Stubborn Set Intuition Explained

Antti Valmari and Henri Hansen

Department of Mathematics, Tampere University of Technology
P.O. Box 553, FI–33101 Tampere, Finland

Abstract. This study focuses on the differences between stubborn sets and other partial order methods. First a major problem with step graphs is pointed out with an example. Then the deadlock-preserving stubborn set method is compared to the deadlock-preserving ample set and persistent set methods. Next, conditions are discussed whose purpose is to ensure that the reduced state space preserves the ordering of visible transitions, that is, transitions that may change the truth values of the propositions that the formula under verification has been built from. Finally solutions to the ignoring problem are analysed both when the purpose is to preserve only safety properties and when also liveness properties are of interest.

1 Introduction

Ample sets [10, 11, 1], *persistent sets* [5, 6], and *stubborn sets* [15, 19] are methods for constructing reduced state spaces. In each found state, they compute a subset of transitions and only fire the enabled transitions in it to find more states. We call this subset an *aps set*.

The choice of aps sets depends on the properties under verification. Attempts to obtain good reduction for various classes of properties have led to the development of many different methods. Even when addressing the same class of properties, stubborn set methods often differ from other aps set methods. The present study focuses on these differences. The goal is to explain the intuition behind the choices made in stubborn set methods.

To get a concrete starting point, Section 2 presents a simple (and non-optimal) definition of stubborn sets for Petri nets that suffices for preserving all reachable deadlocks. The section also introduces the "$\overline{\rightsquigarrow}_M$"-relation that underlies many algorithms for computing stubborn sets, and sketches one good algorithm. This relation and algorithm are one of the major differences between stubborn set and other aps set methods. The section also contains a small new result, namely an example showing that always choosing a singleton stubborn set if one is available does not necessarily guarantee best reduction results.

With Petri nets, it might seem natural to fire sets of transitions called steps, instead of individual transitions. Section 3 discusses why this is not necessarily a good idea. Ample and persistent sets are compared to stubborn sets in Section 4, in the context of deadlock-preservation. Furthermore, the difference between weak and strong stubborn sets is explained. The verification of many properties

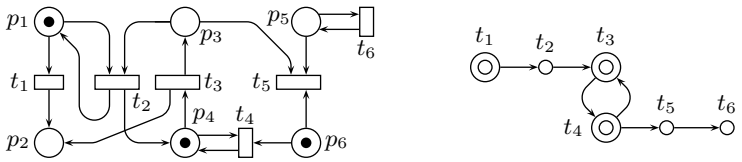

Fig. 1. A marked Petri net and its "$\rightsquigarrow_M$"-graph, with $p_{t_5} = p_5$.

relies on a distinction between *visible* and *invisible* transitions. This distinction is introduced in Section 5. Its ample and stubborn set versions are compared to each other.

Because of the so-called *ignoring problem*, deadlock-preserving aps set methods fail to preserve most other classes of properties. For many classes, it suffices to solve the ignoring problem in the terminal strong components of the reduced state space. To this end, two slightly different methods have been suggested. Section 6 first introduces them, and then presents and proves correct a novel idea that largely combines their best features.

The above-mentioned solutions to the ignoring problem do not suffice for so-called liveness properties. Section 7 discusses the stubborn set and ample set methods for liveness. A drawback in the most widely known implementation of the methods is pointed out. Section 8 concludes the study.

2 The Basic Idea of Stubborn Sets

In this section we illustrate the basic idea of stubborn sets and of one good algorithm for computing them.

We use T to denote the set of (all) transitions of a Petri net. Let M be a marking. The set of *enabled* transitions in M is denoted with $\mathsf{en}(M)$ and defined as $\{t \in T \mid M\,[t\rangle\}$. A *deadlock* is a marking that has no enabled transitions.

In Figure 1 left, only firing t_1 in the initial marking leads to the loss of the deadlock that is reached by firing $t_3t_2t_3t_1$. To find a subset of transitions that cannot lead to such losses, we first define a marking-dependent relation "$\rightsquigarrow_M$" between Petri net transitions.

PNd If $\neg(M\,[t\rangle)$, then choose $p_t \in {\bullet t}$ such that $M(p_t) < W(p_t, t)$ and declare $t \rightsquigarrow_M t'$ for every $t' \in {\bullet p_t}$ except t itself. (If many such p_t are available, only one is chosen. The correctness of what follows does not depend on the choice.)

PNe If $M\,[t\rangle$, then declare $t \rightsquigarrow_M t'$ for every $t' \in ({\bullet t})\bullet$ except t itself.

On the right in Figure 1, enabled transitions are shown with double circles, disabled transitions with single circles, and the "$\rightsquigarrow_M$"-relation with arrows. For instance, t_4 is enabled, ${\bullet t_4} = \{p_4, p_6\}$, and $\{p_4, p_6\}\bullet = \{t_3, t_4, t_5\}$, so **PNe** declares $t_4 \rightsquigarrow_M t_3$ and $t_4 \rightsquigarrow_M t_5$. Regarding t_5, **PNd** allows choosing $p_{t_5} = p_3$ or $p_{t_5} = p_5$. In the example p_5 was chosen, spanning the arrow $t_5 \rightsquigarrow_M t_6$.

Consider any *"$\rightsquigarrow_M$"-closed* set T_M of transitions, that is, for every t and t', if $t \in T_M$ and $t \rightsquigarrow_M t'$, then also $t' \in T_M$. Assume that $t \in T_M$, $t_i \notin T_M$ for $1 \leq i \leq n$, and $M\ [t_1 \cdots t_n\rangle\ M'$. **PNd** guarantees that if t is disabled in M, then t is disabled also in M'. This is because every transition that may increase the number of tokens in p_t is in T_M. **PNe** guarantees that if t is enabled in M, then there is M'' such that $M'\ [t\rangle\ M''$ and $M\ [tt_1 \cdots t_n\rangle\ M''$. This is because t does not consume tokens from the same places as $t_1 \cdots t_n$.

Let $\hat{M}$ be the initial marking of a Petri net. Let $\mathsf{stubb}(M)$ be a function that, for any marking M that is not a deadlock, returns an "$\rightsquigarrow_M$"-closed set of transitions that contains at least one enabled transition. This set is called *stubborn*. If M is a deadlock, then it does not matter what $\mathsf{stubb}(M)$ returns. Let the *reduced state space* be the triple $(S_\mathsf{r}, \Delta_\mathsf{r}, \hat{M})$, where S_r and Δ_r are the smallest sets such that (1) $\hat{M} \in S_\mathsf{r}$ and (2) if $M \in S_\mathsf{r}$, $t \in \mathsf{stubb}(M)$, and $M\ [t\rangle\ M'$, then $M' \in S_\mathsf{r}$ and $(M, t, M') \in \Delta_\mathsf{r}$. It can be constructed like the ordinary state space, except that only the enabled transitions in $\mathsf{stubb}(M)$ are fired in each constructed marking M. We have the following theorem.

Theorem 1. *The set S_r contains all deadlocks that are reachable from $\hat{M}$.*

Proof. The proof proceeds by induction. Let $M \in S_\mathsf{r}$ and $M\ [t_1 \cdots t_n\rangle\ M_\mathsf{d}$, where M_d is a deadlock. If $n = 0$, then $M_\mathsf{d} = M \in S_\mathsf{r}$.

If $n > 0$, then $M\ [t_1\rangle$. So M is not a deadlock and $\mathsf{stubb}(M)$ contains an enabled transition t. If none of t_i is in $\mathsf{stubb}(M)$, then **PNe** implies that t is enabled at M_d, contradicting the assumption that M_d is a deadlock. So there is i such that $t_i \in \mathsf{stubb}(M)$ but $t_j \notin \mathsf{stubb}(M)$ for $1 \leq j < i$. Let M_{i-1} and M_i be the markings such that $M\ [t_1 \cdots t_{i-1}\rangle\ M_{i-1}\ [t_i\rangle\ M_i\ [t_{i+1} \cdots t_n\rangle\ M_\mathsf{d}$. **PNd** implies that $t_i \in \mathsf{en}(M)$, because otherwise t_i would be disabled in M_{i-1}. So **PNe** implies $M\ [t_i t_1 \cdots t_{i-1}\rangle\ M_i\ [t_{i+1} \cdots t_n\rangle\ M_\mathsf{d}$. Let M' be the marking such that $M\ [t_i\rangle\ M'$. Then $M' \in S_\mathsf{r}$ and there is the path $M'\ [t_1 \cdots t_{i-1} t_{i+1} \cdots t_n\rangle\ M_\mathsf{d}$ of length $n - 1$ from M' to M_d. By induction, $M_\mathsf{d} \in S_\mathsf{r}$. □

The next question is how to compute stubborn sets. Clearly only the enabled transitions in $\mathsf{stubb}(M)$ affect the reduced state space. Therefore, we define $T_1 \sqsubseteq_M T_2$ if and only if $T_1 \cap \mathsf{en}(M) \subseteq T_2 \cap \mathsf{en}(M)$. If $\mathsf{stubb}_1(M) \sqsubseteq_M \mathsf{stubb}_2(M)$ for every reachable marking M, then stubb_1 yields a smaller or the same reduced state space as stubb_2. So we would like to use $\sqsubseteq_M$-minimal stubborn sets.

Each "$\rightsquigarrow_M$"-relation spans a directed graph $(T, \text{"}\rightsquigarrow_M\text{"})$ as illustrated in Figure 1 right. We call it the *"$\rightsquigarrow_M$"-graph*. Let C be a strong component of the "$\rightsquigarrow_M$"-graph such that it contains an enabled transition, but no other strong component that is reachable from C contains enabled transitions. In Figure 1, $C = \{t_3, t_4\}$ is such a strong component. Let C' be the set of all transitions that are reachable from C. In Figure 1, $C' = \{t_3, t_4, t_5, t_6\}$. Then C' is an $\sqsubseteq_M$-minimal "$\rightsquigarrow_M$"-closed set that contains an enabled transition. That is, we can choose $\mathsf{stubb}(M) = C'$.

A fast algorithm that is based on this idea was presented in [15, 19, 23], among others. It uses Tarjan's strong component algorithm [14] (see [3] for an improved version). It has been implemented in the ASSET tool [21] (although not for Petri

nets). Its running time is linear in the size of the part of the "$\rightsquigarrow_M$"-graph that it investigates. For instance, if it happens to start at t_2 in Figure 1, then it does not investigate t_1 and its output arrow. Although in this example the resulting savings are small, they are often significant.

The algorithm performs one or more depth-first searches in the "$\rightsquigarrow_M$"-graph, until a search finds an enabled transition or all transitions have been tried. The description above leaves open the order in which transitions are used as the starting points of the searches. The same holds on the order in which the output arrows of each transition are investigated. For instance, when in t_4 in Figure 1, the algorithm may follow the arrow $t_4 \rightsquigarrow t_5$ before or after the arrow $t_4 \rightsquigarrow t_3$. Therefore, the result of the algorithm may depend on implementation details. Furthermore, it may also depend on the choice of p_t if there are more than one alternatives. This is why we sometimes say that the method *may* produce some result, instead of saying that it *does* produce it.

The conditions **PNd** and **PNe** are not the best possible. For instance, $t \rightsquigarrow_M t'$ need not be declared in **PNd**, if $W(p_t, t') \geq W(t', p_t)$. Extreme optimization of the "$\rightsquigarrow_M$"-relation yields very complicated conditions, as can be seen in [15, 19]. A similar claim holds also with formalisms other than Petri nets. For this reason, and also to make the theory less dependent on the formalism used for modelling systems, aps set methods are usually developed in terms of more abstract conditions than **PNd** and **PNe**. We will do so in Section 4.

To analyse more properties than just deadlocks, additional conditions on the choice of stubborn sets are needed. Many of them will be discussed in Sections 5, 6, and 7.

Until the end of Section 5 it will be obvious that if $\mathsf{stubb}_1(M) \sqsubseteq_M \mathsf{stubb}_2(M)$, then $\mathsf{stubb}_1(M)$ never yields worse but may yield better reduction results than $\mathsf{stubb}_2(M)$. In Sections 6 and 7, the choices of $\mathsf{stubb}(M)$ with different M may interfere with each other, making the issue less trivial.

Even in the present section, if $\mathsf{stubb}_1(M) \not\sqsubseteq_M \mathsf{stubb}_2(M)$ and $\mathsf{stubb}_2(M) \not\sqsubseteq_M \mathsf{stubb}_1(M)$, then it is not obvious which one to choose. It was pointed out already in [16] that choosing the smallest number of enabled transitions does not necessarily guarantee best reduction results. In the remainder of this section we demonstrate that always favouring a set with precisely one enabled transition does not guarantee a minimal result. This strengthened observation is new.

A Petri net is *1-safe* if and only if no place contains more than one token in any reachable marking. For simplicity, we express a marking of a 1-safe Petri net by listing the marked places within { and }.

Consider the 1-safe Petri net in Figure 2. Initially the only possibility is to fire t_1 and t_2, yielding $\{2, 8\}$ and $\{3, 8\}$. In $\{2, 8\}$, both $\{t_3, t_4\}$ and $\{t_9\}$ are stubborn. In $\{3, 8\}$, both $\{t_5, t_6\}$ and $\{t_9\}$ are stubborn. We now show that $\{t_3, t_4\}$ and $\{t_5, t_6\}$ yield better reduction than $\{t_9\}$.

If $\{t_3, t_4\}$ is chosen in $\{2, 8\}$ or $\{t_5, t_6\}$ is chosen in $\{3, 8\}$, then $\{4, 8\}$ and $\{5, 8\}$ are obtained. From them, $\{t_7\}$ and $\{t_8\}$ yield $\{6, 8\}$, from which $\{t_9\}$ leads to $\{6, 9\}$ which is a deadlock. Altogether seven markings and nine edges are constructed.

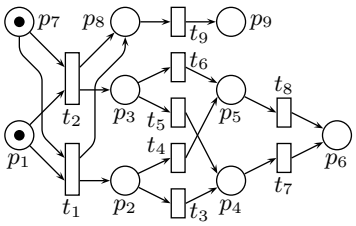

Fig. 2. A stubborn set with one enabled transition is not always the best choice.

If $\{t_9\}$ is chosen in $\{2, 8\}$ and $\{3, 8\}$, then $\{2, 9\}$ and $\{3, 9\}$ are obtained. Then $\{t_3, t_4\}$ or $\{t_5, t_6\}$ yields $\{4, 9\}$ and $\{5, 9\}$, from which $\{t_7\}$ and $\{t_8\}$ produce $\{6, 9\}$. Altogether eight markings and ten edges are constructed.

3 Why Not Steps?

Before comparing aps set methods to each other, in this section we compare them to step graphs. For simplicity, we restrict ourselves to executions that lead to deadlocks. That is, the goal is to find all reachable deadlocks and for each of them at least one path that leads to it.

A *step* is any nonempty subset $\{t_1, \ldots, t_n\}$ of Petri net transitions. It is *enabled* at M if and only if $M(p) \geq \sum_{i=1}^{n} W(p, t_i)$ for every place p. Then there is M' such that $M\ [\pi\rangle\ M'$ for every permutation π of $t_1 \cdots t_n$. The idea of a *step graph* is to fire steps instead of individual transitions. Unlike the traditional state space, the order of firing of the transitions within the step is not represented, and intermediate markings between the firings of two successive transitions in π are not constructed. This is expected to yield a memory-efficient representation of the behaviour of the Petri net.

To maximize the savings, the steps should be as big as possible. Unfortunately, the following example shows that only firing maximal steps is not correct. By firing $t_3t_2t_3t_1$ in Figure 1, a deadlock is reached where $M(p_2) = 3$. The maximal steps in the initial marking are $\{t_1, t_3\}$ and $\{t_1, t_4\}$. If only they are fired in the initial marking, no deadlock with $M(p_2) > 2$ is reached.

This problem can be fixed by also firing a sufficient collection of non-maximal steps. If $\{t_1, t_3\}$, $\{t_1, t_4\}$, $\{t_3\}$, and $\{t_4\}$ are fired in the initial marking of our example, the marking M is saved that satisfies $\hat{M}\ [t_1\rangle\ M$. There is, however, another problem that arises even when it suffices to fire only maximal steps. We will now discuss it.

Consider the Petri net that consists of the black places, transitions, and arcs in Figure 3 left. It models a system of n concurrent processes. It has $n!2^n$ different executions, yielding a state space with 3^n markings and $2n3^{n-1}$ edges. Its initial marking has 2^n different steps of size n, consisting of one transition from each process. They yield a step graph with $2^n + 1$ markings and 2^n edges.

Any reasonable implementation of any aps set method investigates one process at a time in this example. That is, the implementation picks some i such

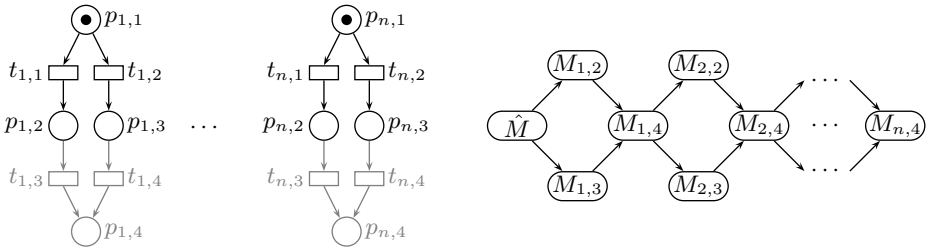

Fig. 3. An example of firing steps vs. aps sets.

that $M(p_{i,1}) = 1$, and chooses $\mathsf{aps}(M) = \{t_{i,1}, t_{i,2}\}$. If there is no such i, then $\mathsf{aps}(M) = \emptyset$. This yields $1+2+4+8+\ldots+2^n = 2^{n+1}-1$ markings and $2^{n+1}-2$ edges.

We see that both methods yield a significant saving over the full state space, and step graphs yield approximately 50 % additional saving over aps sets. Step graphs construct strictly as few markings and edges as necessary in this example.

Assume now that the grey places, transitions, and arcs are added. The step graph now has $2^n + 2$ markings and 2^{n+1} edges.

Aps sets may yield many different results depending on what $\mathsf{aps}(M)$ returns for each M. Assume that the algorithm in Section 2 is used and transitions are tried as the starting points of depth-first searches in the order $t_{1,1}$, $t_{1,2}$, $t_{1,3}$, $t_{1,4}$, $t_{2,1}$, $t_{2,2}$, ... Then $\mathsf{aps}(M)$ is either $\{t_{i,1}, t_{i,2}\}$, $\{t_{i,3}\}$, or $\{t_{i,4}\}$, where i is the smallest index such that either $M(p_{i,1}) = 1$, $M(p_{i,2}) = 1$, or $M(p_{i,3}) = 1$. (If there is no such i, then $\mathsf{aps}(M) = \emptyset$.) In that case, the reduced state space shown at right in Figure 3 is obtained. In $M_{i,j}$, $M(p_{k,4}) = 1$ for $1 \leq k < i$, $M(p_{i,j}) = 1$, $M(p_{k,1}) = 1$ for $i < k \leq n$, and the remaining places are empty. That is, only $3n+1$ markings and $4n$ edges are constructed. This is tremendously better than the result with step graphs.

There is no guarantee that aps sets yield this nice result. If transitions are tried in the order $t_{1,1}$, $t_{2,1}$, ..., $t_{n,1}$, $t_{1,2}$, $t_{2,2}$, ..., then $3 \cdot 2^n - 2$ markings and $2^{n+2} - 4$ edges are obtained.

The point is that in this example, it is *guaranteed* that step graphs do not yield a good result, while aps sets *may* yield a very good result.

Another issue worth noticing in this example is that when aps sets failed to reduce well, they only generated approximately three times the markings and twice the edges that the step graphs generated. This is because where steps avoided many intermediate markings, aps sets investigated only one path through them and thus only generated a small number of them. For this reason, even when aps sets lose to step graphs, they tend not to lose much.

This example brings forward a problem with comparing different methods. Most of the methods in this research field are nondeterministic in the same sense as the algorithm in Section 2. Therefore, the results of a verification experiment may depend on, for instance, the order in which transitions are listed in the input of a tool. Above, the order $t_{1,1}$, $t_{2,1}$, ... gave dramatically worse results

than the order $t_{1,1}$, $t_{1,2}$, ... When comparing verification methods or tools, it might be a good idea to repeat experiments with transitions in some other order.

4 Deadlocks with Ample vs. Persistent vs. Stubborn Sets

In this section we relate the ample set, persistent set, strong stubborn set, and weak stubborn set methods to each other when the goal is to preserve all deadlocks. Details of each method vary in the literature. We use the variant of ample sets described in [1], persistent sets in [6], and stubborn sets in [19]. These versions of the methods are mature and widely used.

We will use familiar or obvious notation for states, transitions, and so forth. A *set of states* is typically denoted with S, a *set of transitions* with T, and an *initial state* with $\hat{s}$. Transitions refer to *structural transitions* such as Petri net transitions or atomic statements of a program. Transition t is *deterministic*, if and only if for every s, t, s_1, and s_2, $s \xrightarrow{t} s_1$ and $s \xrightarrow{t} s_2$ imply $s_1 = s_2$.

Ample, persistent, and stubborn set methods compute an aps set $\mathsf{aps}(s)$ in each state s that they encounter. They construct a reduced state space by only firing the enabled transitions in each $\mathsf{aps}(s)$. It is the triple $(S_\mathsf{r}, \Delta_\mathsf{r}, \hat{s})$, where S_r and Δ_r are the smallest sets such that (1) $\hat{s} \in S_\mathsf{r}$ and (2) if $s \in S_\mathsf{r}$, $t \in \mathsf{aps}(s)$, and $s \xrightarrow{t} s'$, then $s' \in S_\mathsf{r}$ and $(s, t, s') \in \Delta_\mathsf{r}$. Obviously $\hat{s} \in S_\mathsf{r} \subseteq S$ and $\Delta_\mathsf{r} \subseteq \Delta$.

The **ample set method** relies on the notion of *independence* between transitions. It is usually defined as any binary relation on transitions that has the following property:

Independence. If transitions t_1 and t_2 are independent of each other, $s \xrightarrow{t_1} s_1$, and $s \xrightarrow{t_2} s_2$, then there is s' such that $s_1 \xrightarrow{t_2} s'$ and $s_2 \xrightarrow{t_1} s'$.

Independence is not defined as the largest relation with this property, because it may be difficult to find out whether the property holds for some pair of transitions. In such a situation, the pair may be declared as dependent. Doing so does not jeopardize the correctness of the reduced state space, but may increase its size. This issue is similar to the use of non-optimal "$\rightsquigarrow_M$"-relations in Section 2.

Obviously transitions that do not access any variable (or Petri net place) in common can be declared as independent. (Here also the program counter or local state of a process is treated as a variable.) Two transitions that both increment the value of a variable by 42 without testing its value in their enabling conditions can be declared as independent, if they do not access other variables in common. A similar claim holds if they both assign 63 to the variable. Reading from a fifo queue and writing to it can be declared as independent, as can two transitions that are never simultaneously enabled.

An *ample set for deadlocks* in state s_0 is any subset of transitions that are enabled at s_0 that satisfies the following two conditions:

C0 If $\mathsf{en}(s_0) \neq \emptyset$, then $\mathsf{ample}(s_0) \neq \emptyset$.

C1 If $s_0 \xrightarrow{t_1 \cdots t_n}$ and none of $t_1, \ldots, t_n$ is in $\mathsf{ample}(s_0)$, then each one of $t_1, \ldots, t_n$ is independent of all transitions in $\mathsf{ample}(s_0)$.

We show next that every deadlock of the full state space is present also in the reduced state space.

Theorem 2. *Assume that transitions are deterministic, $s \in S_{\mathsf{r}}$, s_{d} is a deadlock, and $s \xrightarrow{t_1 \cdots t_n} s_{\mathsf{d}}$ in the full state space. If **C0** and **C1** are obeyed, then there is a permutation $t'_1 \cdots t'_n$ of $t_1 \cdots t_n$ such that $s \xrightarrow{t'_1 \cdots t'_n} s_{\mathsf{d}}$ in the reduced state space.*

Proof. We only present the parts where the proof differs from the proof of Theorem 1. If $n > 0$, then $\mathsf{ample}(s)$ contains an enabled transition t by **C0** and $\mathsf{ample}(s) \subseteq \mathsf{en}(s)$. If none of $t_1, \ldots, t_n$ is in $\mathsf{ample}(s)$, then $s_{\mathsf{d}} \xrightarrow{t}$ by **C1**, contradicting the assumption that s_{d} is a deadlock. So there is a smallest i such that $t_i \in \mathsf{ample}(s)$. Let s_{i-1} and s_i be the states such that $s \xrightarrow{t_1 \cdots t_{i-1}} s_{i-1} \xrightarrow{t_i} s_i$. Since $\mathsf{ample}(s) \subseteq \mathsf{en}(s)$, there is s' such that $s \xrightarrow{t_i} s'$. By **C1**, applying independence $i-1$ times, there is s'_i such that $s' \xrightarrow{t_1 \cdots t_{i-1}} s'_i$ and $s_{i-1} \xrightarrow{t_i} s'_i$. Because transitions are deterministic, $s'_i = s_i$. As a consequence, $s \xrightarrow{t_i} s' \xrightarrow{t_1 \cdots t_{i-1}} s_i \xrightarrow{t_{i+1} \cdots t_n} s_{\mathsf{d}}$. □

Strong stubborn sets are defined such that they may contain both enabled and disabled transitions. Deadlock-preserving strong stubborn sets satisfy the following three conditions. **D0** is essentially the same as **C0**. **D1** and **D2** will be motivated and related to **C1** after the definition.

D0 If $\mathsf{en}(s_0) \neq \emptyset$, then $\mathsf{stubb}(s_0) \cap \mathsf{en}(s_0) \neq \emptyset$.

D1 If $t \in \mathsf{stubb}(s_0)$, $t_i \notin \mathsf{stubb}(s_0)$ for $1 \leq i \leq n$, and $s_0 \xrightarrow{t_1 \cdots t_n t} s'_n$, then $s_0 \xrightarrow{t t_1 \cdots t_n} s'_n$.

D2 If $t \in \mathsf{stubb}(s_0)$, $t_i \notin \mathsf{stubb}(s_0)$ for $1 \leq i \leq n$, $s_0 \xrightarrow{t_1 \cdots t_n} s_n$, and $s_0 \xrightarrow{t}$, then $s_n \xrightarrow{t}$.

This formulation was suggested by Marko Rauhamaa [12]. The most important reason for its use is that **D1** works well even if transitions are not necessarily deterministic. For deadlocks, also **D2** can be used as such. This is important for applying stubborn sets to process algebras, please see, e.g., [18, 23]. In the proof of Theorem 2, the assumption that transitions are deterministic was explicitly used. Already the definition of independence relies on determinism. This issue makes ample and persistent set theories difficult to apply to process algebras.

Second, **D1** can be used as such and **D2** with a small change in the definition of weak stubborn sets towards the end of this section.

Third, **D1** and **D2** are slightly easier to use in proofs than **C1**. Let $s = s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} s_n = s_{\mathsf{d}}$. **D0** and **D2** yield an i such that $t_i \in \mathsf{stubb}(s)$ and $t_j \notin \mathsf{stubb}(s)$ for $1 \leq j < i$. Then the existence of s' such that $s \xrightarrow{t_i} s' \xrightarrow{t_1 \cdots t_{i-1}} s_i$ is immediate by **D1**. This last piece of reasoning is repeated frequently in stubborn set theory, so it is handy that **D1** gives it as a ready-made step. We have proven the following generalization of Theorem 2.

Theorem 3. *Theorem 2 remains valid, if **D0**, **D1**, and **D2** replace **C0** and **C1**. Then transitions need not be determistic.*

This is a generalization, because it applies to also nondeterministic transitions, and because, as will be seen in Theorem 5, in the case of deterministic transitions **C0** and **C1** imply **D0**, **D1**, and **D2**.

In the case of deterministic transitions, **D1** and **D2** have the following equivalent formulation:

Dd If $t \in \mathsf{stubb}(s_0)$, $\neg(s_0 \xrightarrow{t})$, $t_i \notin \mathsf{stubb}(s_0)$ for $1 \leq i \leq n$, and $s_0 \xrightarrow{t_1 \cdots t_n} s_n$, then $\neg(s_n \xrightarrow{t})$.

De If $t \in \mathsf{stubb}(s_0)$, $s_0 \xrightarrow{t} s'_0$, $t_i \notin \mathsf{stubb}(s_0)$ for $1 \leq i \leq n$, and $s_0 \xrightarrow{t_1 \cdots t_n} s_n$, then there is s'_n such that $s_n \xrightarrow{t} s'_n$ and $s'_0 \xrightarrow{t_1 \cdots t_n} s'_n$.

Dd says that disabled transitions in the stubborn set remain disabled, while outside transitions occur. **De** says that enabled transitions in the stubborn set commute with sequences of outside transitions. It is immediately obvious that **PNd** and **PNe** imply **Dd** and **De**. Let us show that for deterministic transitions, this formulation indeed is equivalent to **D1** and **D2**.

Theorem 4. *If transitions are deterministic, then* ***D1*** $\wedge$ ***D2*** *is equivalent to* ***Dd*** $\wedge$ ***De****.*

Proof. Assume first that **D1** and **D2** hold. Then **Dd** follows immediately from **D1**. If $s_0 \xrightarrow{t} s'_0$ and $s_0 \xrightarrow{t_1 \cdots t_n} s_n$, then **D2** yields an s'_n such that $s_n \xrightarrow{t} s'_n$, after which **D1** yields an s''_0 such that $s_0 \xrightarrow{t} s''_0 \xrightarrow{t_1 \cdots t_n} s'_n$. Because transitions are deterministic, $s''_0 = s'_0$, so **De** is obtained.

Assume now that **Dd** and **De** hold. Then **D2** follows immediately from **De**. If $s_0 \xrightarrow{t_1 \cdots t_n} s_n \xrightarrow{t} s'_n$, then **Dd** yields an s'_0 such that $s_0 \xrightarrow{t} s'_0$, after which **De** yields an s''_n such that $s'_0 \xrightarrow{t_1 \cdots t_n} s''_n$ and $s_n \xrightarrow{t} s''_n$. Because transitions are deterministic, $s''_n = s'_n$, so **D1** is obtained. □

Similarly to the "$\rightsquigarrow_M$"-relation in Section 2, "$\rightsquigarrow_s$"-relations can be defined for Petri nets and other formalisms such that they guarantee **D1** and **D2**. Please see e.g., [19, 22, 23] for more information. This means that the stubborn set construction algorithm in Section 2 can be applied to many formalisms. Indeed, its implementation in ASSET is unaware of the formalism. It only has access to the "$\rightsquigarrow_s$"-relation and to the enabling status of each transition.

It would not be easy to describe this algorithm without allowing disabled transitions in the aps set. Indeed, instead of this algorithm, publications on ample and persistent sets suggest straightforward algorithms that test whether some obviously "$\rightsquigarrow_s$"-closed set is available and if not, revert to the set of all enabled transitions. This means that they waste reduction potential. The running time is not an important issue here, because, as experiments with ASSET have demonstrated [20, 21, 23], the algorithm is very fast.

The first publications on stubborn sets (such as [15]) used formalism-specific conditions resembling **PNd** and **PNe** instead of abstract conditions such as **D1** and **D2**.

It is now easy to show that every ample set is strongly stubborn.

Theorem 5. *Assume that transitions are deterministic,* $\mathsf{ample}(s_0) \subseteq \mathsf{en}(s_0)$*, and* $\mathsf{ample}(s_0)$ *satisfies **C0** and **C1**. Then* $\mathsf{ample}(s_0)$ *satisfies **D0**, **D1**, and **D2**.*

Proof. Clearly **C0** implies **D0**. **Dd** follows trivially from $\mathsf{ample}(s_0) \subseteq \mathsf{en}(s_0)$, and **De** follows immediately from **C1**. Now Theorem 4 gives the claim. □

The opposite does not hold, because it may be that $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t} s_3$, $s_0 \xrightarrow{t} s_2 \xrightarrow{t_1} s_3$, and $s_2 \xrightarrow{t} s_4$, and there are no other edges. Clearly $\{t\}$ satisfies **D0**, **D1**, and **D2** in s_0, but not **C1**, because t is not independent of t_1 because of s_2.

To relate strong stubborn sets to persistent sets, the following theorem is useful.

Theorem 6. *Let* s_0 *be a state and* $\mathsf{stubb}(s_0)$ *be a set of transitions. If* $\mathsf{stubb}(s_0)$ *obeys **D0**, **D1**, and **D2** in* s_0*, then also* $\mathsf{stubb}(s_0) \cap \mathsf{en}(s_0)$ *obeys them in* s_0*.*

Proof. That $\mathsf{stubb}(s_0) \cap \mathsf{en}(s_0)$ obeys **D0** is immediate from **D0** for $\mathsf{stubb}(s_0)$.

Assume that $s_0 \xrightarrow{t_1 \cdots t_n}$, where $t_i \notin \mathsf{stubb}(s_0) \cap \mathsf{en}(s_0)$ for $1 \leq i \leq n$. We prove that no t_i is in $\mathsf{stubb}(s_0)$. To derive a contradiction, let i be the smallest such that $t_i \in \mathsf{stubb}(s_0)$. So $t_i \in \mathsf{stubb}(s_0)$, $t_i \notin \mathsf{en}(s_0)$, $s_0 \xrightarrow{t_1 \cdots t_{i-1} t_i}$, and $t_j \notin \mathsf{stubb}(s_0)$ for $1 \leq j < i$. This contradicts **D1** for $\mathsf{stubb}(s_0)$.

If the if-part of **D1** holds for $\mathsf{stubb}(s_0) \cap \mathsf{en}(s_0)$, then by the above, the if-part of **D1** holds also for $\mathsf{stubb}(s_0)$. So the then-part for $\mathsf{stubb}(s_0)$ holds, which is the same as the then-part for $\mathsf{stubb}(s_0) \cap \mathsf{en}(s_0)$. Similar reasoning applies to **D2**. □

Persistent sets also assume that transitions are deterministic. They rely on *independence in a state*. If t and t' are independent in s, then the following hold [6, Def. 3.17]:

1. If $s \xrightarrow{t}$ and $s \xrightarrow{t'}$, then there is s' such that $s \xrightarrow{tt'} s'$ and $s \xrightarrow{t't} s'$.
2. If $s \xrightarrow{tt'}$, then $s \xrightarrow{t'}$.
3. If $s \xrightarrow{t't}$, then $s \xrightarrow{t}$.

A set $\mathsf{pers}(s_0)$ is *persistent* in s_0 if and only if $\mathsf{pers}(s_0) \subseteq \mathsf{en}(s_0)$ and for every $t_1, \ldots, t_n$ and $s_1, \ldots, s_n$ such that $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} s_n$ and $t_i \notin \mathsf{pers}(s_0)$ for $1 \leq i \leq n$, it holds that every element of $\mathsf{pers}(s_0)$ is independent of t_i in s_{i-1} [6, Def. 4.1].

It is worth noticing that the concept of persistency would not change if items 2 and 3 were removed from the definition of independence in a state. Let $t \in \mathsf{pers}(s_0)$, and let s'_0 be such that $s_0 \xrightarrow{t} s'_0$. Repeated application of item 1 yields $s'_1, \ldots, s'_n$ such that $s'_0 \xrightarrow{t_1} s'_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} s'_n$ and $s_i \xrightarrow{t} s'_i$ for $1 \leq i \leq n$. Because for $1 \leq i \leq n$, both t and t_i are enabled in s_{i-1}, the then-parts of items 2 and 3 hold, and thus the items as a whole hold. That is, items 2 and 3 can be proven for the states s_{i-1}, so they need not be assumed. It seems plausible that items 2 and 3 were originally adopted by analogy to the independence relation in Mazurkiewicz traces [9].

The next theorem, from [24, Lemma 4.14], says that persistent sets are equivalent to strong stubborn sets restricted to deterministic transitions.

Theorem 7. *Assume that transitions are deterministic. Every nonempty persistent set satisfies **D0**, **D1**, and **D2**. If a set satisfies **D1** and **D2**, then its set of enabled transitions is persistent.*

Proof. Persistency immediately implies **De**. It also trivially implies **Dd** because $\mathsf{pers}(s_0) \subseteq \mathsf{en}(s_0)$. These yield **D1** and **D2** by Theorem 4. If a persistent set is nonempty, then it trivially satisfies **D0**.

Assume that $\mathsf{stubb}(s_0)$ satisfies **D1** and **D2**. Let $\mathsf{pers}(s_0) = \mathsf{stubb}(s_0) \cap \mathsf{en}(s_0)$. By Theorems 6 and 4, $\mathsf{pers}(s_0)$ satisfies **De**. Let $t \in \mathsf{pers}(s_0)$, $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} s_n$, and $t_i \notin \mathsf{pers}(s_0)$ for $1 \le i \le n$. **De** implies $s'_0 \xrightarrow{t_1 \cdots t_n} s'_n$. Let $s'_1, \ldots, s'_{n-1}$ be the states such that $s'_0 \xrightarrow{t_1} s'_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} s'_n$. Let $1 \le i \le n$. By giving **De** $t_1 \cdots t_i$ instead of $t_1 \cdots t_n$ we see that **De** implies $s_i \xrightarrow{t} s'_i$ for $1 \le i \le n$. As a consequence, **De** implies for $1 \le i \le n$ that t is independent of t_i in s_{i-1}. This means that $\mathsf{pers}(s_0)$ is persistent. □

Deadlock-preserving **weak stubborn sets** use **D1** and the following condition **D2w**, that replaces both **D0** and **D2**. When transitions may be nondeterministic, some other weak stubborn set methods need stronger conditions that we will not discuss in this study.

D2w If $\mathsf{en}(s_0) \neq \emptyset$, then there is $t_\mathsf{k} \in \mathsf{stubb}(s_0)$ such that if $t_i \notin \mathsf{stubb}(s_0)$ for $1 \le i \le n$ and $s_0 \xrightarrow{t_1 \cdots t_n} s_n$, then $s_n \xrightarrow{t_\mathsf{k}}$.

By choosing $n = 0$ we see that $s_0 \xrightarrow{t_\mathsf{k}}$. That is, instead of requiring that all enabled transitions in a stubborn set remain enabled while outside transitions occur, **D2w** requires that one of them exists and remains enabled. This one is called *key transition* and denoted above with t_k.

Every strong stubborn set is also weak but not necessarily vice versa. Therefore, weak stubborn sets have potential for better reduction results. The first publication on stubborn sets [15] used weak stubborn sets. The added reduction potential of weak stubborn sets has only recently found its way to tools [4, 7, 8]. The proof of Theorem 3 goes through with **D2w** instead of **D2**. Indeed, weak stubborn sets preserve most, but not necessarily all of the properties that strong stubborn sets preserve.

Excluding a situation that does not occur with most verification tools, if the system has infinite executions, then all methods in this section preserve at least one. The nondeterministic case of this theorem is new or at least little known.

Theorem 8. *Assume that $s_0 \in S_\mathsf{r}$ and $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \ldots$. If transitions are deterministic or the reduced state space is finitely branching, then there are t'_1, t'_2, ... such that $s_0 \xrightarrow{t'_1 t'_2 \cdots}$ in the reduced state space.*

Proof. If any of the t_i is in $\mathsf{stubb}(s_0)$, then, for the smallest such i, by **D1**, there is s'_0 such that $s_0 \xrightarrow{t_i} s'_0 \xrightarrow{t_1 \cdots t_{i-1} t_{i+1} \cdots}$. Otherwise, by **D2w**, for every $j \in \mathbb{N}$ there are s'_j such that $s_j \xrightarrow{t_\mathsf{k}} s'_j$. If transitions are deterministic, then **D1** yields

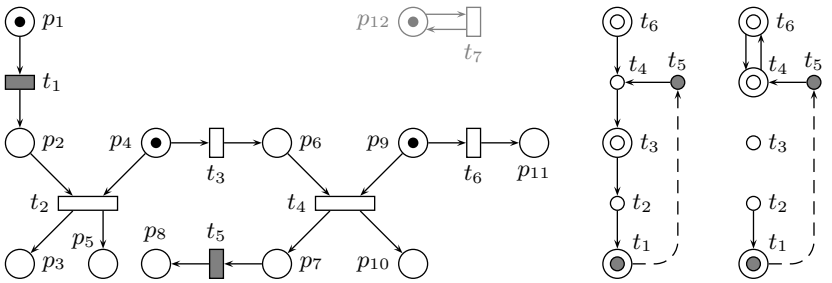

Fig. 4. A Petri net with two visible transitions and its "$\rightsquigarrow_{\{1,4,9\}}$"- and "$\rightsquigarrow_{\{1,6,9\}}$"-graphs. In the latter, p_2 is chosen as p_{t_2}. The dashed arrows arise from $\mathbf{V}$.

$s'_0 \xrightarrow{t_1} s'_1 \xrightarrow{t_2} \cdots$. This argument can be repeated at s'_0 and so on without limit, yielding the claim.

If transitions are not necessarily deterministic, then, for every $n \in \mathbb{N}$, **D1** can be applied to $s_0 \xrightarrow{t_1 \cdots t_n}$ or to $s_0 \xrightarrow{t_1 \cdots t_n t_k}$. This can be repeated n times, yielding an execution of length n in the reduced state space starting at s_0. If the reduced state space is finitely branching, then König's Lemma -type of reasoning yields the claim. □

Please notice that it is even possible that $\{t_1, t_2, \ldots\} \cap \{t'_1, t'_2, \ldots\} = \emptyset$. Often with aps set methods, if an original execution does not lead to a deadlock, then its representative in the reduced state space does not consist of precisely the same transitions. As a consequence, in the opinion of the present authors, when trying to understand aps set methods, Mazurkiewicz traces [9] and partial orders of transition occurrences are not a good starting point.

5 Visible and Invisible Transitions

Figure 4 shows a 1-safe Petri net, the directed graph that the "$\rightsquigarrow_M$"-relation spans in the shown marking $\{1, 4, 9\}$, and the similar graph for the marking $\{1, 6, 9\}$ that is obtained by firing t_3. Please ignore the grey p_{12} and t_7 until Section 6. Please ignore the dashed arrows for the moment. They will be explained soon.

Assume that we want to check whether always at least one of p_1 and p_8 is empty. We denote this property with $\Box((M(p_1) = 0) \vee (M(p_8) = 0))$. It does not hold, as can be seen by firing $t_3 t_4 t_5$.

According to the theory developed this far, $\{t_1\}$ is stubborn. Therefore, it suffices to fire just t_1 in the initial marking. After firing it, p_1 is permanently empty. As a consequence, no counterexample to $\Box((M(p_1) = 0) \vee (M(p_8) = 0))$ is found. We see that the basic strong stubborn set method does not preserve the validity of this kind of properties.

This problem can be solved in two steps. The second step will be described in Sections 6 and 7, where systems that may exhibit cyclic behaviour are discussed. The first step consists of classifying transitions as *visible* and *invisible*,

and adopting an additional requirement. The *atomic propositions* of $\Box((M(p_1) = 0) \vee (M(p_8) = 0))$ are $M(p_1) = 0$ and $M(p_8) = 0$. If a transition is known not to change the truth value of any atomic proposition in any reachable marking, it is classified as invisible. If the transition is known to change the truth value in at least one reachable marking or it is not known whether it can change it, then it is classified as visible. The additional requirement is the following.

V If $\mathsf{stubb}(s_0)$ contains an enabled visible transition, then it contains all visible transitions (also disabled ones).

In the example, the grey transitions are visible and the rest are invisible. **V** adds the dashed arrows to the "$\rightsquigarrow_M$"-graphs in Figure 4.

Assume **V**. Consider **D1**. Its t is enabled because $s_0 \xrightarrow{tt_1 \cdots t_n}$. If t is visible, then **V** implies that $t_1, \ldots, t_n$ are invisible, because they are not in $\mathsf{stubb}(s_0)$ by the assumption in **D1**. This means that when $t_1 \cdots t_n$ and $t'_1 \cdots t'_n$ are like in Theorems 2 and 3, the sequence of visible transitions within $t_1 \cdots t_n$ (that is, the projection of $t_1 \cdots t_n$ on visible transitions) is the same as the sequence of visible transitions within $t'_1 \cdots t'_n$. With Theorem 8, the projection of $t_1 t_2 \cdots$ is a prefix of the projection of $t'_1 t'_2 \cdots$ or vice versa. Sections 6 and 7 tell how they can be made the same.

For instance, $t_3 t_4 t_5 t_1$ leads to a deadlock in Figure 4. In it, t_5 occurs before t_1. **V** guarantees that t_5 occurs before t_1 also in the permutation of $t_3 t_4 t_5 t_1$ whose existence Theorem 3 promises. By executing the permutation to a point where t_5 has but t_1 has not yet occurred, a state in the reduced state space is found that violates $\Box((M(p_1) = 0) \vee (M(p_8) = 0))$. In this way **V** makes it possible to check many kinds of properties from the reduced state space.

Indeed, with the dashed arrow, the "$\rightsquigarrow_M$"-graph in Figure 4 middle yields two stubborn sets: $\{t_1, \ldots, t_5\}$ and T. In both cases, t_3 is in the stubborn set. By firing t_3, the marking $\{1, 6, 9\}$ is obtained whose "$\rightsquigarrow_M$"-graph is shown in Figure 4 right. This graph yields the stubborn sets $\{t_4, t_6\}$, $\{t_1, t_4, t_5, t_6\}$, and some others that have the same enabled transitions as one of these, such as $\{t_3, t_4, t_5, t_6\}$. All of them contain t_4. After firing it, each stubborn set contains t_1, t_5, and possibly some disabled transitions. So the sequence $t_3 t_4 t_5$ is fired in the reduced state space (after which t_1 is fired).

In the ample set theory, instead of **V** there is the following condition:

C2 If $\mathsf{ample}(s_0)$ contains a visible transition, then make $\mathsf{ample}(s_0) = \mathsf{en}(s_0)$.

This condition is stronger than **V** in the sense that **C2** always forces at least the same enabled transitions to be taken as **V**, but not necessarily vice versa. In particular, although $\{t_1, \ldots, t_5\}$ obeys **V** in the initial marking of our example, its set of enabled transitions (that is, $\{t_1, t_3\}$) does not obey **C2**. Indeed, **C2** commands to fire all enabled transitions in $\{1, 4, 9\}$, including also t_6. Therefore, ample sets yield worse reduction in this example than stubborn sets.

It is difficult to formulate **V** without talking about disabled transitions in the stubborn set. For instance, consider "if the stubborn set contains an enabled visible transition, then it contains all enabled visible transitions". It allows to

choose $\{t_1\}$ in $\{1, 4, 9\}$. However, we already saw that $\{t_1\}$ loses all counterexamples to the property. The ability to formulate better conditions than **C2** is an example of the advantages of allowing disabled transitions in stubborn sets.

The basis of the running example of this section (but not most of the details) is from [23].

6 The Ignoring Problem, Part 1: Finite Executions

Assume that the initially marked place p_{12}, transition t_7, and arcs between them are added to the Petri net in Figure 4. Before the addition, the state space of the net is acyclic and has the deadlocks $\{3, 5, 11\}$, $\{2, 8, 10\}$, and $\{2, 6, 11\}$. The addition adds number 12 and the self-loop $M \xrightarrow{t_7} M$ to each reachable marking. It adds the stubborn set $\{t_7\}$ to each reachable marking and otherwise keeps the $\sqsubseteq_M$-minimal stubborn sets the same.

If t_7 is investigated first in the initial marking $\{1, 4, 9, 12\}$, then the stubborn set $\{t_7\}$ is chosen. Firing t_7 leads back to the initial marking. Therefore, the method only constructs the initial marking and its self-loop — that is, one marking and one edge. This is correct, because also this reduced state space has no deadlocks but has an infinite execution. As a matter of fact, from the point of view of checking these two properties, the obtained reduction is ideal.

On the other hand, this reduced state space is clearly invalid for disproving $\Box((M(p_1) = 0) \vee (M(p_8) = 0))$. This problem is known as the *ignoring problem*. After finding out that t_7 causes a self-loop in every reachable marking, the method stopped and ignored the rest of the Petri net.

Let $s \xrightarrow{\text{key}} s'$ denote that there are $s_0, \ldots, s_n$ and $t_1, \ldots, t_n$ such that $s = s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} s_n = s'$ and t_i is a key transition of $\mathsf{stubb}(s_{i-1})$ for $1 \leq i \leq n$. In [17, 18], the ignoring problem was solved with the following condition **Sen**, and in [18] also with **SV**:

Sen For every $t \in \mathsf{en}(s_0)$ there is s_t such that $s_0 \xrightarrow{\text{key}} s_t$ and $t \in \mathsf{stubb}(s_t)$.
SV For every visible t there is s_t such that $s_0 \xrightarrow{\text{key}} s_t$ and $t \in \mathsf{stubb}(s_t)$.

With deterministic transitions, **D1**, **D2w**, and **Sen** guarantee that if $s \in S_\mathsf{r}$ and $s \xrightarrow{t_1 \cdots t_n}$, then there are $t'_1, \ldots, t'_m$ such that $s \xrightarrow{\pi}$ in the reduced state space for some permutation π of $t_1 \cdots t_n t'_1 \cdots t'_m$. This facilitates the verification of many properties. For instance, a transition is Petri net live (that is, from every reachable state, a state can be reached where it is enabled) if and only if it is Petri net live in the reduced state space. With deterministic transitions, **D1**, **D2w**, **V**, and **SV** guarantee that if $s \in S_\mathsf{r}$ and $s \xrightarrow{t_1 \cdots t_n}$, then there is some transition sequence π such that $s \xrightarrow{\pi}$ in the reduced state space and the projection of π on the visible transitions is the same as the projection of $t_1 \cdots t_n$.

With strong stubborn sets, **Sen** can be implemented efficiently as follows [17, 19]. Terminal strong components of the reduced state space can be recognized efficiently on-the-fly with Tarjan's algorithm [14]. (This resembles the algorithm

in Section 2, but the directed graph in question is different.) If some transition is enabled in some state of a terminal strong component but does not occur in the component, then it is enabled in every state of the component. When the algorithm is about to backtrack from the component, it checks whether there are such transitions. If there are, it expands the stubborn set of the current state such that it contains at least one of such transitions.

SV can be implemented similarly, except that the algorithm checks whether any visible transition occurs in the terminal component, and if not, it expands the stubborn set towards containing a visible transition [23].

SV is nonoptimal in the sense that expanding the stubborn set with a visible transition may force to add two enabled transitions to the stubborn set, while **Sen** and **V** together only add one of them. Also **Sen** and **V** together guarantee that projections on visible transitions are preserved (indeed, they were used in [18]), but they are nonoptimal for this purpose in the sense that they unnecessarily solve the ignoring problem also for the invisible transitions. We now present and prove correct a novel condition that is free from both of these problems.

Let $T_{\mathsf{i}} \subseteq T$ be any set of transitions. Typical examples of T_{i} are the set of visible transitions and the set of all transitions. We call its elements *the interesting transitions*. The following condition solves the ignoring problem.

S There is T' such that $T_{\mathsf{i}} \subseteq T'$, T' (in the place of $\mathsf{stubb}(s_0)$) satisfies **Dd** in s_0, and for every $t \in T' \cap \mathsf{en}(s_0)$ there is s_t such that $s_0 \xrightarrow{\text{key}} s_t$ and $t \in \mathsf{stubb}(s_t)$.

By choosing $T_{\mathsf{i}} = T$ we see that **Sen** implies **S**. **S** and **SV** treat enabled visible transitions in the same way. For each disabled visible transition t, both **S** and **SV** compute a sufficient set of transitions $T'(t)$ such that t remains disabled until at least one transition from $T'(t)$ occurs. Intuitively, **S** is better than **SV** in that **SV** forces to investigate all elements of its $T'(t)$ in the same state, while **S** allows to use many states. However, they compute $T'(t)$ in different states, so they may get different results. Therefore, this intuitive argument does not constitute a proof on the superiority of **S**. More research is needed here.

The set T' can be computed similarly to the computation of stubborn sets. We now prove that **S** is correct.

Lemma 9. *Assume that transitions are deterministic,* $s_0 \in S_{\mathsf{r}}$, $\mathsf{stubb}(s_0)$ *obeys* ***S****,* $\mathsf{stubb}(s)$ *obeys* ***D2w*** *in every* $s \in S_{\mathsf{r}}$*, and* $s_0 \xrightarrow{t_1 \cdots t_n} s_n$*, where* $t_n \in T_{\mathsf{i}}$*. There are* $s'_0, \ldots, s'_m$ *and* $t^1_{\mathsf{k}}, \ldots, t^m_{\mathsf{k}}$ *such that* $s'_0 = s_0$, $s'_0 \xrightarrow{t^1_{\mathsf{k}}} s'_1 \xrightarrow{t^2_{\mathsf{k}}} \cdots \xrightarrow{t^m_{\mathsf{k}}} s'_m$, $\{t_1, \ldots, t_n\} \cap \mathsf{stubb}(s'_i) = \emptyset$ *for* $0 \le i < m$*, and* $\{t_1, \ldots, t_n\} \cap \mathsf{stubb}(s'_m) \neq \emptyset$.

Proof. Because $t_n \in T_{\mathsf{i}} \subseteq T'$, there is $1 \le i \le n$ such that $t_i \in T'$ but $t_j \notin T'$ for $1 \le j < i$. By **Dd** $t_i \in \mathsf{en}(s_0)$. By the assumption there is s_{t_i} such that $t_i \in \mathsf{stubb}(s_{t_i})$ and $s_0 \xrightarrow{\text{key}} s_{t_i}$. Let the states along this path be called $s'_0, \ldots, s'_h$. So $s'_0 = s_0$, $s'_h = s_{t_i}$ and $t_i \in \{t_1, \ldots, t_n\} \cap \mathsf{stubb}(s'_h)$. Thus there is the smallest m such that $\{t_1, \ldots, t_n\} \cap \mathsf{stubb}(s'_m) \neq \emptyset$, completing the proof. □

Theorem 10. *Assume that transitions are deterministic and* $\mathsf{stubb}(s)$ *obeys* ***D1***, ***D2w***, *and* ***S*** *in every* $s \in S_\mathsf{r}$. *Let* $s_0 \in S_\mathsf{r}$ *and* $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} s_n$. *There are* $t'_1, \ldots, t'_m$, *and* s_m *such that* $s_0 \xrightarrow{t'_1 \cdots t'_m} s_m$ *in the reduced state space and each* $t \in T_\mathsf{i}$ *occurs in* $t'_1 \cdots t'_m$ *at least as many times as it occurs in* $t_1 \cdots t_n$. *Furthermore,*

- *If* $T_\mathsf{i} = T$, *then there are* $t_{n+1}, \ldots, t_m$ *such that* $s_n \xrightarrow{t_{n+1} \cdots t_m} s_m$ *and* $t'_1 \cdots t'_m$ *is a permutation of* $t_1 \cdots t_m$.
- *If* T_i *is the set of visible transitions and* $\mathsf{stubb}(s)$ *obeys* ***V*** *in every* $s \in S_\mathsf{r}$, *then the projections of* $t_1 \cdots t_n$ *and* $t'_1 \cdots t'_m$ *on* T_i *are the same.*

Proof. If none of $t_1, \ldots, t_n$ is in T_i, the first claim holds vacuously with $m = 0$. Otherwise let $1 \leq n' \leq n$ be the biggest such that $t_{n'} \in T_\mathsf{i}$. Lemma 9 yields s' and $t^1_\mathsf{k}, \ldots, t^{m'}_\mathsf{k}$ such that $s_0 \xrightarrow{t^1_\mathsf{k} \cdots t^{m'}_\mathsf{k}} s'$ and $\{t_1, \ldots, t_{n'}\} \cap \mathsf{stubb}(s') \neq \emptyset$. Applying **D2w**, **D1**, and determinism m' times yields s'' such that $s' \xrightarrow{t_1 \cdots t_{n'}} s''$ and $s_{n'} \xrightarrow{t^1_\mathsf{k} \cdots t^{m'}_\mathsf{k}} s''$. **D1** produces from $s' \xrightarrow{t_1 \cdots t_{n'}} s''$ a transition occurrence in the reduced state space that consumes one of $t_1, \ldots, t_{n'}$. The first claim follows by induction.

If $T_\mathsf{i} = T$, then always $n' = n$. The $t^1_\mathsf{k}, \ldots, t^{m'}_\mathsf{k}$ introduced in each application of Lemma 9 are the $t_{n+1}, \ldots, t_m$.

In the case of the last claim, each key transition is invisible, because otherwise $t_{n'}$ would be in the stubborn set of the key transition by **V**, contradicting Lemma 9. Therefore, the applications of **D2w** neither add visible transitions nor change the order of the visible transitions. By **V**, the same holds for the applications of **D1**. □

In the literature, **S** may refer to any condition that plays the role of **Sen**, **SV**, or (from now on) the **S** of the present study. This is because there is usually no need to talk about more than one version of the condition in the same publication. The name **S** refers to "safety properties", which is the class of properties whose counterexamples are finite (not necessarily deadlocking) executions.

In [20] it was pointed out that it is often possible and perhaps even desirable to modify the model such that from every reachable state, a deadlock is reachable. Reduction with deterministic transitions, **D0**, **D1**, and **D2** preserves this property. Two efficient algorithms were given for checking from the reduced state space that this property holds. Such systems trivially satisfy **S**. This solution to the ignoring problem is simple. As far it is known, it gives good reduction results. (Little is known on the relative performance of alternative solutions to the ignoring problem.)

7 The Ignoring Problem, Part 2: Diverging Executions

Figure 5 demonstrates that **S** does not always suffice to preserve a property. Consider $\Diamond\Box(M(p_2) = 0)$, that is, from some point on, p_2 remains empty. It

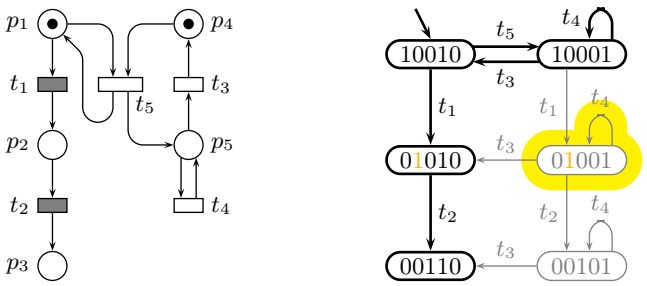

Fig. 5. Terminal strong components vs. cycles.

fails because of $t_5t_1t_4t_4t_4\cdots$. However, the figure shows a reduced state space that obeys **D0**, **D1**, **D2**, **V**, and **S**, but contains no counterexample.

This problem only arises with *diverging* counterexamples, that is, those which end with an infinite sequence of invisible transitions. When finite counterexamples apply, the methods in Section 6 suffice. If the reduced state spaces are finite (as they usually are with practical computer tools), they suffice also for counterexamples that contain an infinite number of visible transitions. This is because the methods preserve every finite prefix of the projection on visible transitions, from which König's Lemma type of reasoning proves that also the infinite projection is preserved.

With stubborn sets, this problem has been solved by two conditions that together replace **S**:

I If $\mathsf{en}(s_0)$ contains an invisible transition, then $\mathsf{stubb}(s_0)$ contains an invisible key transition.

L For every visible transition t, every cycle in the reduced state space contains a state s such that $t \in \mathsf{stubb}(s)$.

Let $t_1t_2\cdots$ be such that $s_0 \xrightarrow{t_1t_2\cdots}$ and only a finite number of the t_i are visible. Assume that $t_1t_2\cdots$ contains at least one visible transition t_v. Similarly to the proof of Theorem 10, key transitions and **D2w** are used to go to a state whose stubborn set contains some t_i, and then **D1** is used to move a transition occurrence from the sequence to the reduced state space. At most $|S_\mathsf{r}|-1$ applications of **D2w** and **D1** may be needed before some t_i such that $i \leq v$ is consumed, because otherwise the reduced state space would contain a cycle without t_v in any of its stubborn sets, violating **L**. As a consequence, each visible transition of $t_1t_2\cdots$ is eventually consumed.

When that has happened, **I** ensures that the reduced state space gets an infinite invisible suffix. Without **I**, it could happen that only visible transitions are fired immediately after consuming the last t_v, spoiling the counterexample.

A diverging execution ξ is minimal if and only if there is no infinite execution whose projection on visible transitions is a proper prefix of the projection of ξ. Minimal diverging counterexamples are preserved even without **L** and **S**. This implies that if the reduced state space is finite, then **D1**, **V**, **I**, and a variant

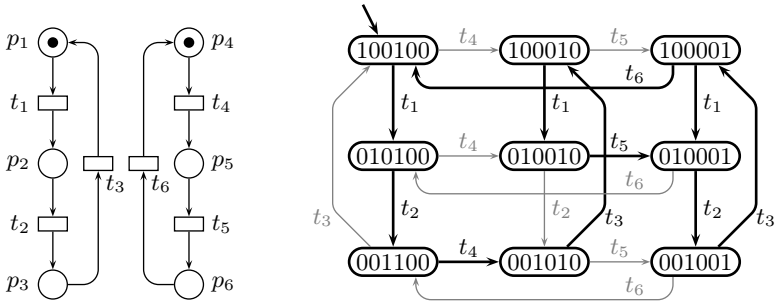

Fig. 6. Transitions are tried in the order of their indices until one is found that does not close a cycle. If such a transition is not found, then all transitions are taken.

of **D2w** preserve the failures–divergences semantics in CSP theory [13]. **D2w** is replaced by a variant, because CSP uses nondeterministic transitions.

Ample sets do not mention **I**, because it follows from **C0**, **C2**, and the fact that all transitions in an ample set are key transitions by **C1**. Instead of **L**, ample sets use the following condition.

C3 For every t and every cycle in the reduced state space, if t is enabled in some state of the cycle, then the cycle contains a state s such that $t \in \mathsf{ample}(s)$.

The relation of **L** to **C3** resembles the relation of **SV** to **Sen**. This suggests that an improvement on **C3** could be developed similarly to how **S** improves **Sen**. We leave this for future research.

The recommended implementation of **C3** is called **C3'** in [1]. It assumes that the reduced state space is constructed in depth-first order. It also implements **L**.

C3' If $\mathsf{ample}(s) \neq \mathsf{en}(s)$, then for every $t \in \mathsf{ample}(s)$ and every s' such that $s \xrightarrow{t} s'$, s' is not in the depth-first stack.

Figure 6 illustrates that **C3'** sometimes leads to the construction of unnecessarily many states. In it, all reachable states are constructed, although the processes do not interact at all. Better results are obtained if the component (either $\{t_1, t_2, t_3\}$ or $\{t_4, t_5, t_6\}$) is preferred to which the most recent transition belongs. Then the sequence $t_1 t_2 t_4 t_5 t_3 t_1$ is fired, after which both t_2 and t_6 are fired. This improved idea fails badly with the three-dimensional version of the example. In [2], the bad performance of **C3'** was illustrated with other examples.

C3' fully expands the last state of the cycle it closes. If the first state of the cycle is expanded instead and if the component is remembered, then the leftmost column and topmost row of Figure 6 right are constructed. This is better than with **C3'**, and works well also in the three-dimensional example. There has been very little research on the performance of cycle conditions besides [2], although the problem is clearly important.

8 Conclusions

The goal in the development of stubborn sets has been as good reduction as possible, while ample and persistent sets have favoured straightforward easily implementable conditions and algorithms. As a consequence, where stubborn set methods differ from other aps set methods, stubborn sets tend to be more difficult to implement but yield better reduction results. Very little is known on the differences of the reduction power between different methods. Reliable information is difficult to obtain experimentally, because in addition to the issue that is being experimented, the results may depend on the optimality of the chosen "$\rightsquigarrow_s$"- or independence relation, on the order in which the transitions are listed in the input file (Section 3), and other things.

Some stubborn set ideas are difficult to implement efficiently. For instance, no very fast algorithm is known that can utilize the freedom to choose any one from among the places that disable a transition (the p_t in Section 2). On the other hand, the likelyhood of finding good ways of exploiting some reduction potential decreases significantly, if the existence of the potential is never pointed out.

The algorithm in Section 2 seems intuitively very good, and experiments with the ASSET tool strongly support this view [20, 21, 23]. The present authors believe that it deserves more attention than it has received.

The biggest immediate difference between stubborn sets and other aps set methods is the possibility of disabled transitions in the set. It is difficult to think of the above-mentioned algorithm without this possibility. Furthermore, in Section 5 it was shown how it facilitates an improvement to the visibility condition. It is also important that stubborn sets allow nondeterministic transitions.

Perhaps the most important area where more research is needed is the ignoring problem. The example in Figure 6 may be extreme and thus not representative of the typical situation. Unfortunately, very little is known on what happens in the typical situation with each solution.

References

1. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999) 314 p
2. Evangelista, S., Pajault, C.: Solving the Ignoring Problem for Partial Order Reduction. Software Tools for Technology Transfer 12(2) (2010) 155–170
3. Eve, J., Kurki-Suonio, R.: On Computing the Transitive Closure of a Relation. Acta Informatica 8(4) (1977) 303–314
4. Gibson-Robinson, T., Hansen, H., Roscoe, A.W., Wang, Xu: Practical Partial Order Reduction for CSP. In: Havelund, K., Holzmann, G., Joshi, R. (eds.): NASA Formal Methods 2015, LNCS, vol. 9058 (2015) 188–203
5. Godefroid, P.: Using Partial Orders to Improve Automatic Verification Methods. In: Clarke, E.M., Kurshan, R.P. (eds.) Computer-Aided Verification '90, AMS–ACM DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 3 (1991) 321–340
6. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem. LNCS, vol. 1032 (1996)

7. Hansen H., Lin, S., Liu Y., Nguyen, T.K., Sun, J.: Diamonds Are a Girl's Best Friend: Partial Order Reduction for Timed Automata with Abstractions. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification, 26th International Conference, LNCS, vol. 8559 (2014) 391–406
8. Laarman, A., Pater, E., van de Pol, J., Hansen, H.: Guard-Based Partial-Order Reduction. Software Tools for Technology Transfer (2014) 1–22
9. Mazurkiewicz, A.: Trace Theory. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II, LNCS, vol. 255 (1987) 279–324
10. Peled, D.: All from One, One for All: On Model Checking Using Representatives. In: Courcoubetis, C. (ed.) Computer Aided Verification, 5th International Conference, LNCS, vol. 697 (1993) 409–423
11. Peled, D: Ten Years of Partial Order Reduction. In: Hu, A.J., Vardi, M.Y. (eds.) Computer Aided Verification, 10th International Conference, LNCS, vol. 1427 (1998) 17–28
12. Rauhamaa, M.: A Comparative Study of Methods for Efficient Reachability Analysis. Lic. Tech. Thesis, Helsinki University of Technology, Digital Systems Laboratory, Research Report A-14. Espoo, Finland (1990)
13. Roscoe, A.W.: Understanding Concurrent Systems. Springer, Heidelberg, Germany (2010) 533 p
14. Tarjan, R.E.: Depth-First Search and Linear Graph Algorithms. SIAM Journal on Computing 1(2) (1972) 146–160
15. Valmari, A.: Error Detection by Reduced Reachability Graph Generation. In: Proceedings of the 9th European Workshop on Application and Theory of Petri Nets (1988) 95–122
16. Valmari, A: State Space Generation: Efficiency and Practicality. Dr. Techn. Thesis, Tampere University of Technology Publications 55, Tampere (1988)
17. Valmari, A.: Stubborn Sets for Reduced State Space Generation. In: Rozenberg, G. (ed.) Advances in Petri Nets 1990. LNCS, vol. 483 (1991) 491–515
18. Valmari, A.: Stubborn Set Methods for Process Algebras. In: Peled, D., Pratt, V., Holzmann, G. (eds.) Partial Order Methods in Verification, Proceedings of a DIMACS Workshop, DIMACS Series in Discrete Mathematics and Theoretical Computer Science Vol. 29, American Mathematical Society (1997) 213–231
19. Valmari, A.: The State Explosion Problem. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, LNCS, vol. 1491 (1998) 429–528
20. Valmari, A.: Stop It, and Be Stubborn! In: Haar, S., Meyer, R. (eds.) 15th International Conference on Application of Concurrency to System Design, IEEE Computer Society (2015) 10–19, DOI 10.1109/ACSD.2015.14
21. Valmari, A.: A State Space Tool for Concurrent System Models Expressed In C++. In: Nummenmaa, J., Sievi-Korte, O., Mäkinen, E. (eds.) SPLST 2015, Symposium on Programming Languages and Software Tools, CEUR Workshop Proceedings 1525 (2015) 91–105
22. Valmari, A., Hansen, H.: Can Stubborn Sets Be Optimal? Fundamenta Informaticae 113(3–4) (2011) 377–397
23. Valmari, A., Vogler, W.: Fair Testing and Stubborn Sets. In: Bošnački, D., Wijs, A. (eds.) Model Checking Software, 23rd International Symposium, LNCS, vol. 9641 (2016) 225–243
24. Varpaaniemi, K.: On the Stubborn Set Method in Reduced State Space Generation. PhD Thesis, Helsinki University of Technology, Digital Systems Laboratory Research Report A-51, Espoo, Finland (1998)