

This is a self-archived version of an original article. This version may differ from the original in pagination and typographic details.

Author(s): Neele, Thomas; Valmari, Antti; Willemse, Tim A. C.

Title: A Detailed Account of The Inconsistent Labelling Problem of Stutter-Preserving Partial-Order Reduction

Year: 2021

Version: Published version

Copyright: © T. Neele, A. Valmari, and T.A.C. Willemse

Rights: CC BY 4.0

Rights url: <https://creativecommons.org/licenses/by/4.0/>

Please cite the original version:

Neele, T., Valmari, A., & Willemse, T. A. C. (2021). A Detailed Account of The Inconsistent Labelling Problem of Stutter-Preserving Partial-Order Reduction. *Logical Methods in Computer Science*, 17(3), Article 8. [https://doi.org/10.46298/lmcs-17\(3:8\)2021](https://doi.org/10.46298/lmcs-17(3:8)2021)

A DETAILED ACCOUNT OF THE INCONSISTENT LABELLING PROBLEM OF STUTTER-PRESERVING PARTIAL-ORDER REDUCTION *

THOMAS NEELE ^a, ANTTI VALMARI ^b, AND TIM A.C. WILLEMSE ^c

^a Royal Holloway University of London, Egham, UK
e-mail address: thomas.neele@rhul.ac.uk

^b University of Jyväskylä, Jyväskylä, Finland
e-mail address: antti.valmari@jyu.fi

^c Eindhoven University of Technology, Eindhoven, The Netherlands
e-mail address: t.a.c.willemse@tue.nl

ABSTRACT. One of the most popular state-space reduction techniques for model checking is partial-order reduction (POR). Of the many different POR implementations, stubborn sets are a very versatile variant and have thus seen many different applications over the past 32 years. One of the early stubborn sets works shows how the basic conditions for reduction can be augmented to preserve stutter-trace equivalence, making stubborn sets suitable for model checking of linear-time properties. In this paper, we identify a flaw in the reasoning and show with a counter-example that stutter-trace equivalence is not necessarily preserved. We propose a stronger reduction condition and provide extensive new correctness proofs to ensure the issue is resolved. Furthermore, we analyse in which formalisms the problem may occur. The impact on practical implementations is limited, since they all compute a correct approximation of the theory.

1. INTRODUCTION

In the field of formal methods, model checking is a push-button technique for establishing the correctness of systems according to certain criteria. A fundamental issue in model checking is the *state-space explosion* problem: the size of the state space can grow exponentially with the number of concurrent components, due to all their possible interleavings. One of the prime methods of reducing the number of states is *partial-order reduction* (POR). The literature contains many different implementations of POR, but they are all centred around the idea that some interleavings may be considered similar and thus only one interleaving from each equivalence class needs to be explored. The main variants of POR are *ample sets* [Pel93], *persistent sets* [God96] and *stubborn sets* [Val91b, VH17]. The basic conditions set out by each of these variants can be strengthened, such that the resulting conditions are sufficient for the preservation of stutter-trace equivalence. The extra conditions resolve the

Key words and phrases: partial-order reduction, stutter equivalence, LTL, stubborn sets.

* An extended abstract of this paper appeared earlier as [NVW20].

so-called *action-ignoring problem* [Val91b]. Since LTL without the next operator ($LTL_{\neg X}$) is invariant under finite stuttering, this allows one to check most LTL properties under POR.

However, the correctness proofs for these methods are intricate and not reproduced often. For stubborn sets, $LTL_{\neg X}$ -preserving conditions and an accompanying correctness result were first presented in [Val91a]; the corresponding proofs appeared in [Val92]. When attempting to reproduce the proof of [Val92, Theorem 2] (see also Theorem 2.5 in the current work), we were unable to show that the two alternative paths considered by [Val92, Construction 1], a core component of the proof, are stutter equivalent. The consequence is that stutter-trace equivalence is not necessarily preserved, contrary to what the theorem states! We call this the *inconsistent labelling problem*.

The essence of the problem is that POR in general, and the proofs in [Val92] in particular, reason mostly about actions, which label the transitions. In POR theory, the only relevance of the state labelling is that it determines which actions must be considered *visible*. On the other hand, stutter-trace equivalence and the LTL semantics are purely based on state labels. The correctness proof in [Val92] does not deal properly with this disparity. Consequently, any application of stubborn sets in $LTL_{\neg X}$ model checking is possibly unsound, both for safety and liveness properties. In literature, the correctness of several theories [LPvdPH16, LW19, Val96] relies on the incorrect theorem.

In earlier work [NVW20], we identified the inconsistent labelling problem and investigated the theoretical and practical consequences. As detailed in *ibid.*, the problem is witnessed by a counter-example, which is valid for weak stubborn sets and, with a small modification, in a non-deterministic setting for strong stubborn sets. A slight strengthening of one of the stubborn set conditions is sufficient to repair the issue (Theorems 5.2 and 5.3 in the current work). The fix is local, in the sense that it reduces the reduction potential in those places where the inconsistent labelling problem might otherwise occur. Petri nets can be susceptible to the issue, depending on what notion of invisibility and what types of atomic propositions are used. We used this knowledge about formalisms in which the inconsistent labelling problem may manifest itself to determine its impact on related work. The investigation in [NVW20] shows that probably all practical implementations of stubborn sets compute an approximation which resolves the inconsistent labelling problem. Furthermore, POR methods based on the standard independence relation, such as ample sets and persistent sets, are not affected. The current paper improves on [NVW20] with extended explanation and full proofs. In particular, we introduce each of the existing stubborn set conditions with reworked proofs to aid the reader's intuition.

The rest of the paper is structured as follows. In Section 2, we introduce the basic concepts of transition systems and stutter-trace equivalence. Section 3 introduces the stubborn set conditions one by one and shows what they preserve through several lemmata. Our counter-example to the preservation of stutter-trace equivalence is presented in Section 4. We propose a solution to the inconsistent labelling problem in Section 5, together with an updated correctness proof. Sections 6 and 7 discuss several settings in which correctness is not affected. Finally, Section 8 discusses related work and Section 9 presents a conclusion.

2. PRELIMINARIES

2.1. Labelled State Transition Systems and Paths. Since LTL relies on state labels and POR relies on edge labels, we assume the existence of some fixed set of atomic propositions

AP to label the states and a fixed set of edge labels Act , which we will call *actions*. Actions are typically denoted with the letter a .

Definition 2.1. A *labelled state transition system*, short *LSTS*, is a directed graph $TS = (S, \rightarrow, \hat{s}, L)$, where:

- S is the state space;
- $\rightarrow \subseteq S \times Act \times S$ is the transition relation;
- $\hat{s} \in S$ is the initial state; and
- $L : S \rightarrow 2^{AP}$ is a function that labels states with atomic propositions.

We write $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \rightarrow$. An action a is *enabled* in a state s , notation $s \xrightarrow{a}$, if and only if there is a transition $s \xrightarrow{a} s'$ for some s' . In a given LSTS TS , $enabled_{TS}(s)$ is the set of all enabled actions in a state s . We may drop the subscript TS if it is clear from the context. A state s is a *deadlock* in TS if and only if $enabled_{TS}(s) = \emptyset$.

A *path* is a (finite or infinite) alternating sequence of states and actions that respects the transition relation: $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \dots$. We sometimes omit the intermediate and/or final states if they are clear from the context or not relevant, and write $s \xrightarrow{a_1 \dots a_n} s'$ or $s \xrightarrow{a_1 \dots a_n}$ for finite paths and $s \xrightarrow{a_1 a_2 \dots}$ for infinite paths. The empty sequence is denoted with ε . Thus, for all states s and s' , $s \xrightarrow{\varepsilon} s'$ holds if and only if $s = s'$. A path is *deadlocking* if and only if it ends in a deadlock. A path is *complete* if and only if it is infinite or deadlocking. Paths that start in the initial state \hat{s} are called *initial paths*.

Given a path $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \dots$, the *trace* of π is the sequence of state labels observed along π , *viz.* $L(s_0)L(s_1)L(s_2) \dots$. The *no-stutter trace* of π , notation $\text{no-stut}(\pi)$, is the sequence of those $L(s_i)$ such that $i = 0$ or $L(s_i) \neq L(s_{i-1})$.

A set \mathcal{I} of *invisible* actions is chosen such that if (but not necessarily only if) $a \in \mathcal{I}$, then for all states s and s' , $s \xrightarrow{a} s'$ implies $L(s) = L(s')$. Note that this definition allows the set \mathcal{I} to be under-approximated. An action that is not invisible is called *visible*. The *projection* of $a_1 \dots a_n$ on the visible actions is the result of the removal of all elements of \mathcal{I} from $a_1 \dots a_n$. We denote it with $\text{vis}_{\mathcal{I}}(a_1 \dots a_n)$. The notion extends naturally to infinite sequences $a_1 a_2 \dots$. We furthermore lift the function vis to paths, such that $\text{vis}_{\mathcal{I}}(s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots) = \text{vis}_{\mathcal{I}}(a_1 a_2 \dots)$. The subscript \mathcal{I} is omitted when it is clear from the context.

We say TS is *deterministic* if and only if $s \xrightarrow{a} s_1$ and $s \xrightarrow{a} s_2$ imply $s_1 = s_2$, for all states s, s_1 and s_2 and actions a . To indicate that TS is not necessarily deterministic, we say TS is *non-deterministic*.

2.2. Petri Nets. *Petri nets* are a widely-known formalism for modelling concurrent processes and have seen frequent use in the application of stubborn set theory [BJLM19, LW19, VH17, Var05]. We will use Petri nets for presenting examples. In Section 7, we will also reassess the correctness of some published POR theories that use Petri nets. Other than that, the theory in the present paper is fairly general, that is, it does not depend on Petri Nets.

A Petri net (P, T, W, \hat{m}) contains a set of *places* P and a set of *structural transitions* T . These sets are disjoint. In this paper they are finite. Figure 1 shows an example of a Petri net. Places are drawn as circles and structural transitions as rectangles.

Arcs between places and structural transitions and their *weights* are specified via a total function $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$. The values $W(p, t)$ and $W(t, p)$ are called weights. There is an arc from place p to structural transition t , drawn as an arrow, if and only if $W(p, t) > 0$; and similarly in the opposite direction if and only if $W(t, p) > 0$. If $W(p, t) > 1$

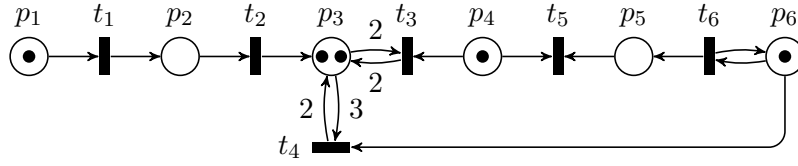


Figure 1. An example Petri net.

or $W(t, p) > 1$, then it is written as a number next to the arc. Figure 1 contains 11 arcs of weight 1, three arcs of weight 2, and one arc of weight 3.

A *marking* $m : P \rightarrow \mathbb{N}$ is a function that assigns a number of *tokens* to each place. Let \mathcal{M} denote the set of all markings. A Petri net has an *initial marking* \hat{m} . The initial marking of the example satisfies $\hat{m}(p_3) = 2$, $\hat{m}(p_1) = \hat{m}(p_4) = \hat{m}(p_6) = 1$ and $\hat{m}(p_2) = \hat{m}(p_5) = 0$.

Structural transition t is *enabled* in marking m if and only if $m(p) \geq W(p, t)$ for every $p \in P$, and *disabled* otherwise. In our example, t_1 , t_3 and t_6 are enabled. Because $\hat{m}(p_3) = 2$ but $W(p_3, t_4) = 3$, t_4 is disabled. An enabled transition may *occur* resulting in the marking m' such that $m'(p) = m(p) - W(p, t) + W(t, p)$ for every $p \in P$. We denote this with $m \xrightarrow{t} m'$, and extend the notation to paths similarly to Section 2.1. If m is the marking such that $\hat{m} \xrightarrow{t_1} m$ in our example, then $m(p_1) = 0$, $m(p_2) = 1$, and $m(p) = \hat{m}(p)$ for the remaining places. If $\hat{m} \xrightarrow{t_3} m'$, then $m'(p_4) = 0$ and $m'(p) = \hat{m}(p)$ for the remaining places.

A marking m is *reachable* if and only if there are t_1, \dots, t_n such that $\hat{m} \xrightarrow{t_1 \dots t_n} m$. Let \mathcal{M}_{reach} denote the set of reachable markings, and \rightarrow' the restriction of \rightarrow on $\mathcal{M}_{reach} \times T \times \mathcal{M}_{reach}$. Assume that a set of atomic propositions AP and a function $L' : \mathcal{M}_{reach} \rightarrow 2^{AP}$ are given. A Petri net together with these induces the LSTS $(\mathcal{M}_{reach}, \rightarrow', \hat{m}, L')$. In this context $Act = T$.

It is customary to abuse notation by forgetting about the distinction between \rightarrow and \rightarrow' , and using the same symbol for both. This is done because it is often not known in advance whether a marking is reachable, making it impractical to define \rightarrow' instead of \rightarrow . Similarly instead of L' , it is customary to define a function L from all markings \mathcal{M} to 2^{AP} , let L' be its restriction on \mathcal{M}_{reach} , and abuse notation by using the same symbol for both. These are general practice instead of being restricted to Petri nets.

2.3. Weak and Stutter Equivalence. Stubborn sets save effort by constructing, instead of the full LSTS $TS = (S, \rightarrow, \hat{s}, L)$, a reduced LSTS $TS_r = (S_r, \rightarrow_r, \hat{s}, L_r)$ such that $S_r \subseteq S$, $\rightarrow_r \subseteq \rightarrow$ and L_r is the restriction of L on S_r (more details will be given in Section 3). To reason about the similarity of an LSTS TS and its reduced LSTS TS_r , we introduce the notions *weak equivalence*, which operates on actions, and *stutter equivalence*, which operates on states. For the purpose of the discussion in Section 7, these concepts respectively depend on a set of actions and a labelling function.

Definition 2.2. Two paths π and π' are weakly equivalent with respect to a set of actions A , notation $\pi \sim_A \pi'$, if and only if they are both finite or both infinite, and their respective projections on $Act \setminus A$ are equal, *i.e.*, $vis_A(\pi) = vis_A(\pi')$.

Definition 2.3. Paths π and π' are stutter equivalent under L , notation $\pi \triangleq_L \pi'$, if and only if they are both finite or both infinite, and they yield the same no-stutter trace under L .

We typically consider weak equivalence with respect to the set of invisible actions \mathcal{I} . In that case, we simply refer to the equivalence as weak equivalence and we write $\pi \sim \pi'$,

which intuitively means that π and π' contain the same visible actions. We also omit the subscript for stutter equivalence when reasoning about the labelling function of the LSTS under consideration and write $\pi \triangleq \pi'$. Note that stutter equivalence is invariant under finite repetitions of state labels, hence its name. We lift both equivalences to LSTSs, and say that TS and TS' are *weak-trace equivalent* iff for every complete initial path π in TS , there is a weakly equivalent complete initial path π' in TS' and vice versa. Likewise, TS and TS' are *stutter-trace equivalent* iff for every complete initial path π in TS , there is a stutter equivalent complete initial path π' in TS' and vice versa.

In general, weak equivalence and stutter equivalence are incomparable, even for complete initial paths. However, for some LSTSs, these notions *are* related in a certain way. We formalise this in the following definition.

Definition 2.4. An LSTS is *labelled consistently* iff for all complete initial paths π and π' , $\pi \sim \pi'$ implies $\pi \triangleq \pi'$.

It follows from the definition that, if an LSTS TS is labelled consistently and weak-trace equivalent to a subgraph TS' , then TS and TS' are also stutter-trace equivalent.

Stubborn sets as defined in the next section aim to preserve stutter-trace equivalence between the original and the reduced LSTS. The motivation behind this is that two stutter-trace equivalent LSTSs satisfy exactly the same formulae [BK08] in LTL_{-X} . The following theorem, which is frequently cited in the literature [LPvdPH16, LW19, Val96], aims to show that stubborn sets indeed preserve stutter-trace equivalence. Its original formulation reasons about the validity of an arbitrary LTL_{-X} formula. Here, we give the alternative formulation based on stutter-trace equivalence.

Theorem 2.5. [Val92, Theorem 2] *For every LSTS TS , the reduced LSTS TS_r (defined in Section 3) is stutter-trace equivalent to TS .*

The original proof correctly establishes the four items listed below. For a long time it was believed that they suffice to ensure that TS_r gives the same truth values to LTL_{-X} formulas as TS gives. While investigating the application of stubborn sets to parity games [NWW20], Thomas Neele (the main author of the current paper, but not the author of this sentence) took the effort of checking this self-evident “fact”, and found out that it does not hold. We call this the *inconsistent labelling problem*. A counter-example is in Section 4.

- (1) Every initial deadlocking path of TS has a weakly equivalent initial deadlocking path in TS_r .
- (2) Every initial deadlocking path of TS_r has a weakly equivalent initial deadlocking path in TS .
- (3) Every initial infinite path of TS has a weakly equivalent initial infinite path in TS_r .
- (4) Every initial infinite path of TS_r has a weakly equivalent initial infinite path in TS .

Because the four items in this list are sufficient for $TS \sim TS_r$, the issue could be resolved with the additional requirement that TS is consistently labelled, which would yield $TS \triangleq TS_r$ (since TS_r is a subgraph of TS , see Definition 3.1). However, this requirement is rather strong; we propose a more local solution in Section 5.

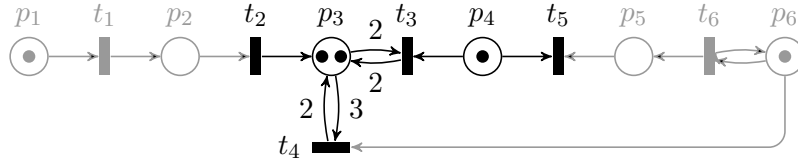


Figure 2. An example motivating **D1** and **D2w**.

3. STUBBORN SETS

3.1. Basic Ideas. In POR, *reduction functions* play a central role. A reduction function $r : S \rightarrow 2^{Act}$ indicates which actions to explore in each state. When starting at the initial state \hat{s} , a reduction function induces a *reduced LSTS* as follows.

Definition 3.1. Let $TS = (S, \rightarrow, \hat{s}, L)$ be an LSTS and $r : S \rightarrow 2^{Act}$ a reduction function. Then the *reduced LSTS* induced by r is defined as $TS_r = (S_r, \rightarrow_r, \hat{s}, L_r)$, where L_r is the restriction of L on S_r , and S_r and \rightarrow_r are the smallest sets such that the following holds:

- $\hat{s} \in S_r$; and
- If $s \in S_r$, $s \xrightarrow{a} s'$ and $a \in r(s)$, then $s' \in S_r$ and $s \xrightarrow{a}_r s'$.

Note that we have $\rightarrow_r \subseteq \rightarrow$.

In the first paper on stubborn sets [Val88], the set $r(s)$ was constructed so that if enabled actions exist, then it contains an enabled action that the outside world cannot disable. This inspired the thought that the set is “stubborn”, that is, determined to do something and not letting the outside world prevent it. Much more than this is needed to make TS_r yield correct answers to verification questions concerning TS . Furthermore, some more recent methods do not necessarily put an enabled action in $r(s)$ even if enabled actions do exist. So the name is imprecise, but has remained in use.

The main question now is how to implement a practical reduction function so that answers to interesting verification questions can be obtained from the reduced LSTSs. Because this publication is about fixing an error that had been lurking for decades, we feel appropriate to present the full proof of the affected theorem anew as clearly as possible, in more detail than originally, to minimise the possibility that other errors remain. To this end, we proceed in small steps.

We first discuss the motivating example from Figure 1, reproduced here in Figure 2. Assume that we know that the places adjacent to t_3 are p_3 and p_4 ; they contain 2 and 1 tokens, respectively; the transitions adjacent to p_3 and p_4 are t_2 to t_5 ; and the arcs between them and their weights are as is shown in Figure 2. That is, we know the black part but not the grey part in the figure. Although our knowledge is incomplete, we can reason as follows that t_3 is enabled and remains enabled until t_3 or t_5 occurs. It is enabled by the numbers of tokens in p_3 and p_4 , and by the weights of the arcs from them to t_3 . An occurrence of t_2 does not decrement the numbers of tokens in p_3 and p_4 , so it cannot disable t_3 . The same applies to t_1 and t_6 . An occurrence of t_4 decrements the number of tokens in p_3 (but not in p_4). However, thanks to the arc weight 2, it is guaranteed to leave at least 2 tokens in p_3 . So it cannot disable t_3 either.

This is an example of the kind of observations that stubborn set methods exploit. Together with some other observations that will be discussed soon, it will let us choose

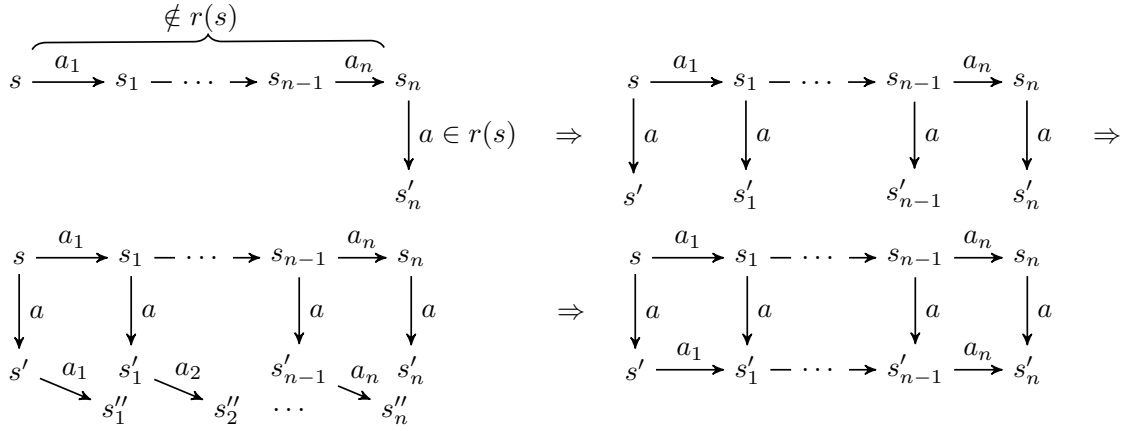


Figure 3. Visual representation of why **D1** holds on the example.

$r(s) = \{t_3, t_5, t_6\}$, where s denotes the marking shown in Figure 2. Unfortunately, the observation is Petri net-specific. We now introduce a more abstract notion that captures the same idea: t_3 is a *key action* of $r(s) = \{t_3, t_5, t_6\}$ in the sense of the following definition.

Definition 3.2. An action a is a *key action* of $r(s)$ in s if and only if for all paths $s \xrightarrow{a_1 \dots a_n} s'$ such that $a_1 \notin r(s), \dots, a_n \notin r(s)$, it holds that $s' \xrightarrow{a}$.

We typically denote key actions by a_{key} . Note that a key action must be enabled in s : by setting $n = 0$, we have $s = s'$ and $s \xrightarrow{a}$.

Many stubborn set methods assume that the sets $r(s)$ satisfy the following condition.

D2w: If $\text{enabled}(s) \neq \emptyset$, then $r(s)$ contains a key action in s .

In Figure 2, t_5 is not a key action of $\{t_3, t_5, t_6\}$, because it is disabled. Also t_6 is not, because the sequence $t_1 t_2 t_4$ disables it.

On the other hand, we now show that t_3, t_5 and t_6 have another property that stubborn set methods exploit: Figure 3 holds for each of them in the role of a and any finite sequence of elements of $\{t_1, t_2, t_4\}$ in the role of $a_1 \dots a_n$. We call t_1, t_2 and t_4 the outside transitions.

Although the outside transitions can disable t_6 , they cannot enable it again, because none of them can add tokens to p_6 . Therefore, if t_6 is enabled after the occurrence of some sequence $a_1 \dots a_n$ of outside transitions, then it was enabled in the original marking s and in every marking between s and s_n . This is illustrated by the first implication in Figure 3, with t_6 in the role of a . The first implication applies to t_3 as well, because its right-hand side applies, because t_3 is a key action of $\{t_3, t_5, t_6\}$ in s .

Neither t_3 nor t_6 can disable outside transitions, because although they temporarily consume tokens from p_3 or p_6 , they put the same number of tokens back to them; and the outside transitions do not need tokens from p_4 . This yields the second implication in the figure. Furthermore, Petri nets are commutative in the sense that if $m \xrightarrow{t't} m'$ and $m \xrightarrow{tt'} m''$, then $m' = m''$. The last implication in the figure holds because of this.

The implication chain also applies to t_5 as a , but for a different reason: t_5 is disabled, and no sequence of outside transitions can enable it, because only t_6 can enable it. Therefore, no member of the chain holds for t_5 , so the chain holds vacuously.

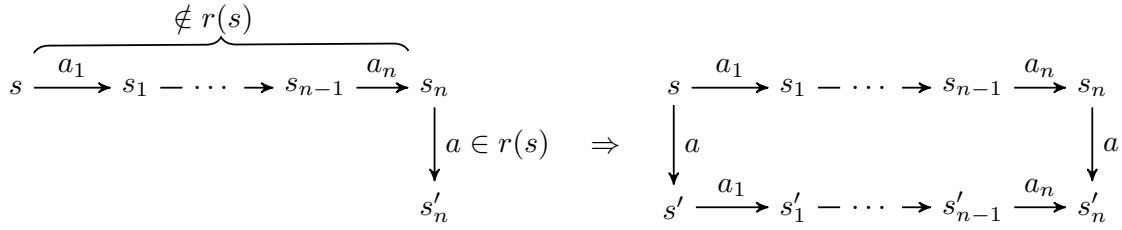


Figure 4. Visual representation of condition **D1**.

Again, we appealed to particular properties of Petri nets. To make the ideas applicable to a wide variety of formalisms for representing systems, we introduce the following condition, which is required to hold for all $r(s)$. It is illustrated in Figure 4. We showed above that it holds for $r(s) = \{t_3, t_5, t_6\}$ in Figure 2.

D1: For all states s_1, \dots, s_n, s'_n and all $a \in r(s)$ and $a_1 \notin r(s), \dots, a_n \notin r(s)$, if $s \xrightarrow{a_1} \dots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$, then there are states $s', s'_1, \dots, s'_{n-1}$ such that $s \xrightarrow{a} s' \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s'_n$.

3.2. Deadlock Detection and Its Implementation. The conditions **D2w** and **D1** are important, because they suffice for proving that all reachable deadlocks of the full LSTS are present also in the reduced LSTS. Furthermore, the deadlocks can be reached in the reduced LSTS by re-ordering the actions in the paths in the full LSTS that lead to them. In the theory below, recall that \rightarrow_r indicates which transitions occur in the reduced LSTS.

Theorem 3.3. *Assume that each $r(s)$ obeys **D1** and **D2w**. If $s_0 \in S_r$, s_n is a deadlock in TS , and $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n$, then there is a permutation $b_1 \dots b_n$ of $a_1 \dots a_n$ such that $s_0 \xrightarrow{b_1 \dots b_n}_r s_n$.*

Proof. We prove the claim by induction on n . If $n = 0$ then $s_n = s_0$ and $a_1 \dots a_n = \varepsilon$, so the claim holds trivially with $b_1 \dots b_n = \varepsilon$.

From now on, let $n > 0$. We have $s_0 \xrightarrow{a_1}$ and thus $\text{enabled}(s_0) \neq \emptyset$. By **D2w**, $r(s_0)$ contains a key action a_{key} . If none of a_1, \dots, a_n is in $r(s_0)$, then by definition $s_n \xrightarrow{a_{\text{key}}}$. However, that cannot be the case, because we assumed that s_n is a deadlock. Therefore, there is $1 \leq i \leq n$ such that $a_i \in r(s_0)$. We choose the smallest such i , yielding $a_j \notin r(s_0)$ for $1 \leq j < i$. By this choice, **D1** applies with a_i in the role of a . So there are states $s'_0, s'_1, \dots, s'_{i-1}$ such that $s'_{i-1} = s_i$ and $s_0 \xrightarrow{a_i} s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots \xrightarrow{a_{i-1}} s'_{i-1}$. Because $a_i \in r(s_0)$ we have $s'_0 \in S_r$ and $(s_0, a_i, s'_0) \in \rightarrow_r$, that is, $s_0 \xrightarrow{a_i}_r s'_0$. We remember that $s_i \xrightarrow{a_{i+1} \dots a_n} s_n$, so $s'_0 \xrightarrow{a_1 \dots a_{i-1} a_{i+1} \dots a_n} s_n$. This path is one shorter than $s_0 \xrightarrow{a_1 \dots a_n} s_n$. Therefore, the induction assumption yields a permutation $b_2 \dots b_n$ of $a_1 \dots a_{i-1} a_{i+1} \dots a_n$ such that $s'_0 \xrightarrow{b_2 \dots b_n}_r s_n$. As a consequence, $s_0 \xrightarrow{a_i b_2 \dots b_n}_r s_n$. \square

The preservation of deadlocks needs also the following facts, which are easy to check from the definitions.

- If $s_0 \xrightarrow{a_1 \dots a_n}_r s_n$, then $s_0 \xrightarrow{a_1 \dots a_n} s_n$.
- $s \in S_r$ is a deadlock in TS_r if and only if it is a deadlock in TS .

To implement this deadlock detection method, an algorithm is needed that, given state s , computes a set $r(s)$ that satisfies **D1** and **D2w**. We already illustrated with Figure 2 that this may depend on the details of the formalism used to represent the system under

verification. Because it is sometimes very difficult to check whether **D1** and **D2w** hold, the algorithms rely on formalism-specific heuristics that may give a false negative but cannot give a false positive. The set of all actions satisfies **D1** and **D2w**. While it yields no reduction, it can be used as a fall-back when attempts to find a better set fail.

The algorithm design problem also involves a trade-off between the time it takes to compute the set, and the quality of the set: smaller sets tend to result in smaller reduced LSTSs (although this issue is not straightforward [VH17]). In the case of 1-safe Petri nets, testing whether a singleton set $\{t\}$ is a valid $r(s)$ for the purpose of preserving all deadlocks is **PSPACE**-hard [VH11]. This means that there is not much hope of a fast algorithm that always yields the best possible $r(s)$.

Instead, algorithms range from quick and simple that exploit only the most obvious reduction possibilities, to very complicated that spend unreasonable amounts of time and memory in trying to find a set with few enabled actions. For instance, after finding out that t_5 may disable t_3 in Figure 2, $\{t_3\}$ must be rejected as a candidate $r(s)$. A simple algorithm might revert to the set of all actions, while a more complicated algorithm might try $\{t_3, t_5\}$, detect that t_6 might enable t_5 , try $\{t_3, t_5, t_6\}$, and find out that it works.

Fortunately, the kind of analysis that led us from $\{t_3\}$ to $\{t_3, t_5, t_6\}$ is not at all too expensive, if we are okay with some imperfection. It can be performed in linear time by formulating it as the problem of finding certain kinds of maximal strongly connected components in a directed graph whose edges $t \rightsquigarrow t'$ represent the notion “if $t \in r(s)$, then also $t' \in r(s)$ ” (e.g., [VH17]). The result is optimal in a sense that is meaningful albeit not perfect [VH11]. (In the light of **PSPACE**-hardness, we should not expect perfection.)

One of the things that it cannot optimise is which enabled action to choose as a key action, if many are available. In our example, it would have been possible to choose t_1 or t_6 instead of t_3 . Because t_6 may be disabled by t_4 , which is disabled until t_2 occurs, which is disabled until t_1 occurs, the choice of t_6 introduces the edges $t_6 \rightsquigarrow t_4 \rightsquigarrow t_2 \rightsquigarrow t_1$. The resulting $r(s)$ would be $\{t_1\}$, because t_1 is enabled and does not compete for tokens with any other transition. That is, the algorithm is clever enough to drop t_6 in favour of t_1 , but not clever enough to drop t_3 in favour of t_1 .

The linear time algorithm discussed above makes all enabled actions in $r(s)$ its key actions. Some other stubborn set methods than the deadlock detection method exploit this (e.g., [Val17]), so it is a good idea to make it show in the conditions. Therefore, an alternative to **D2w** has been defined that says that all enabled actions in $r(s)$ must be key actions. To avoid choosing $r(s) = \emptyset$ when there are enabled actions, yet another condition **D0** is introduced.

D0: If $enabled(s) \neq \emptyset$, then $r(s) \cap enabled(s) \neq \emptyset$.

D2: Every enabled action in $r(s)$ is its key action in s .

Clearly **D0** and **D2** together imply **D2w**, and **D2w** implies **D0**. Methods that build on **D2** are called *strong* stubborn set methods, while those only assuming **D2w** are *weak*.

Please remember that the set Act of all actions is partitioned to the set $\mathcal{I} \subseteq Act$ of invisible actions and the set $Act \setminus \mathcal{I}$ of visible actions. We recall how **D1** was used in the proof of Theorem 3.3. The full LSTS contains the path $s_0 \xrightarrow{a_1 \dots a_i} s_i$ where $s_0 \in S_r$, $a_i \in r(s)$ and $a_j \notin r(s)$ for $1 \leq j < i$. **D1** implies the existence of s'_0 and the path $s_0 \xrightarrow{a_i} s'_0 \xrightarrow{a_1 \dots a_{i-1}} s_i$ such that $s_0 \xrightarrow{a_i}_r s'_0$. This pattern repeats in many proofs in the stubborn set theory. The following condition guarantees that when using the pattern, the projection of the action sequence on the visible actions does not change.

V: If $r(s)$ contains an enabled visible action, then it contains all visible actions.

Lemma 3.4. *Assume that **D1** yields $\rho = s \xrightarrow{a_i a_1 \dots a_{i-1}} s'$ from $\pi = s \xrightarrow{a_1 \dots a_{i-1} a_i} s'$. If **V** holds, then $\text{vis}(\pi) = \text{vis}(\rho)$.*

Proof. If a_i is invisible, then $\text{vis}(a_i a_1 \dots a_{i-1}) = \text{vis}(a_1 \dots a_{i-1}) = \text{vis}(a_1 \dots a_{i-1} a_i)$. From now on assume that a_i is visible. Because **D1** only applies to $s \xrightarrow{a_1 \dots a_{i-1} a_i} s'$ when $a_i \in r(s)$, $r(s)$ contains an enabled visible action. By **V**, $r(s)$ contains all visible actions. Because none of a_1, \dots, a_{i-1} is in $r(s)$, they must be invisible. So $\text{vis}(a_i a_1 \dots a_{i-1}) = a_i = \text{vis}(a_1 \dots a_{i-1} a_i)$. \square

The application of Lemma 3.4 to the proof of Theorem 3.3 yields the following.

Theorem 3.5. *Assume that each $r(s)$ obeys **D1**, **D2w** and **V**. If $s \in S_r$ and s_n is a deadlock in TS , then for all paths $\pi = s \xrightarrow{a_1 \dots a_n} s_n$, there is a path $\rho = s \xrightarrow{b_1 \dots b_n} s_n$ such that $\text{vis}(\pi) = \text{vis}(\rho)$.*

This theorem almost gives item (1) of the list in Section 2.3. What is missing is that the path $s \xrightarrow{b_1 \dots b_n} s_n$ is deadlocking. It is, because $\rightarrow_r \subseteq \rightarrow$, so $\text{enabled}_{TS_r}(s_n) \subseteq \text{enabled}_{TS}(s_n) = \emptyset$. Item (2) is next to trivial. If $s \xrightarrow{b_1 \dots b_n} s_n$ is deadlocking, then $s \xrightarrow{b_1 \dots b_n} s_n$, and s_n is a deadlock by **D2w**.

We now have sufficient background on stubborn sets to illustrate the inconsistent labelling problem, but insufficient background to illustrate it in a street-credible context. Therefore, we continue and develop the LTL_X -preserving stubborn set method in full, and postpone the illustration of the inconsistent labelling problem to Section 4.

3.3. Infinite Paths. In the remainder of this paper, we will assume that the reduced LSTS is finite. This assumption is needed to make the next lemma hold in the presence of non-deterministic actions. It will be used in proving that each infinite path in TS maps to an infinite path in TS_r with certain properties.

Lemma 3.6. *Assume that $r(s_0)$ obeys **D1**, **D2w** and **V**, and the reduced LSTS is finite. Let $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots$ be any path where none of the a_i is in $r(s_0)$. Then there is a path $\rho = s_0 \xrightarrow{a_{\text{key}}} s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots$ for some action a_{key} . If, furthermore, a_{key} is visible, then all the a_i are invisible.*

Proof. We use König's Lemma type of reasoning [Kön27]. Let $a_{\text{key}} \in r(s_0)$ be some key action for $r(s_0)$. Its existence follows from **D2w**. By the key action property there are $s'_{0,0}, s'_{1,1}, \dots$ such that $s_i \xrightarrow{a_{\text{key}}} s'_{i,i}$. If a_{key} is visible, then **V** and $a_n \notin r(s_0)$ for $n \geq 1$ imply that a_1, a_2, \dots are invisible. By **D1**, for each i and each $0 \leq j < i$ there are $s'_{i,j}$ such that $s_0 \xrightarrow{a_{\text{key}}} s'_{i,0} \xrightarrow{a_1} s'_{i,1} \xrightarrow{a_2} \dots \xrightarrow{a_i} s'_{i,i}$. See Figure 5. We prove by induction that for every k , there is s'_k such that $s_0 \xrightarrow{a_{\text{key}}} s'_0$ (for $k = 0$) or $s'_{k-1} \xrightarrow{a_k} s'_k$ (for $k > 0$), and $s'_k = s'_{i,k}$ for infinitely many values of i .

Because there are only finitely many states, there is a state s'_0 that is the same as $s'_{i,0}$ for infinitely many values of i . This constitutes the base case.

To prove the induction step, we observe that all or all but one of the infinitely many i with $s'_{i,k} = s'_k$ satisfy $i > k$, and thus have an $s'_{i,k+1}$ such that $s'_k \xrightarrow{a_{k+1}} s'_{i,k+1}$. Infinitely many of these $s'_{i,k+1}$ are the same state, again because there are only finitely many states. This state qualifies as s'_{k+1} . \square

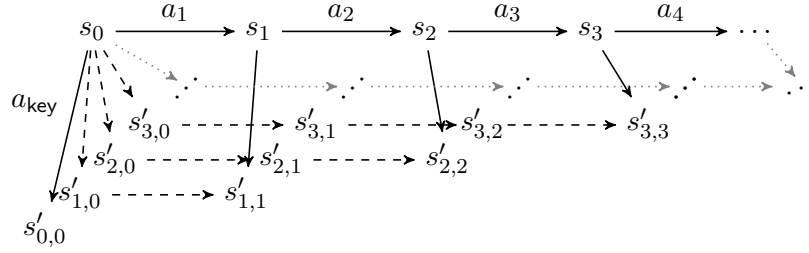


Figure 5. Illustration of the proof of Lemma 3.6. Vertical transitions are labelled with a_{key} ; dashed transitions have been obtained by applying **D1**.

Lemma 3.7. *Assume that each $r(s)$ obeys **D1**, **D2w** and **V**, and the reduced LSTS is finite. If $s_0 \in S_r$ and $\pi = s_0 \xrightarrow{a_1 a_2 \dots}$, then there is a path $\rho = s_0 \xrightarrow{b_1 b_2 \dots}_r$ such that $\text{vis}(\pi)$ is a prefix of $\text{vis}(\rho)$ or $\text{vis}(\rho)$ is a prefix of $\text{vis}(\pi)$.*

Proof. We use induction to prove, for each $i > 0$, the existence of b_i and s_i such that $s_{i-1} \xrightarrow{b_i}_r s_i$, and for each $i \geq 0$, the existence of a path $s_i \xrightarrow{a_{i,1} a_{i,2} \dots}$ and either

- (1) $\text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i \xrightarrow{a_{i,1} a_{i,2} \dots}) = \text{vis}(\pi)$, or
- (2) $a_{i,1}, a_{i,2}, \dots$ are invisible and $\text{vis}(\pi)$ is a prefix of $\text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i)$.

The base case $i = 0$ of the induction is obtained by choosing $s_0 \xrightarrow{a_0,1 a_0,2 \dots} = \pi$. Thanks to $b_1 \dots b_0 = \varepsilon$, (1) holds.

Regarding the induction step, if at least one of $a_{i,1}, a_{i,2}, \dots$ is in $r(s_i)$, then **D1** can be applied to the first such $a_{i,j}$, yielding $s_i \xrightarrow{a_{i,j}}_r s_{i+1} \xrightarrow{a_{i,1} \dots a_{i,j-1} a_{i,j+1} \dots}$. This specifies s_{i+1} , and we choose $b_{i+1} = a_{i,j}$ and $a_{i+1,1} a_{i+1,2} \dots = a_{i,1} \dots a_{i,j-1} a_{i,j+1} \dots$. We call this “moving $a_{i,j}$ to the front”. If (1) holds, then we apply Lemma 3.4 and the induction hypothesis to deduce

$$\begin{aligned} & \text{vis}(s_0 \xrightarrow{b_1 \dots b_{i+1}} s_{i+1} \xrightarrow{a_{i+1,1} a_{i+1,2} \dots}) \\ &= \text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i \xrightarrow{a_{i,j}} s_{i+1} \xrightarrow{a_{i,1} \dots a_{i,j-1} a_{i,j+1} \dots}) \\ &\stackrel{(L3.4)}{=} \text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i \xrightarrow{a_{i,1} a_{i,2} \dots}) \\ &\stackrel{(IH)}{=} \text{vis}(\pi) \end{aligned}$$

Therefore, $i + 1$ satisfies (1). Otherwise (2) holds, implying $\{b_{i+1}, a_{i+1,1}, a_{i+1,2}, \dots\} = \{a_{i,1}, a_{i,2}, \dots\} \subseteq \mathcal{I}$ and $\text{vis}(s_0 \xrightarrow{b_1 \dots b_{i+1}} s_{i+1}) = \text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i)$, of which $\text{vis}(\pi)$ is a prefix. Thus $i + 1$ satisfies (2).

In the opposite case none of $a_{i,1}, a_{i,2}, \dots$ is in $r(s_i)$. By **D2w**, $r(s_i)$ contains at least one key action. To present later a further result, we choose an invisible key action if available, and otherwise a visible one. Lemma 3.6 yields s_{i+1} such that $s_i \xrightarrow{a_{\text{key}}}_r s_{i+1} \xrightarrow{a_{i,1} a_{i,2} \dots}$. We choose $b_{i+1} = a_{\text{key}}$ and $a_{i+1,1} a_{i+1,2} \dots = a_{i,1} a_{i,2} \dots$. We call this “introducing a key action”. If $a_{\text{key}} \in \mathcal{I}$, then the equations

$$\begin{aligned} \text{vis}(s_0 \xrightarrow{b_1 \dots b_{i+1}} s_{i+1} \xrightarrow{a_{i+1,1} a_{i+1,2} \dots}) &= \text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i \xrightarrow{a_{i,1} a_{i,2} \dots}) \\ \text{vis}(s_0 \xrightarrow{b_1 \dots b_{i+1}} s_{i+1}) &= \text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i) \end{aligned}$$

both hold, so (1) or (2) remains valid in the step from i to $i + 1$. Otherwise, a_{key} is visible. Lemma 3.6 says that $a_{i,1}, a_{i,2}, \dots$ are invisible. Then both (1) and (2) imply that $\text{vis}(\pi)$ is a prefix of $\text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i)$, which is a (proper) prefix of $\text{vis}(s_0 \xrightarrow{b_1 \dots b_{i+1}} s_{i+1})$. Thus also $i + 1$ satisfies (2).

If (1) holds for every $i \geq 0$, then $\text{vis}(\rho)$ is a prefix of $\text{vis}(\pi)$. Otherwise there is i such that (2) holds. For that and every bigger i , $\text{vis}(\pi)$ is a prefix of $\text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i)$. Therefore, $\text{vis}(\pi)$ is a prefix of $\text{vis}(\rho)$. \square

The above result is a step towards item (3) of the list in Section 2.3, but not sufficient as such. Instead, $\text{vis}(\pi) = \text{vis}(\rho)$ is needed. We next add a condition, *viz.* condition **I**, guaranteeing that $\text{vis}(\rho)$ is a prefix of $\text{vis}(\pi)$. Then we add another condition (*viz.* **L**) for the opposite direction.

I: If an invisible action is enabled, then $r(s)$ contains an invisible key action.

Lemma 3.8. *If **I** is added to the assumptions of Lemma 3.7, then $\text{vis}(\rho)$ is a prefix of $\text{vis}(\pi)$.*

Proof. Consider the proof of Lemma 3.7. By Lemma 3.6, when none of $a_{i,1}, a_{i,2}, \dots$ is in $r(s_i)$, then either a_{key} or $a_{i,1}$ is invisible. Obviously $a_{i,1}$ is enabled in $r(s_i)$. So **I** guarantees that there is an invisible key action. This makes (1) remain true throughout the proof of Lemma 3.7, from which the claim follows. \square

Both **V** and **I** are easy to take into account in \rightsquigarrow -based algorithms for computing strong stubborn sets. It is much harder to ensure that $\text{vis}(\pi)$ is a prefix of $\text{vis}(\rho)$. The following condition is more or less the best known. It is usually implemented by constructing the reduced LSTS in depth-first order so that cycles can be recognised, and using a set that contains all visible actions as $r(s)$ in one or the other end of the edge that closes the cycle.

L: For every visible action a , every cycle in the reduced LSTS contains a state s such that $a \in r(s)$.

Lemma 3.9. *If **L** is added to the assumptions of Lemma 3.7, then $\text{vis}(\pi)$ is a prefix of $\text{vis}(\rho)$.*

Proof. To derive a contradiction, assume that $\text{vis}(\pi)$ is not a prefix of $\text{vis}(\rho)$. By Lemma 3.7, $\text{vis}(\rho)$ is a proper prefix of $\text{vis}(\pi)$. Therefore, $\text{vis}(\rho)$ is finite, that is, there is i such that $\text{vis}(\rho) = \text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i)$. These contradict (2) in the proof of the lemma, so (1) holds. By it and the proper prefix property, there is v such that $a_{i,v}$ is visible. We use the smallest such v .

Observe that if **D1** is applied at s_i to move action $a_{i,j}$ to the front, where $j > v$, or **D2w** is applied, then $a_{i+1,k} = a_{i,k}$ for $1 \leq k \leq v$. If the same also happens at s_{i+1} then $a_{i+2,k} = a_{i,k}$ for $1 \leq k \leq v$, and so on, either forever or until **D1** is applied such that $j \leq v$, whichever comes first. We show next that the latter comes first.

Because S_r is finite, we may let $n = i + |S_r|$. By the pigeonhole principle, s_i, \dots, s_n cannot all be distinct. So the path $s_i \xrightarrow{b_{i+1} \dots b_n} s_n$ contains a cycle. **L** implies that there is $i \leq \ell < n$ such that $a_{i,v} \in r(s_\ell)$. This guarantees that there is the smallest h such that $i \leq h < i + |S_r|$ and $\{a_{i,1}, \dots, a_{i,v}\} \cap r(s_h) \neq \emptyset$. Observe that at any step $i \leq i' < h$, whether **D1** is applied to move $a_{i',j}$ forward, where $j > v$, or **D2w** is applied to introduce a key action, we have $a_{i'+1,v} = a_{i',v}$. By **D1**, b_{h+1} is one of $a_{h,1}, \dots, a_{h,v}$. So either $b_{h+1} = a_{i,v}$ or $a_{i,v} = a_{h+1,v-1}$.

Repeating the argument at most v times proves that there is $i \leq h < i + v|S_r|$ such that $b_{h+1} = a_{i,v}$. Because $a_{i,v}$ is visible, this contradicts $\text{vis}(\rho) = \text{vis}(s_0 \xrightarrow{b_1 \dots b_i} s_i)$. \square

We have proven the following.

Theorem 3.10. *Assume that each $r(s)$ obeys **D1**, **D2w**, **V**, **I** and **L**. For all $s \in S_r$ and $\pi = s \xrightarrow{a_1 a_2 \dots}$, there is a path $\rho = s \xrightarrow{b_1 b_2 \dots}_r$ such that $\text{vis}(\pi) = \text{vis}(\rho)$.*

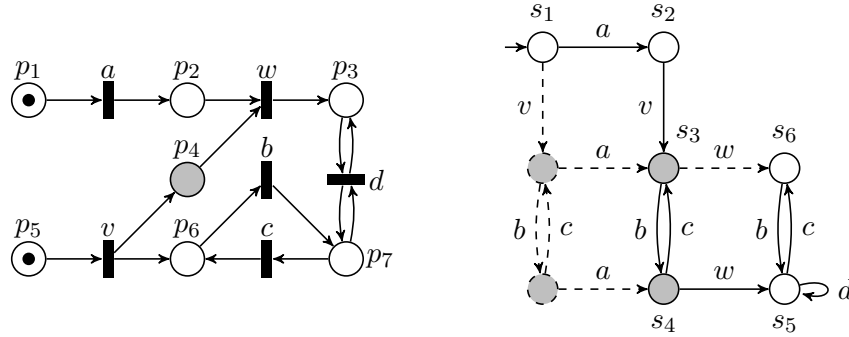


Figure 6. Example of a Petri net and its corresponding LSTS, which is reduced under **D1**, **D2w**, **V**, **I** and **L**.

This theorem gives item (3) of the list in Section 2.3. Item (4) follows immediately from $\rightarrow_r \subseteq \rightarrow$. We have proven items (1) to (4) of the list in Section 2.3. Before we continue with an example of the conditions at work, we restate them for convenience.

- D0** : If $enabled(s) \neq \emptyset$, then $r(s) \cap enabled(s) \neq \emptyset$.
- D1** : For all states s_1, \dots, s_n, s'_n and all $a \in r(s)$ and $a_1 \notin r(s), \dots, a_n \notin r(s)$, if $s \xrightarrow{a_1} \dots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$, then there are states $s', s'_1, \dots, s'_{n-1}$ such that $s \xrightarrow{a} s' \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s'_n$.
- D2** : Every enabled action in $r(s)$ is its key action in s .
- D2w**: If $enabled(s) \neq \emptyset$, then $r(s)$ contains a key action in s .
- V** : If $r(s)$ contains an enabled visible action, then it contains all visible actions.
- I** : If an invisible action is enabled, then $r(s)$ contains an invisible key action.
- L** : For every visible action a , every cycle in the reduced LSTS contains a state s such that $a \in r(s)$.

Recall that weak stubborn sets assume that conditions **D1**, **D2w**, **V**, **I** and **L** hold for all $r(s)$, while strong stubborn sets assume **D0**, **D1**, **D2**, **V**, **I** and **L** for all $r(s)$.

3.4. An Example. Consider the Petri net and its LSTS in Figure 6. We choose $AP = \{q\}$, and $L(m) = \{q\}$ if and only if $m(p_4) > 0$ (otherwise $L(m) = \emptyset$), and illustrate this choice with grey colour on p_4 and on those states where q holds. The dashed states and transitions are present in the original LSTS, but not in the reduced version. Other LSTSs later in this paper are visualised in a similar way.

Actions v and w must be declared visible, because they may change the truth value of q (v from false to true and w in the opposite direction). In the LSTS such events manifest themselves as transitions whose one end state is white and the opposite end state is grey, labelled with v or w . Please notice that not every occurrence of a visible action must change the truth value. For instance, if there were initially two tokens in p_5 , then both $\hat{m} \xrightarrow{avvw}$ and $\hat{n} \xrightarrow{avvw}$ would be possible, the first one inducing the label sequence $\emptyset\emptyset\{q\}\{q\}\{q\}$, and the second $\emptyset\emptyset\{q\}\emptyset\{q\}$.

Actions a , b , c and d may be invisible. In this case, we choose the set of invisible actions to be maximal, *i.e.*, $\mathcal{I} = \{a, b, c, d\}$. In the initial state s_1 , we have $r(s_1) = \{a\}$. Remark that a is a key action in s_1 , since for all prefixes π of $v(bc)^\omega$, we have $s_1 \xrightarrow{\pi a}$. That is, $\{a\}$

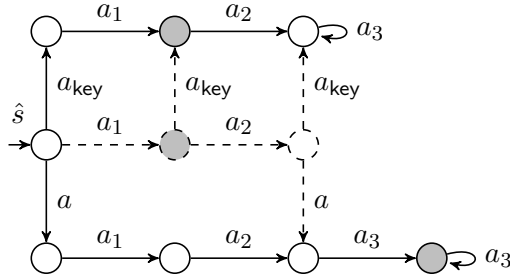


Figure 7. Counter-example showing that stubborn sets do not preserve stutter-trace equivalence. Grey states are labelled with $\{q\}$. The dashed transitions and states are not present in the reduced LSTS.

satisfies **D2w** in s_1 . It also satisfies **D1**, because it is easy to check that for those π and s for which $s_1 \xrightarrow{\pi a} s$ holds, also $s_1 \xrightarrow{a\pi} s$ holds.

In states s_3 and s_4 we must have $b \in r(s_3)$, respectively $c \in r(s_4)$, by condition **I**. Condition **L** can be satisfied in the cycle consisting of s_3 and s_4 by either setting $w \in r(s_3)$ or $w \in r(s_4)$; here we have opted for the latter. Actually, $w \in r(s_4)$ is also enforced by **D1**, since we have $s_4 \xrightarrow{wdc} s_6$ and $s_4 \xrightarrow{wd} s_5$, but not $s_4 \xrightarrow{c wd} s_6$ or $s_4 \xrightarrow{dw} s_5$. Consequently, $\{c\}$ and $\{c, d\}$ are not stubborn sets in s_4 .

4. COUNTER-EXAMPLE

Consider the LSTS in Figure 7, which we will refer to as TS^C . There is only one atomic proposition q , which holds in the grey states and is false in the other states. The initial state \hat{s} is marked with an incoming arrow. First, note that this LSTS is deterministic. The actions a_1, a_2 and a_3 are visible and a and a_{key} are invisible.

In the initial state, we choose $r(\hat{s}) = \{a, a_{key}\}$, which is a weak stubborn set by the following reasoning. Conditions **D2w** and **I** are satisfied, since a_{key} is an invisible key action in \hat{s} . The path $\hat{s} \xrightarrow{a_1 a_2}$ commutes with both a and a_{key} (and $\hat{s} \xrightarrow{a_1}$ furthermore commutes with a_{key}), satisfying **D1**. Conditions **V** and **L** are trivially true. In all other states s , we choose $r(s) = Act$.

As a result, we obtain a reduced LSTS TS_r^C that does not contain the dashed states and transitions. The original LSTS contains the trace $\emptyset\{q\}\emptyset\{q\}^\omega$, obtained by following the path with actions $a_1 a_2 a a_3^\omega$. However, the reduced LSTS does not contain a stutter equivalent trace. This is also witnessed by the LTL $_X$ formula $\Box(q \Rightarrow \Box(q \vee \Box\neg q))$, which holds for TS_r^C , but not for TS^C .

A very similar example can be used to show that strong stubborn sets suffer from the same problem. Consider again the LSTS in Figure 7, but assume that $a = a_{key}$, making the LSTS no longer deterministic. Now, $r(\hat{s}) = \{a\}$ is a strong stubborn set: **D0** is satisfied because $r(s) \cap enabled(s) = \{a\}$ and **D2** and **I** are satisfied because a is an invisible key action. Condition **D1** holds as well, since there is path $\hat{s} \xrightarrow{a a_1 a_2} s'$ (resp $\hat{s} \xrightarrow{a a_1} s'$) for every path of the shape $\hat{s} \xrightarrow{a_1 a_2 a} s'$ (resp. $\hat{s} \xrightarrow{a_1 a} s'$). Conditions **V** and **L** are trivially true as before. Again, the trace $\emptyset\{q\}\emptyset\{q\}^\omega$ is not preserved in the reduced LSTS. In Section 5.3, we will see why the inconsistent labelling problem does not occur for deterministic systems under strong stubborn sets.

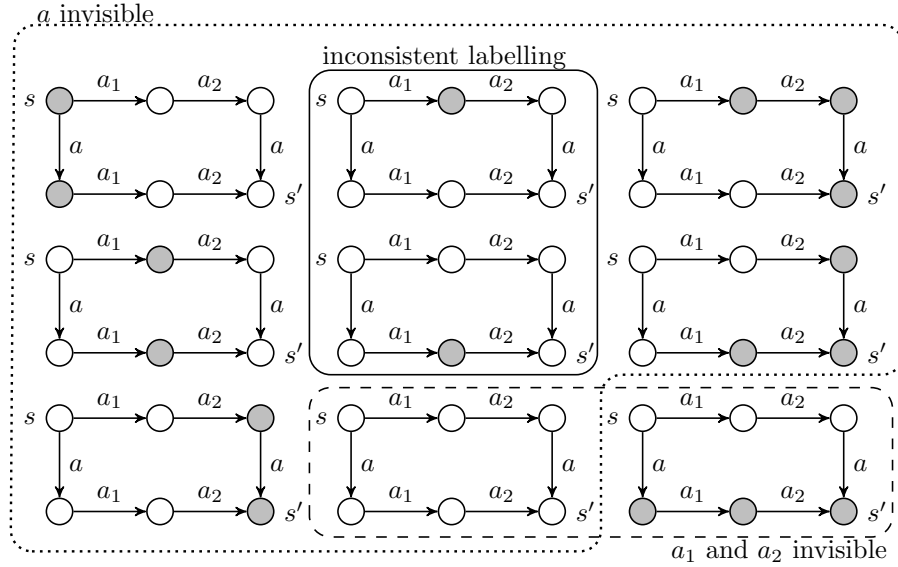


Figure 8. Nine possible scenarios when $a \in r(s)$ and $a_1 \notin r(s), a_2 \notin r(s)$, according to conditions **D1** and **V**. The dotted and dashed lines indicate when a or a_1, a_2 are invisible, respectively.

The core of the problem lies in the fact that condition **D1**, even when combined with **V**, does not enforce that the two paths it considers are stutter equivalent. Consider the paths $s \xrightarrow{a}$ and $s \xrightarrow{a_1 a_2 a}$ and assume that $a \in r(s)$ and $a_1 \notin r(s), a_2 \notin r(s)$. Condition **V** ensures that at least one of the following two holds: (i) a is invisible, or (ii) a_1 and a_2 are invisible. Half of the possible scenarios are depicted in Figure 8; the other half are symmetric. Again, the grey states (and only those states) are labelled with $\{q\}$.

The two cases delimited with a solid line are problematic. In both LSTSs, the paths $s \xrightarrow{a_1 a_2 a}$ and $s \xrightarrow{a a_1 a_2}$ are weakly equivalent, since a is invisible. However, they are not stutter equivalent, and therefore these LSTSs are not labelled consistently. The topmost of these two LSTSs forms the core of the counter-example TS^C , with the rest of TS^C serving to satisfy condition **D2/D2w**.

5. STRENGTHENING CONDITION D1

To fix the issue with inconsistent labelling, we propose to strengthen condition **D1** as follows¹.

¹Based on a comment by one of the journal’s reviewers, we noticed that condition **D1’** can be weakened further, by changing the last sentence to: “Furthermore, if none of a_1, \dots, a_n is visible, then $s_i \xrightarrow{a} s'_i$ for every $1 \leq i < n$.” This weakening additionally allows a reduction in the bottom-middle LSTS of Figure 8, although this is hard to exploit in practice (see Section 5.2). However, given the nature of this study, we chose to not make any last-minute changes to avoid making new mistakes. This choice was further motivated by the following remark by another reviewer (for which we are grateful): “I really carefully checked all the results and proofs and can accept the arguments and conclusions.”

D1': For all states s_1, \dots, s_n, s'_n and all $a \in r(s)$ and $a_1 \notin r(s), \dots, a_n \notin r(s)$, if $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$, then there are states $s', s'_1, \dots, s'_{n-1}$ such that $s \xrightarrow{a} s' \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s'_n$. Furthermore, if a is invisible, then $s_i \xrightarrow{a} s'_i$ for every $1 \leq i < n$.

On top of what is stated in **D1**, the new condition **D1'** requires the presence of intermediate vertical transitions $s_i \xrightarrow{a} s'_i$ whenever a is invisible. In this case, **V** implies that a is invisible and, consequently, the presence of transitions $s_i \xrightarrow{a} s'_i$ implies $L(s_i) = L(s'_i)$. Thus, condition **D1'** provides a form of *local* consistent labelling. Hence, the problematic cases of Figure 8 are resolved; a correctness proof is given below.

Condition **D1'** is very similar to condition **C1** [GKPP99], which is common in the context of ample sets. However, **C1** requires that action a is *globally* independent of each of the actions a_1, \dots, a_n , while **D1'** merely requires a kind of *local* independence. Persistent sets [God96] also rely on a condition similar to **D1'**, and require local independence. Thus, under ample sets and persistent sets, the vertical transitions $s_i \xrightarrow{a} s'_i$ are always present, and hence they do not suffer from the inconsistent labelling problem.

5.1. Correctness. To show that **D1'** indeed resolves the inconsistent labelling problem, we amend the lemmata and proofs of Section 3. The core of the revised argument lies in a new version of Lemma 3.4 that relates the state labels of the two paths considered by **D1'**.

Lemma 5.1. *Assume that **D1'** yields $\rho = s_0 \xrightarrow{a} s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s'_n$ from $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$. If **V** holds, then $\pi \triangleq \rho$.*

Proof. If a is invisible, then **D1'** enforces that $s_i \xrightarrow{a} s'_i$ for every $1 \leq i < n$. Thus, we have $L(s_i) = L(s'_i)$ for $1 \leq i \leq n$ and $\pi \triangleq \rho$ follows. From now on assume that a is visible. Because **D1'** only applies if $a \in r(s)$, $r(s)$ contains an enabled visible action. By **V**, $r(s)$ contains all visible actions. Because none of a_1, \dots, a_n is in $r(s)$, they must be invisible and we have $L(s_0) = L(s_1) = \dots = L(s_n)$ and $L(s'_0) = L(s'_1) = \dots = L(s'_n)$. So the traces of π and ρ are $L(s_0)^{n+1}L(s'_n)$ and $L(s_0)L(s'_n)^{n+1}$, respectively. We conclude that $\pi \triangleq \rho$. \square

We use the same reasoning to derive the existence of a transition $s_k \xrightarrow{a_{\text{key}}} s'_k$ for every $k > 0$ in the proof of Lemma 3.6, which yields the stronger result that, if a_{key} is invisible, $\text{no-stut}(\pi) = \text{no-stut}(\rho)$. The other lemmata are changed by replacing every occurrence of *vis* by *no-stut*. Furthermore, in the proof of Lemma 3.9, we reason about a visible action $a_{i,v}$ that actually changes the state labelling. This results in the following two theorems that replace Theorems 3.5 and 3.10 respectively.

Theorem 5.2. *Assume that each $r(s)$ obeys **D1'**, **D2w** and **V**. If $s \in S_r$, s_n is a deadlock in TS , then for all paths $\pi = s \xrightarrow{a_1 \dots a_n} s_n$, there is a path $\rho = s \xrightarrow{b_1 \dots b_n} s_n$ such that $\text{no-stut}(\pi) = \text{no-stut}(\rho)$.*

Theorem 5.3. *Assume that each $r(s)$ obeys **D1'**, **D2w**, **V**, **I** and **L**. For all $s \in S_r$ and $\pi = s \xrightarrow{a_1 a_2 \dots} s_n$, there is a path $\rho = s \xrightarrow{b_1 b_2 \dots} s_n$ such that $\text{no-stut}(\pi) = \text{no-stut}(\rho)$.*

With $\rightarrow_r \subseteq \rightarrow$, it follows immediately that the replacement of condition **D1** by **D1'** is sufficient to ensure the reduced transition system TS_r is stutter-trace equivalent to the original transition system TS . Thus, the problem with Theorem 2.5 is resolved.

5.2. Implementation. As discussed in Section 3.2, most, if not all, implementations of stubborn sets approximate **D1** based on a binary relation \rightsquigarrow on actions. This relation may even (partly) depend on the current state s , in which case we write \rightsquigarrow_s , and it should be such that condition **D1** is satisfied whenever $a \in r(s)$ and $a \rightsquigarrow_s a'$ together imply $a' \in r(s)$. A set satisfying **D0**, **D1**, **D2**, **V** and **I** or **D1**, **D2w**, **V** and **I** can be found by searching for a suitable *strongly connected component* in the graph $(Act, \rightsquigarrow_s)$. Condition **L** is dealt with by other techniques.

Practical implementations construct \rightsquigarrow_s by analysing how any two actions a and a' interact. If a is enabled, the simplest (but not necessarily the best possible) strategy is to make $a \rightsquigarrow_s a'$ if and only if a and a' access at least one place (in the case of Petri nets) or variable (in the more general case) in common. This can be relaxed, for instance, by not considering commutative accesses, such as writing to and reading from a FIFO buffer. As a result, \rightsquigarrow_s can only detect reduction opportunities in (sub)graphs of the shape

$$\begin{array}{ccccccc} s & \xrightarrow{a_1} & s_1 & - \dots - & s_{n-1} & \xrightarrow{a_n} & s_n \\ \downarrow a & & \downarrow a & & \downarrow a & & \downarrow a \\ s' & \xrightarrow{a_1} & s'_1 & - \dots - & s'_{n-1} & \xrightarrow{a_n} & s'_n \end{array}$$

where $a \in r(s)$ and $a_1 \notin r(s), \dots, a_n \notin r(s)$. The presence of the vertical a transitions in s_1, \dots, s_{n-1} implies that **D1'** is also satisfied by such implementations.

5.3. Deterministic LSTSs. As already noted in Section 4, strong stubborn sets for deterministic systems do not suffer from the inconsistent labelling problem. The following lemma, which also appeared as [Val17, Lemma 4.2], shows why.

Lemma 5.4. *For deterministic LSTSs, conditions **D1** and **D2** together imply **D1'**.*

Proof. Let TS be a deterministic LSTS, $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$ a path in TS and r a reduction function that satisfies **D1** and **D2**. Furthermore, assume that $a \in r(s_0)$ and $a_1 \notin r(s_0), \dots, a_n \notin r(s_0)$. By applying **D1**, we obtain the path $\pi' = s_0 \xrightarrow{a} s'_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} s'_n$, which satisfies the first part of condition **D1'**. With **D2**, we have $s_i \xrightarrow{a} s'_i$ for every $1 \leq i \leq n$. Then, we can also apply **D1** to every path $s_0 \xrightarrow{a_1} \dots \xrightarrow{a_i} s_i \xrightarrow{a} s'_i$ to obtain, for all $1 \leq i \leq n$, paths $\pi_i = s_0 \xrightarrow{a} s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots \xrightarrow{a_i} s'_i$. Since TS is deterministic, every path π_i must coincide with a prefix of π' . We conclude that $s'_i = s'_i$ and so the requirement that $s_i \xrightarrow{a} s'_i$ for every $1 \leq i \leq n$ is also satisfied. \square

6. SAFE LOGICS

In this section, we will identify two logics, *viz.* reachability and CTL_X , which are not affected by the inconsistent labelling problem. This is either due to their limited expressivity or the additional POR conditions that are required on top of the conditions we have introduced so far.

6.1. Reachability properties. Although the counter-example of Section 4 shows that stutter-trace equivalence is in general not preserved by stubborn sets, some fragments of LTL_{-X} are preserved. One such class of properties is reachability properties, which are of the shape $\Box f$ or $\Diamond f$, where f is a formula not containing temporal operators.

Theorem 6.1. *Let TS be an LSTS, r a reduction function that satisfies either **D0**, **D1**, **D2**, **V** and **L** or **D1**, **D2w**, **V** and **L** and TS_r the reduced LSTS. For all possible labellings $l \subseteq AP$, TS contains an initial path to a state s such that $L(s) = l$ iff TS_r contains an initial path to a state s' such that $L(s') = l$.*

Proof. The “if” case is trivial, since TS_r is a subgraph of TS . For the “only if” case, we reason as follows. Let $TS = (S, \rightarrow, \hat{s}, L)$ be an LSTS and $\pi = s_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_n$ an initial path, *i.e.*, $s_0 = \hat{s}$. We mimic this path by repeatedly taking some enabled action a that is in the stubborn set, according to the following schema. Below, we assume the path to be mimicked contains at least one visible action. Otherwise, its first state would have the same labelling as s_n .

- (1) If there is an i such that $a_i \in r(s_0)$, we consider the smallest such i , *i.e.*, $a_1 \notin r(s_0), \dots, a_{i-1} \notin r(s_0)$. Then, we can shift a_i forward by **D1**, move towards s_n along $s_0 \xrightarrow{a_i} s'_0$ and continue by mimicking $s'_0 \xrightarrow{a_1} \dots \xrightarrow{a_{i-1}} s_i \xrightarrow{a_{i+1}} \dots \xrightarrow{a_n} s_n$.
- (2) If all of $a_1 \notin r(s_0), \dots, a_n \notin r(s_0)$, then, by **D0** and **D2** or by **D2w**, there is a key action a_{key} in s_0 . By the definition of key actions and **D1**, a_{key} leads to a state s'_0 from which we can continue mimicking the path $s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s'_n$. Note that $L(s_n) = L(s'_n)$, since a_{key} is invisible by condition **V**.

The second case cannot be repeated infinitely often, due to condition **L**. Hence, after a finite number of steps, we reach a state s'_n with $L(s'_n) = L(s_n)$. \square

We remark that more efficient mechanisms for reachability checking under POR have been proposed, such as condition **S** [VH17], which can replace **L**, or conditions based on *up-sets* [Sch00]. Another observation is that model checking of LTL_{-X} properties can be reduced to reachability checking by computing the cross-product of a Büchi automaton and an LSTS [BK08], in the process resolving the inconsistent labelling problem. Peled [Pel96] shows how this approach can be combined with POR, but please note the correctness issues detailed in [Sie19].

6.2. Deterministic LSTSs and CTL_{-X} Model Checking. In this section, we consider the inconsistent labelling problem in the setting of CTL_{-X} model checking. When applying stubborn sets in that context, stronger conditions are required to preserve the branching structure that CTL_{-X} reasons about. Namely, the original LSTS must be deterministic and one more condition needs to be added [GKPP99]:

C4: Either $r(s) = Act$ or $r(s) \cap enabled(s) = \{a\}$ for some $a \in Act$.

We slightly changed its original formulation to match the setting of stubborn sets. A weaker condition, called **Ä8**, which does not require determinism of the whole LSTS is proposed in [Val97]. With **C4**, strong and weak stubborn sets collapse, as shown by the following lemma.

Lemma 6.2. *Conditions **D2w** and **C4** together imply **D0** and **D2**.*

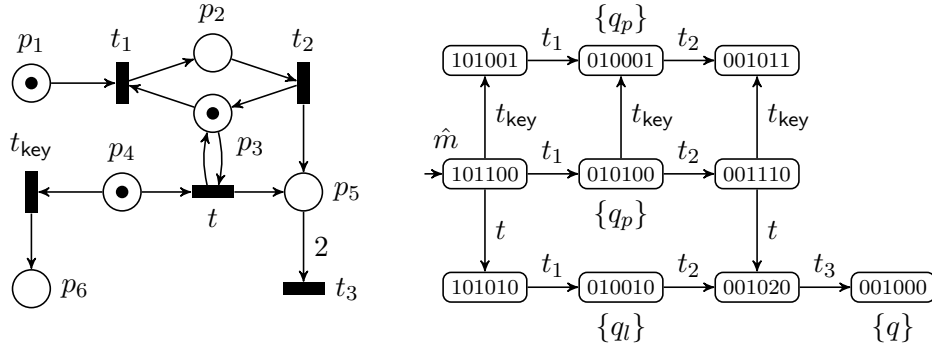


Figure 9. Example of a Petri net whose LSTS suffers from the inconsistent labelling problem.

Proof. Let TS be an LSTS, s a state and r a reduction function that satisfies **D2w** and **C4**. Condition **D0** is trivially implied by **C4**. Using **C4**, we distinguish two cases: either $r(s)$ contains precisely one enabled action a , or $r(s) = Act$. In the former case, this single action a must be a key action, according to **D2w**. Hence, **D2**, which requires that all enabled actions in $r(s)$ are key actions, is satisfied. Otherwise, if $r(s) = Act$, we consider an arbitrary action a that satisfies **D2**'s precondition that $s \xrightarrow{a}$. Given a path $s \xrightarrow{a_1 \dots a_n}$, the condition that $a_1 \notin r(s), \dots, a_n \notin r(s)$ only holds if $n = 0$. We conclude that **D2**'s condition $s \xrightarrow{a_1 \dots a_n a}$ is satisfied by the assumption $s \xrightarrow{a}$. \square

It follows from Lemmata 5.4 and 6.2 and Theorems 5.2 and 5.3 that CTL_X model checking of deterministic systems with stubborn sets does not suffer from the inconsistent labelling problem. The same holds for condition **A8**, as already shown in [Val97].

7. PETRI NETS

In this section, we discuss the impact of the inconsistent labelling problem on Petri nets. Contrary to Section 2.2, here we assume the LSTS of a Petri net has the set of all markings \mathcal{M} as its set of states. This does not affect the correctness of POR, as long as the set of reachable states \mathcal{M}_{reach} is finite. As before, we assume that the LSTS contains some labelling function $L : \mathcal{M} \rightarrow 2^{AP}$. More details on how a labelling function arises from a Petri net are given below. Like in the Petri net examples we saw earlier, markings and structural transitions take over the role of states and actions respectively. Note that the LSTS of a Petri net is deterministic. We want to stress that all the theory in this section is specific for the semantics defined in Section 2.2.

Example 7.1. Consider the Petri net with initial marking \hat{m} on left of Figure 9. Here, all arcs are weighted 1, except for the arc from p_5 to t_3 , which is weighted 2. Its LSTS is infinite, but the substructure reachable from \hat{m} is depicted on the right. The number of tokens in each of the places p_1, \dots, p_6 is inscribed in the nodes, the state labels (if any) are written beside the nodes.

The LSTS practically coincides with the counter-example of Section 4. Only the self-loops are missing and the state labelling, with atomic propositions q , q_p and q_i , differs slightly; the latter will be explained later. For now, note that t and t_{key} are invisible and

that the trace $\emptyset\{q_p\}\emptyset\emptyset\{q\}$, which occurs when firing transitions $t_1t_2tt_3$ from \hat{m} , can be lost when reducing with weak stubborn sets. \square

In the remainder of this section, we fix a Petri net (P, T, W, \hat{m}) and its LSTS $(\mathcal{M}, \rightarrow, \hat{m}, L)$. Below, we consider three different types of atomic propositions. Firstly, polynomial propositions [BJLM19] are of the shape $f(p_1, \dots, p_n) \bowtie k$ where f is a polynomial over p_1, \dots, p_n , $\bowtie \in \{<, \leq, >, \geq, =, \neq\}$ and $k \in \mathbb{Z}$. Such a proposition holds in a marking m iff $f(m(p_1), \dots, m(p_n)) \bowtie k$. A linear proposition [LW19] is similar, but the function f over places must be linear and $f(0, \dots, 0) = 0$, *i.e.*, linear propositions are of the shape $k_1p_1 + \dots + k_np_n \bowtie k$, where $k_1, \dots, k_n, k \in \mathbb{Z}$. Finally, we have arbitrary propositions [Var05], whose shape is not restricted and which can hold in any given set of markings.

Several other types of atomic propositions can be encoded as polynomial propositions. For example, *fireable*(t) [BJLM19, LW19], which holds in a marking m iff t is enabled in m , can be encoded as $\prod_{p \in P} \prod_{i=0}^{W(p,t)-1} (p-i) \geq 1$. The proposition *deadlock*, which holds in markings where no structural transition is enabled, does not require special treatment in the context of POR, since it is already preserved by **D1** and **D2w**. The sets containing all linear and polynomial propositions are henceforward called AP_l and AP_p , respectively. The corresponding labelling functions are defined as $L_l(m) = L(m) \cap AP_l$ and $L_p(m) = L(m) \cap AP_p$ for all markings m . Below, the two stutter equivalences \triangleq_{L_l} and \triangleq_{L_p} that follow from the new labelling functions are abbreviated \triangleq_l and \triangleq_p , respectively. Note that $AP \supseteq AP_p \supseteq AP_l$ and $\triangleq \subseteq \triangleq_p \subseteq \triangleq_l$.

For the purpose of introducing several variants of invisibility, we reformulate and generalise the definition of invisibility from Section 2. Given an atomic proposition $q \in AP$, a relation $\mathcal{R} \subseteq \mathcal{M} \times \mathcal{M}$ is *q-invisible* if and only if $(m, m') \in \mathcal{R}$ implies $q \in L(m) \Leftrightarrow q \in L(m')$. We consider a structural transition t *q-invisible* iff its corresponding relation $\{(m, m') \mid m \xrightarrow{t} m'\}$ is *q-invisible*. Invisibility is also lifted to sets of atomic propositions: given a set $AP' \subseteq AP$, relation \mathcal{R} is *AP'-invisible* iff it is *q-invisible* for all $q \in AP'$. If \mathcal{R} is *AP-invisible*, we plainly say that \mathcal{R} is *invisible*. *AP'-invisibility* and *invisibility* carry over to structural transitions. We sometimes refer to invisibility as *ordinary invisibility* for emphasis. Note that the set of invisible structural transitions \mathcal{I} is no longer an under-approximation, but contains exactly those structural transitions t for which $m \xrightarrow{t} m'$ implies $L(m) = L(m')$ (cf. Section 2).

We are now ready to introduce three orthogonal variations on invisibility.

Definition 7.2. Let $\mathcal{R} \subseteq \mathcal{M} \times \mathcal{M}$ be a relation on markings. Then,

- \mathcal{R} is *reach q-invisible* [VH17] iff $\mathcal{R} \cap (\mathcal{M}_{reach} \times \mathcal{M}_{reach})$ is *q-invisible*; and
- \mathcal{R} is *value q-invisible* iff
 - $q = (f(p_1, \dots, p_n) \bowtie k)$ is polynomial and for all pairs of markings $(m, m') \in \mathcal{R}$, we have that $f(m(p_1), \dots, m(p_n)) = f(m'(p_1), \dots, m'(p_n))$; or
 - q is not polynomial and \mathcal{R} is *q-invisible*.

Intuitively, under reach *q-invisibility*, all pairs of reachable markings $(m, m') \in \mathcal{R}$ have to agree on the labelling of q . For value invisibility, the value of the polynomial f must never change between two markings $(m, m') \in \mathcal{R}$. Reach and value invisibility are lifted to structural transitions and sets of atomic propositions as before, *i.e.*, by taking $\mathcal{R} = \{(m, m') \mid m \xrightarrow{t} m'\}$ when considering invisibility of t .

Definition 7.3. A structural transition t is *strongly q-invisible* iff the set $\{(m, m') \mid \forall p \in P : m'(p) = m(p) + W(t, p) - W(p, t)\}$ is *q-invisible*.

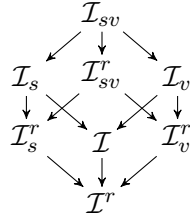


Figure 10. Lattice of sets of invisible actions. Arrows represent a subset relation.

Strong invisibility does not take the presence of a transition $m \xrightarrow{t} m'$ into account, and purely reasons about the effects of t . Value invisibility and strong invisibility are new in the current work, although strong invisibility was inspired by the notion of invisibility that is proposed by Varpaaniemi in [Var05]. Our definition of strong invisibility weakens the conditions of Varpaaniemi.

We indicate the sets of all value, reach and strongly invisible structural transitions with \mathcal{I}_v , \mathcal{I}^r and \mathcal{I}_s respectively. Since $\mathcal{I}_v \subseteq \mathcal{I}$, $\mathcal{I}_s \subseteq \mathcal{I}$ and $\mathcal{I} \subseteq \mathcal{I}^r$, the set of all their possible combinations forms the lattice shown in Figure 10. In the remainder, the weak equivalence relations that follow from each of the eight invisibility notions are abbreviated, *e.g.*, $\sim_{\mathcal{I}_{sv}^r}$ becomes \sim_{sv}^r .

Example 7.4. Consider again the Petri net and LSTS from Example 7.1. We can define q_l and q_p as linear and polynomial propositions, respectively:

- $q_l := p_3 + p_4 + p_6 = 0$ is a linear proposition, which holds when neither p_3 , p_4 nor p_6 contains a token. Structural transition t is q_l -invisible, because $m \xrightarrow{t} m'$ implies that $m(p_3) = m'(p_3) \geq 1$, and thus neither m nor m' is labelled with q_l . On the other hand, t is not value q_l -invisible (by the transition 101100 \xrightarrow{t} 101010) or strongly reach q_l -invisible (by 010100 and 010010). However, t_{key} is strongly value q_l -invisible: it moves a token from p_4 to p_6 and hence never changes the value of $p_3 + p_4 + p_6$.
- $q_p := (1 - p_3)(1 - p_5) = 1$ is a polynomial proposition, which holds in all reachable markings m where $m(p_3) = m(p_5) = 0$ or $m(p_3) = m(p_5) = 2$. Structural transition t is reach value q_p -invisible, but not q_p -invisible (by 002120 \xrightarrow{t} 002030) or strongly reach q_p -invisible. Strong value q_p -invisibility of t_{key} follows immediately from the fact that the adjacent places of t_{key} , *viz.* p_4 and p_6 , do not occur in the definition of q_p .

This yields the state labelling which is shown in Example 7.1. □

Given a weak equivalence relation R_{\sim} and a stutter equivalence relation $R_{\underline{\Delta}}$, we write $R_{\sim} \preceq R_{\underline{\Delta}}$ to indicate that R_{\sim} and $R_{\underline{\Delta}}$ yield consistent labelling (Definition 2.4). We spend the rest of this section investigating under which notions of invisibility and propositions from the literature, the LSTS of a Petri net is labelled consistently. More formally, we check for each weak equivalence relation R_{\sim} and each stutter equivalence relation $R_{\underline{\Delta}}$ whether $R_{\sim} \preceq R_{\underline{\Delta}}$. This tells us when existing stubborn set theory can be applied without problems. The two lattices containing all weak and stuttering equivalence relations are depicted in Figure 11; each dotted arrow represents a consistent labelling result. Before we continue, we first introduce an auxiliary lemma.

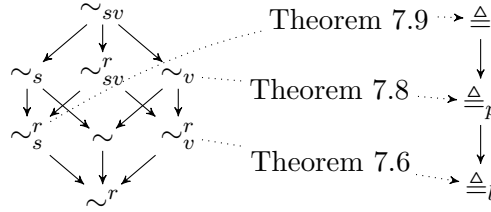


Figure 11. Two lattices containing variations of weak equivalence and stutter equivalence, respectively. Solid arrows indicate a subset relation inside the lattice; dotted arrows follow from the indicated theorems and show when the LSTS of a Petri net is labelled consistently.

Lemma 7.5. *Let I be a set of invisible structural transitions and L some labelling function. If for all $t \in I$ and paths $\pi = m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots$ and $\pi' = m_0 \xrightarrow{t} m'_0 \xrightarrow{t_1} m'_1 \xrightarrow{t_2} \dots$, it holds that $\pi \triangleq_L \pi'$, then $\sim_I \preceq \triangleq_L$.*

Proof. We assume that the following holds for all paths and $t \in I$:

$$m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots \triangleq_L m_0 \xrightarrow{t} m'_0 \xrightarrow{t_1} m'_1 \xrightarrow{t_2} \dots \quad (\dagger)$$

To prove $\sim_I \preceq \triangleq_L$, we need to consider two initial paths π and π' such that $\pi \sim_I \pi'$ and prove that $\pi \triangleq_L \pi'$ (see Definition 2.4). The proof proceeds by induction on the combined number of invisible structural transitions (taken from I) in π and π' . In the base case, π and π' contain only visible structural transitions, and $\pi \sim_I \pi'$ implies $\pi = \pi'$ since Petri nets are deterministic. Hence, $\pi \triangleq_L \pi'$.

For the induction step, we take as hypothesis that, for all initial paths π and π' that together contain at most k invisible structural transitions, $\pi \sim_I \pi'$ implies $\pi \triangleq_L \pi'$. Let π and π' be two arbitrary initial paths such that $\pi \sim_I \pi'$ and the total number of invisible structural transitions contained in π and π' is k . We consider the case where an invisible structural transition is introduced in π' , the other case is symmetric. Let $\pi' = \sigma_1 \sigma_2$ for some σ_1 and σ_2 . Let $t \in I$ be some invisible structural transition and $\pi'' = \sigma_1 t \sigma'_2$ such that σ_2 and σ'_2 contain the same sequence of structural transitions. Clearly, we have $\pi' \sim_I \pi''$. Here, we can apply our original assumption (\dagger) , to conclude that $\sigma_2 \triangleq_L \sigma'_2$, i.e., the extra stuttering step t thus does not affect the labelling of the remainder of π'' . Hence, we have $\pi' \triangleq_L \pi''$ and, with the induction hypothesis, $\pi \triangleq_L \pi''$. Note that π and π'' together contain $k + 1$ invisible structural transitions.

In case π and π' together contain an infinite number of invisible structural transitions, $\pi \sim_I \pi'$ implies $\pi \triangleq_L \pi'$ follows from the fact that the same holds for all finite prefixes of π and π' that are related by \sim_I . \square

The following theorems each focus on a class of atomic propositions and show which notion of invisibility is required for the LSTS of a Petri net to be labelled consistently. In the proofs, we use a function d_t , defined as $d_t(p) = W(t, p) - W(p, t)$ for all places p , which indicates how structural transition t changes the state. Furthermore, we also consider functions of type $P \rightarrow \mathbb{N}$ as vectors of type $\mathbb{N}^{|P|}$. This allows us to compute the pairwise addition of a marking m with d_t ($m + d_t$) and to indicate that t does not change the marking ($d_t = 0$).

Theorem 7.6. *Under reach value invisibility, the LSTS underlying a Petri net is labelled consistently for linear propositions, i.e., $\sim_v^r \preceq \triangleq_L$.*

Proof. Let $t \in \mathcal{I}_v^r$ be a reach value invisible structural transition such that there exist reachable markings m and m' with $m \xrightarrow{t} m'$. If such a t does not exist, then \sim_v^r is the reflexive relation and $\sim_v^r \preceq \triangle_l$ is trivially satisfied. Otherwise, let $q := f(p_1, \dots, p_n) \bowtie k$ be a linear proposition. Since t is reach value invisible and f is linear, we have $f(m) = f(m') = f(m + d_t) = f(m) + f(d_t)$ and thus $f(d_t) = 0$. It follows that, given two paths $\pi = m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots$ and $\pi' = m_0 \xrightarrow{t} m'_0 \xrightarrow{t_1} m'_1 \xrightarrow{t_2} \dots$, the addition of t does not influence f , since $f(m_i) = f(m_i) + f(d_t) = f(m_i + d_t) = f(m'_i)$ for all i . As a consequence, t also does not influence q . With Lemma 7.5, we deduce that $\sim_v^r \preceq \triangle_l$. \square

Whereas in the linear case one can easily conclude that π and π' are stutter equivalent under f , in the polynomial case, we need to show that f is constant under all value invisible structural transitions t , even in markings where t is not enabled. This follows from the following proposition.

Proposition 7.7. *Let $f : \mathbb{N}^n \rightarrow \mathbb{Z}$ be a polynomial function, $a, b \in \mathbb{N}^n$ two constant vectors and $c = a - b$ the difference between a and b . Assume that for all $x \in \mathbb{N}^n$ such that $x \geq b$, where \geq denotes pointwise comparison, it holds that $f(x) = f(x + c)$. Then, f is constant in the vector c , i.e., $f(x) = f(x + c)$ for all $x \in \mathbb{N}^n$.*

Proof. Let f , a , b and c be as above and let $\mathbf{1} \in \mathbb{N}^n$ be the vector containing only ones. Given some arbitrary $x \in \mathbb{N}^n$, consider the function $g_x(t) = f(x + t \cdot \mathbf{1} + c) - f(x + t \cdot \mathbf{1})$. For sufficiently large t , it holds that $x + t \cdot \mathbf{1} \geq b$, and it follows that $g_x(t) = 0$ for all sufficiently large t . This can only be the case if g_x is the zero polynomial, i.e., $g_x(t) = 0$ for all t . As a special case, we conclude that $g_x(0) = f(x + c) - f(x) = 0$. \square

The intuition behind this is that $f(x + c) - f(x)$ behaves like the directional derivative of f with respect to c . If the derivative is equal to zero in infinitely many x , f must be constant in the direction of c . We will apply this result in the following theorem.

Theorem 7.8. *Under value invisibility, the LSTS underlying a Petri net is labelled consistently for polynomial propositions, i.e., $\sim_v \preceq \triangle_p$.*

Proof. Let $t \in \mathcal{I}_v$ be a value invisible structural transition, m and m' two markings with $m \xrightarrow{t} m'$, and $q := f(p_1, \dots, p_n) \bowtie k$ a polynomial proposition. Note that infinitely many such (not necessarily reachable) markings exist in \mathcal{M} , so we can apply Proposition 7.7 to obtain $f(m) = f(m + d_t)$ for all markings m . It follows that, given two paths $\pi = m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots$ and $\pi' = m_0 \xrightarrow{t} m'_0 \xrightarrow{t_1} m'_1 \xrightarrow{t_2} \dots$, the addition of t does not alter the value of f , since $f(m_i) = f(m_i + d_t) = f(m'_i)$ for all i . As a consequence, t also does not change the labelling of q . Application of Lemma 7.5 yields $\sim_v \preceq \triangle_p$. \square

Varpaaniemi shows that the LSTS of a Petri net is labelled consistently for arbitrary propositions under his notion of invisibility [Var05, Lemma 9]. Our notion of strong visibility, and especially strong reach invisibility, is weaker than Varpaaniemi's invisibility, so we generalise the result to $\sim_s^r \preceq \triangle$.

Theorem 7.9. *Under strong reach visibility, the LSTS underlying a Petri net is labelled consistently for arbitrary propositions, i.e., $\sim_s^r \preceq \triangle$.*

Proof. Let $t \in \mathcal{I}_s^r$ be a strongly reach invisible structural transition and $\pi = m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots$ and $\pi' = m_0 \xrightarrow{t} m'_0 \xrightarrow{t_1} m'_1 \xrightarrow{t_2} \dots$ two paths. Since, $m'_i = m_i + d_t$ for all i , it holds that either (i) $d_t = 0$ and $m_i = m'_i$ for all i ; or (ii) each pair (m_i, m'_i) is contained in $\{(m, m') \mid \forall p \in P : m'(p) = m(p) + W(t, p) - W(p, t)\}$, which is the set that underlies strong

reach invisibility of t . In both cases, $L(m_i) = L(m'_i)$ for all i . It follows from Lemma 7.5 that $\sim_s^r \subseteq \triangle$. \square

To show that the results of the above theorems cannot be strengthened, we provide two negative results.

Theorem 7.10. *Under ordinary invisibility, the LSTS underlying a Petri net is not necessarily labelled consistently for arbitrary propositions, i.e., $\sim \not\subseteq \triangle$.*

Proof. Consider the Petri net from Example 7.1 with the arbitrary proposition q_l . Disregard q_p for the moment. Structural transition t is q_l -invisible, hence the paths corresponding to $t_1t_2tt_3$ and $tt_1t_2t_3$ are weakly equivalent under ordinary invisibility. However, they are not stutter equivalent. \square

Theorem 7.11. *Under reach value invisibility, the LSTS underlying a Petri net is not necessarily labelled consistently for polynomial propositions, i.e., $\sim_v^r \not\subseteq \triangle_p$.*

Proof. Consider the Petri net from Example 7.1 with the polynomial proposition $q_p := (1 - p_3)(1 - p_5) = 1$ from Example 7.4. Disregard q_l in this reasoning. Structural transition t is reach value q_p -invisible, hence the paths corresponding to $t_1t_2tt_3$ and $tt_1t_2t_3$ are weakly equivalent under reach value invisibility. However, they are not stutter equivalent for polynomial propositions. \square

It follows from Theorems 7.10 and 7.11 and transitivity of \subseteq that Theorems 7.6, 7.8 and 7.9 cannot be strengthened further. In terms of Figure 11, this means that the dotted arrows cannot be moved downward in the lattice of weak equivalences and cannot be moved upward in the lattice of stutter equivalences. The implications of these findings on related work will be discussed in the next section.

8. RELATED WORK

There are many works in the literature that apply stubborn sets. We will consider several works that aim to preserve LTL_{-X} and discuss whether they are correct when it comes to the inconsistent labelling problem. Furthermore, we also identify several unrelated issues.

Liebke and Wolf [LW19] present an approach for efficient CTL model checking on Petri nets. For some formulas, they can reduce CTL model checking to LTL model checking, which allows greater reductions under POR. They rely on the incorrect LTL preservation theorem, and since they apply the techniques on Petri nets with ordinary invisibility, their theory is incorrect (Theorem 7.10). Similarly, the overview of stubborn set theory presented by Valmari and Hansen in [VH17] applies reach invisibility and does not necessarily preserve LTL_{-X} . Varpaaniemi [Var05] also applies stubborn sets to Petri nets, but relies on a visibility notion that is stronger than strong invisibility. The correctness of these results is thus not affected (Theorem 7.9).

A generic implementation of weak stubborn sets for the LTSmin model checker is proposed by Laarman *et al.* [LPvdPH16]. They use abstract concepts such as guards and transition groups to implement POR in a way that is agnostic of the input language. The theory they present includes condition **D1**, which is too weak and thus incorrect, but the accompanying implementation follows the framework of Section 5.2, and thus it is correct by Theorems 5.2 and 5.3. The implementations proposed in [VH17, Wol18] are similar, albeit specific for Petri nets.

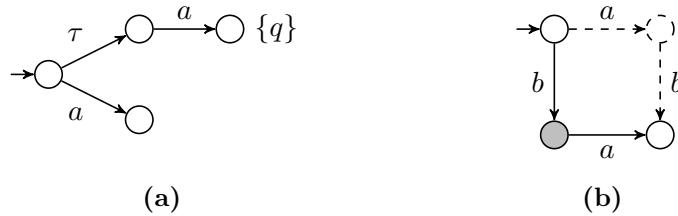


Figure 12. Counter-examples for theories in two related works.

Several works [GRHRW15, HLL⁺14] perform action-based model checking and thus strive to preserve weak trace equivalence or inclusion. As such, they do not suffer from the problems discussed here, which applies only to state labels. Other recent work [DL16] relies on ample sets, and is thus not affected, or only considers safety properties [Laa18].

Although Beneš *et al.* [BBČ⁺09, BBB⁺11] rely on ample sets, and not on stubborn sets, they also discuss weak trace equivalence and stutter-trace equivalence. In fact, they present an equivalence relation for traces that is a combination of weak and stutter equivalence. The paper includes a theorem that weak equivalence implies their new state/event equivalence [BBB⁺11, Theorem 6.5]. However, the counter-example in Figure 12a shows that this consistent labelling theorem does not hold. Here, the action τ is invisible, and the two paths in this transition system are thus weakly equivalent. However, they are not stutter equivalent, which is a special case of state/event equivalence. Although the main POR correctness result [BBB⁺11, Corollary 6.6] builds on the incorrect consistent labelling theorem, its correctness does not appear to be affected. An alternative proof can be constructed based on the reasoning presented in Section 5.1.

Bønneland *et al.* [BJLM19] apply stubborn-set based POR to two-player Petri nets, and their reachability semantics expressed as a *reachability game*. Since their approach only concerns reachability, it is not affected by the inconsistent labelling problem (see Section 6). Unfortunately, their POR theory is nevertheless unsound, contrary to what is claimed in [BJLM19, Theorem 17]. In reachability games, player 1 tries to reach one of the *goal* states, while player 2 tries to avoid them. Bønneland *et al.* propose a condition **R** that guarantees that all goal states in the full game are also reachable in the reduced game. However, the reverse is not guaranteed: paths that do not contain a goal state are not necessarily preserved, essentially endowing player 1 with more power. Consider the (solitaire) reachability game depicted in Figure 12b, in which all edges belong to player 2 and the only goal state is indicated with grey. Player 2 wins the non-reduced game by avoiding the goal state via the edges labelled with a and then b . However, $\{b\}$ is a stubborn set—according to the conditions of [BJLM19]—in the initial state, and the dashed transitions are thus eliminated in the reduced game. Hence, player 2 is forced to move the token to the goal state and player 1 wins in the reduced game. In the mean time, the authors of [BJLM19] confirmed and resolved the issue in [BJL⁺21].

The current work is not the first to point out mistakes in POR theory. In [Sie19], Siegel presents a flaw in an algorithm that combines POR with ample sets and on-the-fly model checking [Pel96]. In that setting, POR is applied on the product of an LSTS and a Büchi automaton. We briefly sketch the issue here. Let q be a state of the LSTS and s a state of the Büchi automaton. While investigating a transition $(q, s) \xrightarrow{a} (q', s')$, condition **C3**, which—like condition **L**—aims to solve the action ignoring problem, incorrectly sets $r(q, s') = \text{enabled}(q)$

instead of $r(q, s) = \text{enabled}(q)$. The issue is repaired by setting $r(q, s) = \text{enabled}(q)$, but only for a certain subclass of Büchi automata.

The setting considered by Laarman and Wijs [LW14] is similar: they discuss how to apply stubborn sets during parallel nested depth-first search in the product of an LSTS and a Büchi automaton. Both the correctness argument and the implementation are based on [LPvdPH16], thus – by the discussion above – incorrect in theory, but correct in practice.

9. CONCLUSION

We discussed the inconsistent labelling problem for preservation of stutter-trace equivalence with stubborn sets. The issue is relatively easy to repair by strengthening condition **D1**. For Petri nets, altering the definition of invisibility can also resolve inconsistent labelling depending on the type of atomic propositions. The impact on applications presented in related works seems to be limited: the problem is typically mitigated in the implementation, since it is very hard to compute **D1** exactly. This is also a possible explanation for why the inconsistent labelling problem has not been noticed for so many years.

Since this is not the first error found in POR theory [Sie19], a more rigorous approach to proving its correctness, *e.g.* using proof assistants, would provide more confidence.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers, including those who reviewed the conference version, for their helpful comments. Special thanks go out to the two journal reviewers. The first reviewer provided many useful suggestions for improvement and noticed that condition **D1'** can be weakened (see the footnote in Section 5). The second reviewer took the significant effort to check all proofs, giving us more confidence in the publication.

REFERENCES

- [BBB⁺11] N. Beneš, L. Brim, B. Buhnova, I. Ern, J. Sochor, and P. Vařeková. Partial order reduction for state/event LTL with application to component-interaction automata. *Science of Computer Programming*, 76(10):877–890, 2011.
- [BBČ⁺09] Nikola Beneš, Lubos Brim, Ivana Černá, Jiri Sochor, Pavlina Vařeková, and Barbora Zimmerova. Partial Order Reduction for State/Event LTL. In *IFM 2009*, volume 5423 of *LNCS*, pages 307–321, 2009.
- [BJL⁺21] Frederik Meyer Bønneland, Peter Gjøøl Jensen, Kim Guldstrand Larsen, Marco Muñiz, and Jiří Srba. Stubborn Set Reduction for Two-Player Reachability Games. *Logical Methods in Computer Science*, 17(1):21:1–21:26, 2021.
- [BJLM19] Frederik Meyer Bønneland, Peter Gjøøl Jensen, Kim Guldstrand Larsen, and Marco Muñiz. Partial Order Reduction for Reachability Games. In *CONCUR 2019*, volume 140, pages 23:1–23:15, 2019.
- [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- [DL16] Iyaylo Dobrikov and Michael Leuschel. Optimising the ProB model checker for B using partial order reduction. *Formal Aspects of Computing*, 28(2):295–323, 2016.
- [GKPP99] Rob Gerth, Ruurd Kuiper, Doron Peled, and Wojciech Penczek. A Partial Order Approach to Branching Time Logic Model Checking. *Information and Computation*, 150(2):132–152, 1999.
- [God96] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems*, volume 1032 of *LNCS*. Springer, 1996.
- [GRHRW15] Thomas Gibson-Robinson, Henri Hansen, A. W. Roscoe, and Xu Wang. Practical Partial Order Reduction for CSP. In *NFM 2015*, volume 9058 of *LNCS*, pages 188–203, 2015.

- [HLL⁺14] Henri Hansen, Shang-wei Lin, Yang Liu, Truong Khanh Nguyen, and Jun Sun. Diamonds Are a Girl's Best Friend: Partial Order Reduction for Timed Automata with Abstractions. In *CAV 2014*, volume 8559 of *LNCS*, pages 391–406, 2014.
- [Kön27] Dénes König. Über eine Schlussweise aus dem Endlichen ins Unendliche. *Acta Sci. Math. (Szeged)*, 3(2-3):121–130, 1927.
- [Laa18] Alfons Laarman. Stubborn Transaction Reduction. In *NFM 2018*, volume 10811 of *LNCS*, pages 280–298, 2018.
- [LPvdPH16] Alfons Laarman, Elwin Pater, Jaco van de Pol, and Henri Hansen. Guard-based partial-order reduction. *International Journal on Software Tools for Technology Transfer*, 18(4):427–448, 2016.
- [LW14] Alfons Laarman and Anton Wijs. Partial-Order Reduction for Multi-core LTL Model Checking. In *HVC 2014*, volume 8855 of *LNCS*, pages 267–283, 2014.
- [LW19] Torsten Liebke and Karsten Wolf. Taking Some Burden Off an Explicit CTL Model Checker. In *Petri Nets 2019*, volume 11522 of *LNCS*, pages 321–341, 2019.
- [NVW20] Thomas Neele, Antti Valmari, and Tim A. C. Willemse. The Inconsistent Labelling Problem of Stutter-Preserving Partial-Order Reduction. In *FoSSaCS 2020*, volume 12077 of *LNCS*, pages 482–501, 2020.
- [NWW20] Thomas Neele, Tim A. C. Willemse, and Wieger Wesselink. Partial-Order Reduction for Parity Games with an Application on Parameterised Boolean Equation Systems. In *TACAS 2020*, volume 12079 of *LNCS*, pages 307–324, 2020.
- [Pel93] Doron Peled. All from One, One for All: on Model Checking Using Representatives. In *CAV 1993*, volume 697 of *LNCS*, pages 409–423, 1993.
- [Pel96] Doron Peled. Combining partial order reductions with on-the-fly model-checking. *Formal Methods in System Design*, 8(1):39–64, 1996.
- [Sch00] Karsten Schmidt. Stubborn sets for model checking the EF/AG fragment of CTL. *Fundamenta Informaticae*, 43(1-4):331–341, 2000.
- [Sie19] Stephen F. Siegel. What's Wrong with On-the-Fly Partial Order Reduction. In *CAV 2019*, volume 11562 of *LNCS*, pages 478–495, 2019.
- [Val88] Antti Valmari. Error detection by reduced reachability graph generation. In *APN 1988*, pages 95–112, 1988.
- [Val91a] Antti Valmari. A Stubborn Attack on State Explosion. In *CAV 1990*, volume 531 of *LNCS*, pages 156–165, 1991.
- [Val91b] Antti Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets*, volume 483, pages 491–515, 1991.
- [Val92] Antti Valmari. A Stubborn Attack on State Explosion. *Formal Methods in System Design*, 1(4):297–322, 1992.
- [Val96] Antti Valmari. The state explosion problem. In *ACPN 1996*, volume 1491 of *LNCS*, pages 429–528, 1996.
- [Val97] Antti Valmari. Stubborn Set Methods for Process Algebras. In *POMIV 1996*, volume 29 of *DIMACS*, pages 213–231, 1997.
- [Val17] Antti Valmari. Stop It, and Be Stubborn! *ACM Transactions on Embedded Computing Systems*, 16(2):46:1–46:26, 2017.
- [Var05] Kimmo Varpaaniemi. On Stubborn Sets in the Verification of Linear Time Temporal Properties. *Formal Methods in System Design*, 26(1):45–67, 2005.
- [VH11] Antti Valmari and Henri Hansen. Can stubborn sets be optimal? *Fundamenta Informaticae*, 113(3-4):377–397, 2011.
- [VH17] Antti Valmari and Henri Hansen. Stubborn Set Intuition Explained. In *ToPNoC XII*, volume 10470 of *LNCS*, pages 140–165, 2017.
- [Wol18] Karsten Wolf. Petri Net Model Checking with LoLA 2. In *Petri Nets 2018*, volume 10877 of *LNCS*, pages 351–362, 2018.