lambda([q_{i-1}, q_i])$*;*
- *for infinitely many* i, $q_i \in F$*.*

Figure 1 shows a few examples of Büchi automata.

The first state of a state sequence is of course the designated initial state of B. Acceptance of a Büchi automaton as defined above is in fact nondeterministic. For some state q and marking m, there may exist several q_1, q_2 such that $q_1 \neq q_2$, $m \models \lambda([q, q_1])$, and $m \models \lambda([q, q_2])$.

The product system $N \cap B$ of a Petri net N and a Büchi automaton B is a Büchi automaton again.

Definition 6 (Product System). *Let $N = [P, T, F, W, m_0]$ be a Petri net and $B = [Q, q_0, \delta, \lambda, F]$ be a Büchi automaton. The product system $N \cap B$ is a Büchi automaton $B^* = [Q^*, q_0^*, \delta^*, \lambda^*, F^*]$ where*

- *q_0^* is some element not contained in $\mathbb{N}^P \times Q$*
- *$Q^* = \{q_0^*\} \cup (\mathbb{N}^P \times Q)$;*
- *$[q_0^*, [m, q]] \in \delta^*$ iff $[q_0, q] \in \delta$, $m_0 \models \lambda([q_0, q])$, and $m = m_0$;*
- *$[[m, q], [m', q']] \in \delta^*$ iff $m \to m'$, $[q, q'] \in \delta$, $m' \models \lambda([q, q'])$;*
- *for all $[q^*, q^{*\prime}] \in \delta^*$, $\lambda([q^*, q^{*\prime}]) = tt$;*
- *$[m, q] \in F^*$ iff $q \in F$.*

The resulting Büchi automaton labels all transition with tt. That is, the actual input sequence is irrelevant and the product system can be seen as a closed system. Nevertheless, standard theory asserts:

Proposition 1 (Emptiness). *There is an infinite sequence of markings realizable in N and accepted by B if and only if the product system $N \cap B$ has an accepting run.*

For bounded Petri nets, the product system has an accepting run if and only if it has a strongly connected component (w.r.t. δ) that is reachable from the initial state and contains a final state.

There is a close link between Büchi automata and the linear time temporal logic LTL. In our setting, LTL formulas are interpreted on infinite sequences of markings. Sloppily introduced, LTL comprises propositions (true of a sequence if satisfied in the first marking), Boolean operations (with the usual meaning), and temporal operators **X** ("holds in the next state"), **F** ("holds eventually"), **G** ("holds invariantly"), and **U** ("one property holds until another one is satisfied"). For LTL, we know that:

Proposition 2 (LTL). *For every LTL formula φ, there is a Büchi automaton that accepts exactly those sequences of markings which violate φ.*

Büchi automaton B_1 (Fig. 1) accepts all sequences that violate the LTL property $\varphi \,\mathbf{U}\, \psi$. B_2 accepts all sequences that violate $\mathbf{G}(\varphi \implies \mathbf{F}\, \psi)$.

In this paper, all results rely on Büchi automata as such. We use LTL only for the purpose of linking our results to related work. For this reason, we skip a formal definition of syntax and semantics of LTL.

4 Simple Büchi Automata

For a binary relation R, let R^* denote its reflexive and transitive closure.

Definition 7 (Simple Property). *A Büchi automaton is* simple *iff, for all $q, q' \in Q$, $[q, q'] \in \delta^*$ and $[q', q] \in \delta^*$ implies $q = q'$. A linear time property is simple iff the set of infinite runs violating the property is accepted by a simple Büchi automaton.*

In a simple Büchi automaton, the only occurring cycles in the transition relation are self-loops at states. In other words, all strongly connected components reachable from the initial state are singletons. In Fig. 1, automata B_1, B_2, and B_3 are simple whereas B_4 is not.

The class of simple properties contains several relevant properties that are expressible in LTL. Among them, there is the formula $\mathbf{G}(\varphi \implies \mathbf{F}\,\neg\psi)$ (for arbitrary propositions φ and ψ, see B_2 in Fig. 1) which has been shown to be central in the verification of distributed algorithms in [12]. $\mathbf{F}\,\varphi$, $\mathbf{G}\,\varphi$, and $\varphi\,\mathbf{U}\,\psi$ are other examples of simple LTL properties. In contrast, $\mathbf{FG}\,\varphi$ is not simple. Marking sequences that violate $\mathbf{FG}\,\varphi$ contain infinitely many markings where φ is not satisfied. For these sequences, the set of states of the Büchi automaton that are entered infinitely often must contain both a final and a non-final state since we need to leave the final state when φ is violated and to enter it when φ is satisfied. In this case, however, these two states would form a cycle contradicting simplicity. Automaton B_4 in Fig. 1 accepts all sequences that violate $\mathbf{FG}\,\neg\varphi$.

As self loops (states q with $[q, q] \in \delta$) are the only cycles in simple Büchi automata, accepting runs have only one state that occurs infinitely often. This must obviously be a final state. In addition, there is no state re-entered after having been left. This can be exploited for replacing simple Büchi automata by even simpler automata.

Definition 8 (Elementary Büchi automaton). *A simple Büchi automaton is elementary iff*

- *for each state q there is at most one state q' such that $[q, q'] \in \delta$ and $q \neq q'$;*
- *if $q \in F$ then $[q, q'] \in \delta$ implies $q = q'$.*

That is, if unreachable states are removed, an elementary Büchi automaton consists of a nonbranching sequence of states which may or may not have self loops and the last of which is a final state.

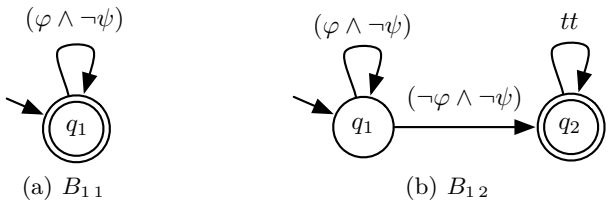
It is not difficult to see

Lemma 1. *For each simple Büchi automaton B, there is a finite set of elementary Büchi automata $\mathcal{M}$ such that a marking sequence is accepted by B if and only if it is accepted by one automaton in $\mathcal{M}$.*

$\mathcal{M}$ is obtained by enumerating all self-loop free paths in B that end in a final state of B. By the simplicity property, there exist only finitely many such paths. For each path, its set of states together with transitions between subsequent states and self loops (as far as present in B) form one automaton to be included in $\mathcal{M}$. Only the last state in the path is final in the resulting automaton.

Figure 3 shows the two elementary Büchi automata that are the result of decomposing B_1 in Fig. 1. Decomposition of B_2 in the same figure yields the automaton depicted in Fig: 3.

If a non-elementary simple Büchi automaton B is decomposed into a set of elementary Büchi automata $\mathcal{M}$, it is clear that each element $B^* \in \mathcal{M}$ has a simpler structure than B. In particular, its structure is embedded in B. That is, we can immediately observe



(a) $B_{1\,1}$ (b) $B_{1\,2}$

Fig. 2. Decomposition of B_1 (Fig.1) into elementary automata

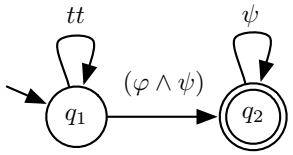


Fig. 3. Decomposition of B_2 (Fig.1) into elementary automata

Corollary 1. *Let N be a Petri net, B a simple Büchi automaton, and $\mathcal{M}$ its decomposition into elementary Büchi automaton. For every $B^* \in \mathcal{M}$, the number of states of $N \cap B^*$ is less than or equal to the number of states of $N \cap B$.*

Of course, the decomposition of B may contain a large number of elementary automata (in fact, up to exponentially many). Moreover, as some of the resulting automata share common prefixes, the accumulated run time for the product systems $N \cap B^*$ for all $B^* \in \mathcal{M}$ may take more time than producing $N \cap B$. However, the limiting resource for model checking is still the available memory and a large number of state spaces each of which fit into memory is more valuable than a single state space that does not.

Another advantage of decomposition is that each resulting automaton B^* operates only on some of the proposition that occur in B. This means that more transitions of N become invisible which is favorable for traditional LTL preserving stubborn set methods (see next section).

Using this argument, the remainder of this paper is concerned with elementary Büchi automata only. Hence, we conclude this section with the introduction of some terminology. Without loss of generality, we represent an elementary Büchi automaton using a set $\{q_0, \ldots, q_n\}$ such that q_0 is the initial state and $\delta = \{[q_i, q_{i+1}] \mid i < n\} \cup \{[q_i, q_i] \mid i \leq n\}$. Consequently, $F = \{q_n\}$. Every elementary automaton can be expressed this way by renaming its states. In particular, missing self loops can be added by labeling them with formula *ff*.

For state q_i, we call $\varphi_i := \lambda([q_i, q_{i+1}])$ the *progressing* formula at q_i as its satisfaction is necessary for leaving q_i and getting closer to the final state. Likewise, call $\psi_i := \lambda([q_i, q_i])$ the *retarding* formula at q_i.

5 Traditional LTL preserving stubborn sets

For self-containedness, we sketch the core concepts of the classical LTL preserving stubborn set technique. Stubborn set reduction aims at replacing a state space with a smaller one that is equivalent to the original one with respect to the original one. For linear time properties, equivalence means that the reduced state space has an accepting run if and only if the original does. Reduction is obtained by considering, in each state only some transitions. The set of transitions to be considered is controlled by a function $stub : \mathbb{N}^P \to 2^T$. In each marking m, only transitions in $stub(m)$ are considered. $stub(m)$ is called the stubborn set in m. For a given function *stub*, the reduced state space is obtained by replacing the condition $m \to m'$ in Def. 6 with the condition *there is a transition* t *in* $stub(m)$ *such that* $m \xrightarrow{t} m'$. While only enabled transitions in $stub(m)$ are considered for producing successor states, it is convenient to permit disabled transitions as members of $stub(m)$, too.

Equivalence of original and reduced transition system can only be established if certain conditions are met by the stubborn set generator *stub*. For preserving a linear time property, $stub(m)$ must contain an enabled transition if there is any in m and meet the requirements of *commutation*, *invisibility*, and *non-ignorance* each of which is discussed subsequently. Furthermore, it is required that the investigated property is *stuttering invariant*. A linear time property is stuttering invariant iff, for all finite marking sequences π_1, all markings m, and all infinite marking sequences π_2, the property is satisfied in the path $\pi_1 m \pi_2$ if and only if it is satisfied in the path $\pi_1 m m \pi_2$. For an LTL formula, absence of the $\mathbf{X}$ operator is sufficient but not necessary for stuttering invariance. For instance, $\mathbf{F}(\varphi \wedge \mathbf{X}\,\psi)$ is stuttering invariant if ψ is the negation of φ.

Properties represented by B_1, B_2, and B_4 in Fig. 1 are stuttering invariant, whereas the one represented by B_3 is not.

5.1 Commutation

The requirement of commutation is the heart of any stubborn set method.

Definition 9 (Commutation). *A stubborn set generator stub satisfies the commutation requirement at marking* m *iff, for at least one* $t^* \in stub(m)$*, all sequences* w *of transitions outside* $stub(m)$ *and all* $t \in stub(m)$*,*

- $m \xrightarrow{wt} m'$ *implies* $m \xrightarrow{tw} m'$*;*
- m *does not enable any transition or* $m \xrightarrow{t^*}$ *and* $m \xrightarrow{w}$ *implies* $m \xrightarrow{wt^*}$ *.*

For preserving linear time properties, the traditional stubborn set method requires that the commutation requirement is satisfied in all markings.

For every marking m and every set $U \subseteq T$ of transitions, there exists a set U' of transitions such that $U \subseteq U'$ and U' meets the commutation requirement at m. This statement is trivial as $U' := T$ always satisfies the commutation requirement. It is nevertheless useful to postulate the existence of a function

$closure : \mathbb{N}^P \times 2^T \to 2^T$ such that, for al markings m and all sets U of transitions, $closure(m, U)$ is a — desirably small — set of transitions that includes U and meets the commutation requirement at m.

Several procedures for implementing *closure* have been discussed in the literature [20,18]. A very simple procedure (implemented in our tool LoLA) transitively includes transitions according to the following requirements: (E) if t is contained and enabled in m, then $(\bullet t)\bullet$ must be included, too, and (D) if t is included and disabled in m then $\bullet p$ must be included, too, for some insufficiently marked place p (i.e., $m(p) < W([p, t])$).

5.2 Invisibility

Invisibility is the only requirement in the traditional LTL preserving stubborn set method that depends on the verified property. For space reasons, we present a rather coarse version of the requirement. A finer version of the requirement can be found in [17].

Definition 10 (Invisibility). *Transition t is invisible for proposition φ iff, for all $m, m' \in \mathbb{N}^P$ with $m \xrightarrow{t} m'$, $m \models \varphi$ if and only if $m' \models \varphi$. A stubborn set generator obeys the invisibility requirement if, for all markings m, it either returns all transitions or only transitions that, if enabled, are invisible to all propositions occurring in the involved Büchi automaton.*

The purpose of the invisibility requirement is to make sure that the sequences wt and tw considered for the commutation requirement produce marking sequences that differ only by stuttering.

5.3 Non-Ignorance

Preservation of a linear time property can be proved by gradually transforming an accepting run of the original system into an accepting run of the reduced system. The main tool for transformation is the commutation requirement, as follows. A path $\pi_1 wt\pi_2$ of the original system where π_1 is executable in the reduced system, $m_0 \xrightarrow{\pi_1} m$, w being a sequence of transitions not using transitions in $stub(m)$, and $t \in stub(m)$, is transformed into $\pi_1 tw\pi_2$ which, by the commutation requirement, is equally well executable in the original. Moreover, as $t \in stub(m)$, at least $\pi_1 t$ is now executable in the reduced system as well. Repeated application of this argument produces the accepting run in the reduced system.

The invisibility property ensures that $\pi_1 wt\pi_2$ and $\pi_1 tw\pi_2$ produce marking sequences that differ only by stuttering. The same is true for any finite number of modifications of the path. Unfortunately, the argument cannot be continued to an infinite number of transformations. Assume that m does not satisfy φ but the occurrence of w makes φ hold. That is, the original path satisfies $\mathbf{F}\,\varphi$. If, however, an infinite number of invisible transitions is shifted in front of w, the resulting path is an infinite sequence of markings none of which satisfies φ, so the resulting path does not satisfy $\mathbf{F}\,\varphi$. The non-ignorance requirement repairs this problem.

Definition 11 (Non-Ignorance). *A stubborn set generator stub obeys the non-ignorance requirement for a Petri net N if, in every cycle of the reduced product system, there is at least one marking where $stub(m) = T$.*

By this condition, only finitely many transformations of the original path are necessary before the first transition of w can be shown to be executable in the reduced system as well. So, all (especially all visible) transitions of the original path can be executed in the reduced system thus ensuring the desired preservation of the verified property.

Non-ignorance is typically implemented by starting with a stubborn set generator *stub* that does not care about ignorance. Whenever a cycle is detected in the reduced state space, $stub(m)$ is augmented to the whole T. In order to safely detect all cycles, the generation of the reduced system is usually done according to the depth first strategy. Here, every cycle contains a state that has a successor on the depth first search stack which is a very simple procedure for cycle detection. This approach is, however, quite problematic especially for distributed state space generation where a strict depth first order blocks the parallel exploration of states. In addition, it is widely believed that the non-ignorance requirement is to a large extent responsible for unsatisfactory reduction results.

6 Our Approach to Stubborn Sets for Simple Properties

In our approach, we would like to use the stubborn sets for navigating the generation of the product system through the Büchi automaton. By the results of Sect. 4, we only need to consider elementary Büchi automata. For being able to tinvolve the current state of the Büchi automaton, we modify the signature of the stubborn set generator. Instead of $stub : \mathbb{N}^P \to 2^T$, we use $stub : \mathbb{N}^P \times Q \to 2^T$. In a non-final Büchi state q_i, two formulas are important. First, there is the (unique) progressing formula φ_i. Its satisfaction can take us to the next Büchi state and hence closer to acceptance. Second, there is the retarding formula ψ_i. This formula must not be violated until the progressing formula becomes true. Otherwise, the Büchi automaton would not accept either. In the unique final state of the Büchi automaton, our goal is to stay there forever. This state has only a retarding formula to be considered. Consequently, we have two objectives in stubborn set generation: we would like to "switch on" the progressing formula while, at the same time, avoid "switching off" the retarding formula. In the sequel, we first study these two objectives in isolation. Then we combine them to our stubborn set approach for elementary Büchi automata, and finally we compare the approach to the traditional LTL preserving method.

6.1 Stubborn sets for switching on formulas

For this part, we import the ideas from [13] and present them just for being self-contained. The main concept in his subsection are up-sets.

Definition 12 (up-set). *Consider a marking m and a proposition φ such that $m \not\models \varphi$. A set U of transitions is an up-set for m and φ if every transition sequence w such that $m \xrightarrow{w} m'$ and $m' \models \varphi$ contains at least one element of U.*

In other words, φ remains violated as long as only transitions in $T \setminus U$ are fired. Let $UP(m, \varphi)$ the family of all up-sets for m and φ and observe that $UP(m, \varphi)$ is not defined if $m \models \varphi$.

For our stubborn set approach, we are interested in small up-sets for given m and φ. Calculation of up-sets can typically be done by just considering m, the structure of φ, and the topology of N. A reasonable up-set for atomic proposition $p \geq k$ is $\bullet p$ while $p\bullet$ can be used for $p < k$. If $m \not\models \varphi$ any up-set of φ is an up-set for $\varphi \wedge \psi$. Symmetrically, if $m \not\models \psi$, any up-set for ψ is an up-set for $\varphi \wedge \psi$ as well. In cases where neither φ nor ψ are satisfied, we can actually choose between an up-set for φ and an up-set for ψ when trying to find one for $\varphi \wedge \psi$. For $\varphi \vee \psi$, an up-set can be constructed as the union of some up-set for φ and an up-set for ψ. Both $UP(m, \varphi)$ and $UP(m, \psi)$ are defined since violation of $\varphi \vee \psi$ at m implies violation of both φ and ψ at m.

The stubborn set approach of [13] just relies on up-sets and the commutation requirement. The following results is concerned with the state space of a Petri net rather than a product system.

Lemma 2 ([13]). *Let N be a Petri net and φ a proposition. Assume that, at every marking m where $m \not\models \varphi$, $stub(m)$ includes U for some $U \in UP(m, \varphi)$ and satisfies the commutation requirement. Then the full state space contains some marking that satisfies φ if and only if the reduced system does.*

If the original state space does not reach a marking that satisfies φ, then such a state cannot be reachable in the reduced state space as we never fire disabled transitions. Assume that the full state space contains a marking m^* satisfying φ but the reduced state space does not. Let $\pi_1\pi_2$ be a transition sequence that is executable in the full state space such that $m_0 \xrightarrow{\pi_1\pi_2} m^*$, π_1 can be executed in the reduced system, and the length of π_2 is minimal. The minimum exists since some m satisfying φ is reachable in the full state space and m_0 is part of the reduced state space (so π_1 is in worst case the empty sequence). Let m be the marking where $m_0 \xrightarrow{\pi_1} m$, so m occurs in the reduced system. By our assumptions, $m \not\models \varphi$, so $stub(m)$ includes an up-set for m and φ. One element of this up-set must occur in π_2 since otherwise φ would be violated in m^*. That is, π_2 contains elements of $stub(m)$ and can thus be separated into $\pi_2 = w_1 t w_2$ where w_1 contains only transitions in $T \setminus stub(m)$ and $t \in stub(m)$. By the commutation requirement, tw_1w_2 can be executed in m as well and still reaches m^*. However, this means that we have $\pi_1' = \pi_1 t$ and $\pi_2' = w_1 w_2$ such that $m_0 \xrightarrow{\pi_1'\pi_2'} m^*$, π_1' can be executed in the reduced system, and the length of π_2' is shorter than the length of π_2. This contradicts the required minimality for the length of π_2.

Stubborn set calculation using up-sets is implemented in the Petri net based verification tool LoLA [21]. In case studies [4,15] and a model checking contest [9],

the technique turned out to perform very well. In nets where a marking satisfying φ is indeed reachable, LoLA often computes only little more than the path from m_0 to a target marking. The computed path tends to be only little longer than the shortest possible one. If a marking that satisfies φ is not reachable, the stubborn set method proposed in [10] produces smaller state spaces but does not have such a strong goal orientation on satisfiable instances.

6.2 Preventing Formulas from Being Switched Off

For preserving the value of a retarding formula, we propose a stubborn set that contains only invisible enabled transitions. In difference to the traditional stubborn set technique for LTL, invisibility refers only to the retarding formula at the current state (and not for all propositions). Another difference is that we completely drop the non-ignorance requirement. For non-final Büchi states, the up-sets replace non-ignorance. For the final Büchi state, infinite stuttering is rather welcome, so the absence of a non-stuttering requirement does not harm.

6.3 Stubborn Sets for Elementary Büchi automata

Let B be an elementary Büchi automaton and N a Petri net. We aim at constructing a reduced product system that accepts some path if and only if $N \cap B$ does. Besides the ideas presented above, we only need two more ingredients. First, there are states where the above ideas cannot be applied. In these situations, the fallback solution is always to include *all* transitions in the stubborn set. Second, we would like to encapsulate the up-set based arguments such that the considered portion of a path does not change the Büchi state. To do that, we will require an up-set that does not contain enabled transitions. This way, the definition of up-sets assures that the progressing formula remains false by firing any transition in the stubborn set. Hence, we stay in the same Büchi state.

Definition 13 (Stubborn set for elementary Büchi automata). *Let B be an elementary Büchi automaton, q_i a state with progressing formula φ_i and retarding formula ψ_i, and m a marking of Petri net N. $stub(m, q_i)$ is any set of transitions of N that satisfies the commutation requirement, the invisibility requirement with respect to ψ_i and:*

- *If q_i is the final state of B then $stub(m, q_i)$ contains at least one enabled transition if m has one;*
- *If q_i is a non-final state of B then either $m \not\models \varphi_i$ and $stub(m, q_i)$ contains an up-set U for m and the progressing formula φ_i such that U does not contain any transition enabled at m, or $stub(m, q_i) = T$.*

This definition brings us immediately to the main theorem of this paper.

Theorem 1 (Preservation of linear time properties). *Let B be an elementary Büchi automaton and N a Petri net. Assume that a reduced state space is constructed using stubborn set $stub(m, q)$ in state $[m, q]$. Assume further that stub meets the conditions of Def. 13. Then $N \cap B$ accepts some path if and only if the reduced product system does.*

Proof. As the reduced system is a subsystem of the full one, acceptance in the reduced system trivially implies acceptance of the same path in the full system. Assume that the full product system accepts some marking sequence while the reduced system does not accept any marking sequence. Let π be an accepting run in $N \cap B$, that is, π is a sequence of elements in $\mathbb{N}^P \times Q$. Separate π into the finite sequence π_1 and the infinite sequence π_2 such that $\pi = \pi_1\pi_2$ and π_1 is the largest prefix of π that is also executable in the reduced system. Assume without loss of generality that π is selected such that the number of elements in π_2 with non-final Büchi states is minimal. Let $[m, q]$ be the last element of π_1. As π_1 is the largest prefix executable in the reduced state space, $stub(m, q)$ does not contain all enabled transitions. We consider two cases.

In the first case, assume that q is the final state of the Büchi automaton. Then, by construction of *stub*, $stub(m, q)$ contains at least one enabled transition if m has one, all enabled transitions in $stub(m, q)$ are invisible with respect to the retarding formula ψ at q, and the commutation requirement is satisfied. Let $t_1t_2\ldots$ be the transition sequence that produces the infinite marking sequence $[m, q]\pi_2$ (i.e. t_1 is fired in m). We separate two sub-cases. In case 1.1, there is a i such that $t_j \in stub(m, q_i)$, then let j be the smallest such i. This means that $t_1 \ldots t_{j-1}$ consists of transitions in $T\backslash stub(m, q)$. By the commutation requirement, sequence $t_jt_1t_2 \ldots t_{j-1}t_{j+1}t_{j+2} \ldots$ is executable at m. As t_j is invisible for ψ, the marking sequence produced by the modified transition sequence is an accepting one in the full product system but has a larger prefix executable in the reduced system. In case 1.2, the whole sequence $t_1t_2\ldots$ consists only of transitions not in $stub(m)$. Then either m does not have enabled transitions in which case acceptance is trivial, or $stub(m)$ contains a transition t^* that satisfies the second condition of the commutation requirement. For this t^*, combining the two conditions of the commutation requirement, we have that $t^*t_1t_2\ldots$ can be executed at m. The produced marking sequence is accepted in the full product system since the invisibility condition applies to t^*. Again, a larger prefix is executable in the reduced system. As we do not leave the final Büchi state, the arguments of the first case can be repeated, finally yielding an infinite accepting run in the reduced system.

In the second case, q is a non-final state of the Büchi automaton. Separate π_2 into $\pi_2 = \pi_{2,1}\pi_{2,2}$ such that all elements of $\pi_{2,1}$ have q as their Büchi state while no element of $\pi_{2,2}$ has q as its Büchi state. This separation is possible since we consider an elementary Büchi automaton. Let $[m^*, q^*]$ be the first element of $\pi_{2,2}$. Let $t_1 \ldots t_n$ be the marking sequence that produces the markings occurring in $[m, q]\pi_{2,1}[m^*, q^*]$. In particular, we have $m \xrightarrow{t_1\ldots t_n} m^*$. Let φ be the progressing formula at q and ψ the retarding formula at q. Since $stub(m, q) \neq T$, we have that $m \not\models \varphi$ and there is an up-set U for m and φ that is contained in $stub(m, q)$ and does not contain transitions enabled at m. On the other hand, since $[m^*, q^*]$ is the first state in $\pi_{2,2}$, we have that $q^* \neq q$, so $m^* \models \varphi$. By the property of up-sets, some element of U occurs in $t_1 \ldots t_n$. Since U is included in $stub(m, q)$, we can rephrase this statement into: some element of $stub(m, q)$ occurs in $t_1 \ldots t_n$. Let i be the smallest index such that $t_i \in stub(m, q)$. Let m' be the marking

where $m \xrightarrow{t_1 \ldots t_i} m'$. By the commutation requirement, t_i is enabled in m, so $t_i \notin U$. Hence, $m' \not\models \varphi$ and, in particular, $m' \neq m^*$. Arguing once more with the commutation requirement, we have $m \xrightarrow{t_i t_1 t_2 \ldots t_{i-1}} m'$. Since none of the involved markings satisfies φ and the invisibility condition for ψ applies to t_i, this sequence produces a run in the full product system that ends in state $[m', q]$. From there, we can continue as in the original path thus having exhibited another accepting run but with smaller distance to the final Büchi state. This contradicts an initial assumption on the choice of π.

6.4 Comparison

Our stubborn set approach differs in various regards from the traditional LTL preserving method.

First, our stubborn sets can be determined purely from N, m, q_i, φ_i, and ψ_i. In particular, we do not need to check for cycles in the reduced state space (except for checking the Büchi acceptance condition). This simplifies reduced state space generation with state space exploration techniques other than depth-first order. Hence, our approach permits flexibility in the search strategy for a accepted run and enables distributed state space generation. We have completely dropped the non-ignorance requirement.

Second, we require invisibility only for retarding formulas, and only one at a time. For progressing formulas not a single invisible transition is formally required. That is, we can expect reduction for properties where progress towards the accepting state depends on non-local events. For the retarding formulas, consideration of one at a time increases the odds that a stubborn set containing only invisible enabled transitions can be found. On the other hand, we require to have a complete up-set included in the stubborn set which may decrease the odds of finding a stubborn set with invisible enabled transitions only. Experimentation beyond this paper will be necessary for finding out whether the overall effect is positive or negative.

Third, we would like to emphasize that we did not require stuttering invariance of the verified property. Stuttering invariance can be violated if some Büchi state q has *ff* as its retarding formula. Our stubborn set method will yield $stub(m, q) = T$. However, if other states do have retarding formulas different from *ff*, the reduced product system can still be smaller than the full one. If we verify using automaton B_3 in Fig. 1, reduction can at least be obtained for states of the product system where the Büchi state is q_1 or q_4.

7 Calculation of Stubborn Sets Using ILP Solving

In the previous section, we learned that the main thread for the size of the stubborn set is fact that two requirements may interfere: the inclusion of an up-set and the invisibility condition. A bad up-set may inevitably cause the insertion of a visible transition while another up-set may not cause this problem. Fortunately, for a fixed marking m and a fixed proposition φ, there may be several

different up-sets. An existing implementation of the up-set approach to simple reachability queries in the tool LoLA does not compute and compare up-sets — it uses just an arbitrary one. In connection with the invisibility condition which is not required for reachability, we feel that it is necessary to optimize the choice of an up-set. When computing stubborn sets in LoLA, there is another source of nondeterminism that is not fully exploited yet. In establishing the commutation requirement (in particular, condition (D) in Sect. 5.1), a disabled transition may have several insufficiently marked pre-places and we need to consider only one of them.

Consequently, we would like to study the potential gain that can be obtained by considering all instead of just one up-set and by considering all instead of an arbitrarily fixed insufficiently marked pre-place. To avoid too bad run-time penalties, we import a mature mechanism for managing the huge amount of resulting nondeterminism. To this end, we translate the stubborn set calculation used in LoLA into an integer linear programming (or ILP) problem. The LoLA procedure starts with an arbitrary up-set. For each enabled transition, it includes the conflicting ones and for each disabled transition, it includes the pre-transitions of some insufficiently marked pre-place. If, during this procedure, a visible transition becomes prt of the stubborn set, LoLA returns the set of all enabled transitions.

Translation to ILP brings us to the question of the optimality criterion for stubborn sets. Optimality of stubborn sets has been studied in depth [20,19]. Two results are relevant here. First, if there are two stubborn sets U and V such that $U \subseteq V$ then U is always the better choice; that is, yields better reduction. If there are two stubborn sets U and V such that $card(U) < card(V)$ then U may or may not be the better choice, that is, replacing V with U may decrease or increase the size of the overall state space. The reason is that transitions in U may generate successor markings where only poor reduction is obtained while transitions in V may avoid these states. However, the results only state that U *may* be not the best choice. It does not state that it never *is* the best choice. For this reason, we decided to use a target function of the ILP problem where the number of enabled transitions in the stubborn set is minimized.

For our ILP problem, we include three sets of integer variables. The first set is T. We shall construct the inequations such that a value of 1 assigned to t in the solution means that t is an element the stubborn set while a value of 0 means that t is not an element of the stubborn set. The second set of variables is P. A value of 1 assigned to p means that p is the selected insufficiently marked pre-place of some disabled transition in the closure operation sketched in Sect. 5.1. The third set of variables consists of one variable for each up-set we want to consider.

We continue with presenting the actual translation of the stubborn set calculation into an ILP problem under the assumption that a family of up-sets to be considered is given. We continue with an enumeration of reasonable up-sets and conclude with experimental results.

7.1 Specification of the ILP Problem

For stubborn set calculation in the case of a non-final state, the inputs are a net N, a marking m, a family of up-sets $\mathcal{M}$, a progressing formula φ and a retarding formula ψ. The target function of or ILP problem states that we want to minimize the number of enabled transition in the set.

$$\sum_{t \in T \mid m \xrightarrow{t}} t = MIN.$$

For all variables, we want them to have integer values between 0 and 1.

$$\forall t \in T : 0 \leq t \leq 1 \quad \forall p \in P : 0 \leq p \leq 1 \quad \forall M \in \mathcal{M} : 0 \leq M \leq 1$$

Next, we want to have one of the up-sets included in the stubborn set. In the sequel, we shall use the inequation $y + 1 - x > 0$ for modeling the Boolean relation $x \implies y$.

$$\sum_{M \in \mathcal{M}} M = 1 \quad \forall M \forall t \in M : t + 1 - M > 0$$

The invisibility condition can be stated as follows:

$$\sum_{t : t \text{ visible to } \varphi,\ m \xrightarrow{t}} t = 0$$

The commutation requirement for enabled transitions can be coded in the following way.

$$\forall t : t \text{ enabled in } m \ \ \forall t' \in (\bullet t)\bullet : t' + 1 - t > 0$$

For disabled transitions, we first need to pick an insufficiently marked pre-place.

$$\forall t : t \text{ disabled in } m \sum_{p \in (\bullet t), m(p) < W(p,t)} p > 0$$

For the picked place, all pre-transitions need to be inserted.

$$\forall p \in P \forall t \in \bullet p : t + 1 - p > 0$$

Since the translation is more or less a re-formulation of the requirements of Def. 13, we can state without proof:

Theorem 2. *If the ILP returns a solution, the set of all transitions that have value 1 in this solution form a stubborn set with minimal number of enabled transitions. If the ILP is infeasible, T is the stubborn set with minimal number of enabled transitions.*

For stubborn sets in final Büchi states, the approach is analogous.

7.2 Enumeration of Up-Sets

The remaining problem is to enumerate reasonable up-sets. Input is a marking m and a formula φ. Output is a family $UP(m, \varphi)$ of up-sets. By Def. 13, we are only interested in up-sets which do not contain enabled transitions.

We follow the discussion in Sect. 6.1 and proceed by induction on the structure of the formula. For atomic propositions, up-sets depend on the particular nature of the propositions. In our example language, we have $UP(m, p \geq k) = \{\bullet p\}$ if $\bullet p$ does not contain enabled transitions, otherwise $UP(m, p \geq k) = \emptyset$. Likewise, $UP(m, p < k)$ is $\{p\bullet\}$ or $\emptyset$.

For the Boolean operations, we naturally derive $UP(m, \varphi_1 \wedge \varphi_2) = UP(m, \varphi_1)$ if $m \models \varphi_2$, $UP(m, \varphi_1 \wedge \varphi_2) = UP(m, \varphi_2)$ if $m \models \varphi_1$, $UP(m, \varphi_1 \wedge \varphi_2) = UP(m, \varphi_1) \cup UP(m, \varphi_2)$, else. Remember that $m \not\models \varphi_1 \wedge \varphi_2$ when we compute up-sets. For disjunction, we get $UP(m, \varphi_1 \vee \varphi_2) = \{M_1 \cup M_2 \mid M_1 \in UP(m, \varphi_1), M_2 \in UP(m, \varphi_1)\}$.

If c_φ denotes the number of elements in $UP(m, \varphi)$, we can give the following constraints for the values c_φ.

$$\begin{aligned} c_{p \geq k} &= 1 \\ c_{p < k} &= 1 \\ c_{\varphi_1 \wedge \varphi_2} &\leq c_{\varphi_1} + c_{\varphi_2} \\ c_{\varphi_1 \vee \varphi_2} &= c_{\varphi_1} \cdot c_{\varphi_2} \end{aligned}$$

In principle, this number can grow exponentially. We believe, however, that most properties occurring in practice will have a limited alternation between conjunction and disjunction, so the number should not be a problem.

The returned up-sets are the smallest ones we can exhibit without further knowledge of the net. If there is no up-set without enabled transitions, or there is one beyond the ones considered by the recursive procedure sketched above, we will not launch the ILP instance but return T as a stubborn set.

7.3 Experimental validation

Unfortunately, we do not have experimental results on nontrivial elementary properties yet. We can, however, provide results for the property $\mathbf{G}\,\varphi$. This property is violated if a state m is reachable where $m \not\models \varphi$. Hence, we can evaluate our approach, in particular the effect of minimizing the stubborn sets by enumeration of up-sets. We compare the new, ILP based approach to the existing implementation in our tool LoLA where an arbitrary up-set is picked, and an arbitrary insufficiently marked pre-place is chosen. We compare these two approaches using benchmark examples from the 2011 issue of the model checking contest [9]. Using the existing LoLA implementation for comparison is justified by the excellent overall performance of LoLA in that contest. The reachablity queries in the contest were all global reachability predicates which as such do not leave any invisible transition (which is not a problem for progressing formulas). For traditional stubborn set methods, the absence of invisible transitions would

Table 1. Comparing the original LoLA with the new ILP approach. All numbers are states explored for checking 20 tasks (fomulas) for each family and scale factor. After 5 million states we stopped the check (–).

model		states with LoLA			states with LoLA-ILP		
family	**scale**	**min**	**avg**	**max**	**min**	**avg**	**max**
FMS	2	11	436	1,230	11	56	203
FMS	5	53,929	147,760	308,435	76	241	665
FMS	10	–	–	–	121	625	1,970
FMS	50	–	–	–	481	9,586	39,410
FMS	100	–	–	–	931	35,524	153,710
FMS	200	–	–	–	1,831	136,527	607,310
FMS	500	–	–	–	4,531	832,534	3,768,110
Kanban	5	54	302,121	860,406	46	304	1,131
Kanban	10	–	–	–	266	2,498	12,832
Kanban	20	–	–	–	516	9,121	51,782
Kanban	50	–	–	–	1,266	52,932	322,232
Kanban	100	–	–	–	2,516	205,749	1,284,982
Kanban	200	–	–	–	5,016	804,110	–
Kanban	500	–	–	–	12,516	2,623,553	–
MAPK	8	132	1,775,353	–	93	32,701	93,948
MAPK	20	–	–	–	539,722	2,206,091	–

require them to produce an unreduced state space. For this reason, we do use any other tool in our experiments.

We checked three family of models: a flexible manufacturing system (FMS), Kanban (a benchmark from the SMART model checker [3]), and a system biological model (MAPK). Each family can be scaled by adjusting the size of the initial marking. The interested reader is referred to [9] for a detailed discussion of the models.

Of course, calculation of stubborn sets using ILP is much slower than classical methods where nondeterminism is resolved arbitrarily but without backtracking. In the experiments, the ILP version of LoLA needed about 50 to 100 times as long as the original LoLA implementation for calculating the same number of states. As the numbers show, this time is well invested, though. First, the investment of more expensive stubborn set calculation pays off by the smaller number of states to be explored. That is, the ILP version typically finishes earlier than the original LoLA. Second, the smaller state spaces increase the odds that a problem instance ca be solved within the given memory. This is actually much more important in the model checking area than run time.

8 Related Work and Conclusion

Most approaches on partial order reduction focus on reduction of the transition system under investigation. To our best knowledge, [8] is the closest approach that also takes care of the Büchi state when calculation stubborn sets. Basically, they relax the original invisibility requirement to those propositions that are ahead of the current Büchi state. In contrast, our approach requires invisibility only for one proposition at a time, and only for retarding formulas. We have to pay for this improvement with the inclusion of a whole up-set in the stubborn set whereas [8] needs to include only an enabled transition. This means that they may be able to find stubborn sets with only invisible enabled transitions more frequently. We address this issue by a more sophisticated calculation procedure where all the available nondeterminism is exploited for the sake of finding a good stubborn set.

This paper lacks experimental evaluation of the actual reduction power. We compensate this problem by collecting other, indirect evidence for the significance of our approach. First, our approach is applicable where other approaches generally fail. We tolerate partial violation of the stuttering invariance, we tolerate lack of invisible transitions w.r.t. the progressing formulas, and we tolerate search strategies where cycles in the product system cannot be found. Furthermore our technique is a proper generalization of [13] which has already shown its merit. That approach has been further explored in [10]. In future work, we have to study whether these ideas can be applied to simple linear time properties as well.

Determining an optimal stubborn set is a question that has received much attention [20,5,19]. With our translation to ILP, we exploit all the nondeterminism that is available in the particular closure operation we rely on, as well as most of the nondeterminism in the choice of an up-set. Experimental results are quite encouraging. Nevertheless, our minimality criterion (minimal number of enabled transitions in the stubborn set) is known not necessarily to be the best choice [20]. It optimizes the size of stubborn sets only locally but cannot guarantee minimal resulting product systems. Hence, optimal stubborn set calculation that takes care of up-sets remains an open question.

Acknowledgement. This work was partially funded by the DFG (German research foundation) in the project WS4Dsec in the priority program Reliably Secure Software Systems (SPP 1496).

References

1. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.
2. E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
3. G. Ciardo et al. The smart model checker. *http://www.cs.ucr.edu/ ciardo/SMART*.
4. D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data Knowl. Eng.*, 70(5):448–466, 2011.

5. J. Geldenhuys, H. Hansen, and A. Valmari. Exploring the scope for partial order reduction. In *ATVA*, volume 5799 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2009.
6. R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *3rd Israel Symp. on the Theory of Computing and Systems, IEEE 1995*, pages 130–140, 1995.
7. P. Godefroid and P. Wolper. A partial approach to model checking. *6th IEEE Symp. on Logic in Computer Science, Amsterdam*, pages 406–415, 1991.
8. I. Kokkarinen, D. Peled, and A. Valmari. Relaxed visibility enhances partial order reduction. *9th Int. Conf. Computer Aided Verification, Haifa, Israel, LNCS 1254*, pages 328–339, 1997.
9. F. Kordon et al. Report on the model checking contest at Petri Nets 2011. *LNCS ToPNoC*, 2012. (Accepted for publication in January 2012, more information provided on `http://sumo.lip6.fr/mcc.html`).
10. L.M. Krisensen and A. Valmari. Improved question-guided stubborn set methods for state properties. *Proc. 21th Int. Conf. Application and Theory of Petri nets*, pages 282–302, 2000.
11. D. Peled. All from one, one for all: on model–checking using representitives. *5th Int. Conf. Computer Aided Verification,Elounda, Greece, LNCS 697*, pages 409–423, 1993.
12. W. Reisig. *Elements Of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer-Verlag, September 1998.
13. K. Schmidt. Stubborn sets for standard properties. *Int. Conf. Application and Theory of Petri nets, LNCS 1639*, pages 46–65, 1999.
14. K. Schmidt. Stubborn sets for model checking the EF/AG fragment of CTL. *Fundam. Inform.*, 43(1-4):331–341, 2000.
15. C. Stahl, W. Reisig, and M. Krstic. Hazard detection in a GALS wrapper: A case study. In *ACSD 2005*, pages 234–243. IEEE Computer Society, 2005.
16. A. Valmari. Error detetction by reduced reachability graph generation. *9th European Workshop on Application and Theory of Petri nets, Venice, Italy*, pages 95–112, 1988.
17. A. Valmari. A stubborn attack on state explosion. *Formal Methods in System Design 1*, pages 297–322, 1992.
18. A. Valmari. Stubborn set methods for process algebras. *Workshop on Partial Order Methods in Verification, Princeton*, pages 192–210, 1996.
19. A. Valmari and H. Hansen. Can stubborn sets be optimal? In *Petri Nets 2010*, LNCS 6128, pages 43–62. Springer, 2010.
20. K. Varpaaniemi. *On the stubborn set method in reduced state space generation*. PhD thesis, Helsinki University of Technology, 1998.
21. K. Wolf. Generating Petri net state spaces. In *ICATPN 2007*, LNCS 4546, pages 29–42. Springer, 2007.