A Lazy Query Scheme for Reachability Analysis in Petri Nets

Loïg Jezequel, Didier Lime, Bastien Sérée

A lazy query scheme for reachability analysis in Petri nets

Loïg Jezequel[1,3], Didier Lime[2,3], and Bastien Sérée[2,3]

[1] Université de Nantes
[2] École Centrale de Nantes
[3] LS2N, UMR CNRS 6004, Nantes, France
firstname.lastname@ls2n.fr

Abstract. In recent works we proposed a lazy algorithm for reachability analysis in networks of automata. This algorithm is optimistic and tries to take into account as few automata as possible to perform its task. In this paper we extend the approach to the more general settings of reachability analysis in unbounded Petri nets and reachability analysis in bounded Petri nets with inhibitor arcs. We consider we are given a reachability algorithm and we organize queries to it on bigger and bigger nets in a lazy manner, trying thus to consider as few places and transitions as possible to make a decision. Our approach has been implemented in the Romeo model checker and tested on benchmarks from the model checking contest.

Keywords: Reachability analysis, unbounded Petri nets, inhibitor arcs, lazy algorithms

1 Introduction

In recent works [8,9] we proposed an algorithm for reachability analysis in networks of automata. This algorithm is called lazy as it tries to use as few automata as possible to complete its task. To that extent, it is a non-trivial instance of a general principle that has been implemented in many approaches (e.g. program slicing [21]). In practice, on many benchmarks this approach proved to be efficient: the LaRA tool (which implements our approach) used only a small portion of the automata in the network to conclude about reachability. Runtime comparisons with LoLA [22] (in a non-timed setting [8]) and Uppaal [2] (in a timed setting [9]) were also frequently in favor of LaRA.

Networks of automata are in fact a subclass of Petri nets as they can be syntactically transformed into safe Petri nets. Extending our lazy reachability algorithm to larger classes of Petri nets is thus a natural next step in our work. Moreover, it is of particular interest for us as it will allow to implement lazy reachability in the Romeo model checker [16], developed in our research team. In fact, Romeo works on models even more expressive than Petri nets, where reachability is not always decidable. In this paper we focus on unbounded Petri

nets and bounded Petri nets with inhibitor arcs, two subclasses of these models for which reachability is decidable.

Reachability analysis in Petri nets (or equivalent models such as vector addition systems) has been widely studied. The problem is known to be decidable in general [17,12,13,15]. Efficient techniques exist for performing it in the particular case of bounded nets, that is nets with a finite state space. One can notice, for example, Petri net unfolding [18,6] or variations around it [3,4], partial order techniques [7], and decision diagram based approaches [5,19].

Here we follow a different approach and do not propose a *standalone* reachability algorithm but rather, given such an algorithm, we propose a scheme to use its results on subnets that are built incrementally from the reachability property by adding only places and transitions that are required to make a decision. This is why we call the approach lazy. Compared to [8], the main challenges we address here are (1) that the *components* of the system are less well-defined in a Petri net than in a network of finite automata, and (2) that the state-space is infinite in general. Note also that even if a net is bounded, its subnets might not be. We propose an algorithm for reachability in plain Petri nets, and also show how to deal with inhibitor arcs in the bounded case.

This paper is organized as follows. We start by giving some definitions and notations in Section 2. Then we present our algorithm for lazy reachability analysis in Petri nets in Section 3 and show its validity. After that, we show how this approach can be transposed to perform reachability analysis for the class of bounded Petri nets with inhibitor arcs in Section 4. Finally, in Section 5 we report on an implementation of our algorithm in the model checker Romeo and give experimental results obtained from a run of our tool on all the benchmarks from the 2020 edition of the model checking contest [11,10].

2 Definitions and notations

We define Petri nets and their semantics, as well as the central notion of reachability of markings in Petri nets. Then, we define the notion of subnets and partial markings, that we use later to perform reachability analysis on a Petri net without considering it in its entirety.

2.1 Petri nets

Definition 1 (Petri net). *A* Petri net *is a tuple $N = (P, T, F, m_0)$ where P and T are disjoint finite sets of places and transitions respectively, $F : P \times T \cup T \times P \to \mathbb{N}$ is a flow function, and $m_0 : P \to \mathbb{N}$ is called the initial marking.*

In a net N, for any $x \in P \cup T$, we define ${}^\bullet x = \{y \ : \ F(y, x) \neq 0\}$ the *preset* of x and $x^\bullet = \{y \ : \ F(x, y) \neq 0\}$ the *postset* of x. We can extend this postset (resp preset) concept to subsets of P or T by doing the union of the postsets (resp presets) of each element of the considered subset.

In a net N, any function $m : P \to \mathbb{N}$ is called a *marking* of N. A transition $t \in T$ is *fireable* from a marking m if and only if $\forall p \in {}^\bullet t, m(p) \geq F(p, t)$. In this

case, firing t from m leads to the new marking m' such that $\forall p \in P, m'(p) = m(p) - F(p,t) + F(t,p)$. We denote it by $m \xrightarrow{t} m'$. Given a sequence $\omega = t_1, \ldots, t_n$ of transitions, we define $m \xrightarrow{\omega} m'$ if there exist markings $m_1, \ldots m_{n-1}$ such that $m \xrightarrow{t_1} m_1$, $\forall 2 \leq i \leq n-1, m_{i-1} \xrightarrow{t_i} m_i$, and $m_{n-1} \xrightarrow{t_n} m'$.

Definition 2 (Reachability). *A marking m is said to be* reachable *in N if and only if there exists a sequence of transitions ω such that $m_0 \xrightarrow{\omega} m$.*

Definition 3 (Boundedness). *A Petri net is said to be* k-bounded*, for a given k, if for every reachable marking m and every place p, we have $m(p) \leq k$. A Petri net is said to be* bounded*, if there exists a k such that it is k-bounded.*

2.2 Subnets and partial markings

In the following, we will perform reachability analysis on parts of a Petri net: not all the places and transitions of the net will be considered. This is formalized through the notion of subnet.

Definition 4 (Subnet). *A* Subnet *N' of a Petri net $N = (P, T, F, m_0)$ is a tuple (P', T', F', m'_0) such that $P' \subseteq P$, $T' \subseteq T$, $F' = F_{|P',T'}$, and $m'_0 = m_{0|P'}$.*

Given a subnet N' of a net N, and for any $x \in P' \cup T'$, we define ${}^{\bullet N}x = \{y \ : \ F(y,x) \neq 0)\}$ and $x^{N\bullet} = \{y \ : \ F(x,y) \neq 0\}$ (that is, intuitively, the preset and postset taken in N rather than in N').

We introduce two notions of completeness with respect to a net N for a subnet N'. They will be central in our algorithms and their proofs. The notion of P-completeness expresses that N' contains all the places from N that are used as preconditions for enabling transitions in N'.

Definition 5 (P-completeness). *A subnet N' of a net N is said to be* P-complete *when $\forall t \in T', {}^{\bullet N}t \subseteq P'$.*

In the other way around, the notion of T-completeness expresses that N' contains all the transitions from N that can add tokens on places in N'.

Definition 6 (T-completeness). *A subnet N' of a net N is said to be* T-complete *when $\forall p \in P', {}^{\bullet N}p \subseteq T'$.*

Partial marking will be used to express reachability objectives that do not concern all the places in a net.

Definition 7 (Partial marking). *For a Petri net $N = (P, T, F, m_0)$, any function $m_p : P \to \mathbb{N} \cup \{\star\}$ is called a* partial marking *of N.*

Intuitively, a partial marking is a marking which is not fully specified: when $m_p(p) = \star$ for some $p \in P$ it means that this value is left unspecified. For a partial marking m_p of a net N, we define $supp(m_p) = \{p \in P \ : \ m_p(p) \neq \star\}$. A marking m such that $m(p) = m_p(p)$ for any $p \in supp(m_p)$ is said to *realize*

m_p. We can notice that a every marking m of a net N is a partial marking such that $supp(m) = P$. For a net N with a subnet N', and a (partial) marking m of N, we denote by m' the (partial) marking of N' such that $m' = m_{|P'}$ and call it the *submarking* of m in N'.

Definition 8 (Reachability). *A partial marking m_p is said to be* reachable *in N if and only if there exists a marking m that realizes m_p and is reachable in N.*

Finally, a third notion of completeness, m-completeness, is defined for partial markings. It expresses the fact that, for a given marking m of N', all the transitions from N that can affect this marking by reducing its value for some place are included in N'.

Definition 9 (m-completeness). *A subnet N' of a net N is said to be* m-complete *with m a marking of N' when $\forall p \in supp(m), p^{N\bullet} \subseteq T'$.*

3 Lazy reachability analysis in Petri nets

In this section we propose an algorithm which, given a Petri net N and a marking m in N, decides whether or not m is reachable in N. However, this algorithm consists in a heuristic way to perform reachability queries on smaller nets, which proves more efficient in some cases than a query on the full net. It therefore requires an algorithm to perform reachability on Petri nets, used as a black-box. The technique works on unbounded nets, provided that the reachability black-box handles them.

We start by demonstrating the concept of our algorithm – in particular, in which way it is lazy – on two examples, then we formalize the algorithm, and finally we prove its validity.

3.1 Preliminary example

Consider the Petri net of Figure 1 – the places are represented by circles, the transitions by squares, the flow function by arrows, the initial marking by black dots. We look at two reachability questions on this net: (Q1) is it possible to reach a partial marking m_1 so that $m_1(p_2) = 1$, and $m_1(p_3) = 1$? (Q2) is it possible to reach a partial marking m_2 so that $m_2(p_4) = 3$?

Let us focus on (Q1) first. To this end, consider the subnets N_1 and N_2 of Figure 2. These two subnets were built from N by considering exactly the places in the support of m_1. If the initial marking of each of these subnets had been a submarking of m_1, then the answer to (Q1) would immediately be a yes. This is not the case however, so we cannot conclude yet.

Consider N_1 first. Our objective with this subnet is to find whether or not it is possible to reach some marking m so that $m(p_2) = 1$. As this is not the case initially, one needs to find how to increase the marking of p_2. This can only be done by using transitions t so that $p_2 \in t^\bullet$. We thus add these transitions to our

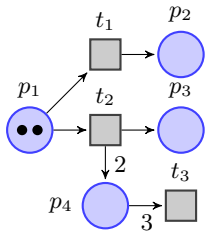

Fig. 1. A Petri net N.

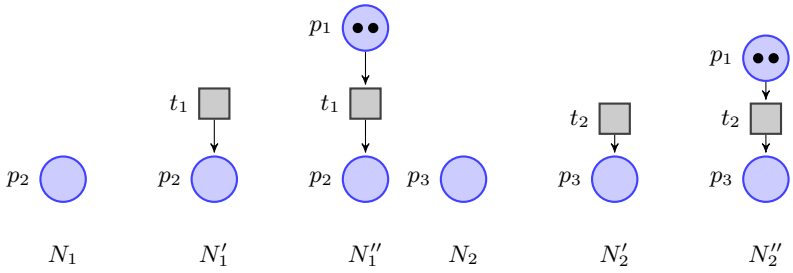

Fig. 2. Six subnets incrementally built from p_2 and p_3.

subnet, leading to N_1' (which is T-complete). Now, some m so that $m(p_2) = 1$ is reachable, however, we cannot yet conclude that this is the case in the full net N because we do not have all the presets of the transitions we use. So we add the places in these presets, leading to the new subnet N_1'' (which is P-complete). From this subnet, one can conclude that some m so that $m(p_2) = 1$ is reachable in N.

A similar process allows to build N_2'' and prove that some m so that $m(p_3) = 1$ is reachable in N. However, having obtained these two results does not guarantee that m_1 is reachable in N because N_1'' and N_2'' overlap (the place p_1 appears in both), which may lead to conflicts between the transition sequences found in these two subnets. We thus merge the subnets (on common places and transitions, here only p_1), which leads to the subnet of Figure 3. In this subnet m_1 is reachable. Moreover – as we prove later – because this subnet is P-complete, m_1 is also reachable in N.

The place p_4 and the transition t_3 were never included in the subnets considered. This is why we call our algorithm lazy: it omits the places and transitions that are not useful for its analysis.

Let us now focus on (Q2). In this case our analysis will start from the subnet N_3 of Figure 4. For similar reasons as before, we first add the transitions that can put tokens in p_4, leading to N_3'. In this subnet, no marking m so that $m(p_4) = 3$ is reachable (because there is always an even number of tokens in p_4). However, markings with $m(p_4) \geq 3$ are reachable. Hence, we add the transitions that can

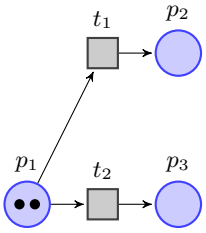

Fig. 3. Merging of N_1'' and N_2''.

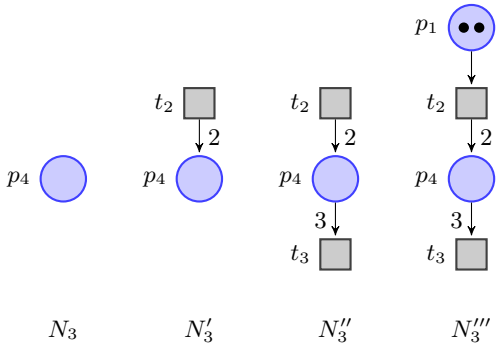

Fig. 4. Four subnets built from p_4.

remove tokens from p_4, leading to N_3''. In this subnet, such an m is reachable (for example by firing t_2 three times and then t_3 one time). As before, in order to conclude one needs to verify that (at least some of) the sequences allowing to reach such an m are fireable in the original net. For that, the places in the presets of t_2 and t_3 need to be added, leading to N_3'''. In this subnet it is not possible to fire t_2 three times. Moreover, this subnet is T-complete and so, as we prove later, if no m such that $m(p_4) = 3$ is reachable in N_3''', no such m is reachable at all in N.

3.2 An algorithm for lazy reachability analysis in Petri nets

The formalization of the ideas presented in the above examples leads to Algorithm 1[4]. It is a lazy algorithm that, given a net N and a (partial) marking m, tells whether or not m is reachable in N. This algorithm works on subnets of N. These subnets are identified by their sets of places and transitions.

Algorithm 1 starts with subnets built from a partition of the set of places involved in m. This allows to handle each part of the objective separately as long as they do not interact. Initially, each element of the List $LNets$ is a pair (P, T) representing one of these subnets. The algorithm does two main tasks: concretisation (addition of places and transitions to subnets) and merging (union of interacting subnets). At each iteration of its main loop it does at least one of those two tasks.

Concretisation. Concretisation consists in expanding one subnet. If the partial objective is not reachable one adds new transitions to add new ways to reach it. If the partial objective is reachable, one needs to add new places to ensure that

[4] It uses the classical list data structure. The length of a list L is given by $length(L)$. The k^{th} element of L is $L[k]$.

Algorithm 1 Lazy algorithm checking if a marking m is reachable in a Petri net $N = (P, T, F, m_0)$

```
1: choose a partition {P_1, ..., P_p} of supp(m)
2: LNets ← [(P_1, ∅), ..., (P_p, ∅)]
3: Complete ← false
4: Consistent ← true
5: while not Complete or not Consistent do
6:     Complete ← ∀k, LNets[k] is complete
7:     if not Complete then
8:         optional unless Consistent
9:             mayHaveSol ← Concretise(LNets, m)
10:            if not mayHaveSol then
11:                return false
12:            end if
13:        end option
14:    end if
15:    Consistent ← LNets is consistent
16:    if not Consistent then
17:        optional unless Complete
18:            Merge(LNets)
19:        end option
20:    end if
21: end while
22: return true
```

the transitions used in the subnet can also be used in the original net (i.e. to ensure that their full preset is taken into account).

More formally, the objective of the concretisation is to ensure that each subnet verifies the completeness notion of definition 10. If this is the case, then m is reachable in N (provided that there is not interaction with other subnets, which is ensured by the notion of consistency described below). If at least one subnet cannot be made complete, then it is granted that m is not reachable in N.

Definition 10. *Let $N = (P, T, F, m_0)$ be a Petri net and m a marking. A subnet N' of N is* complete *(with respect to N and m) if N' is P-complete and the submarking m' is reachable in N'.*

Remark that completeness can be effectively checked provided reachability can be checked. The rest of the conditions is syntactic.

Concretisation can be implemented as described in Algorithm 2, which alternately adds places and transitions to a subnet.

Merging. Merging consists in replacing two subnets in $LNets$ by a single subnet obtained by union of places and transitions sets. Merging is needed when two subnets share places, as in this case the solutions to the reachability problem

Algorithm 2 Auxiliary function $Concretise(LNets, m)$ for Algorithm 1

1: **choose** k such that $LNets[k]$ is not complete
2: $(P_k, T_k) \leftarrow LNets[k]$
3: $m' \leftarrow m_{|P_k}$
4: **if not** $Reachable(LNets[k], m')$ **then**
5: **choose** T'_k such that $T_k \subset T'_k \subseteq supp(m')^{N\bullet} \cup {}^{\bullet N}P_k$
6: **if not possible then**
7: **return** false
8: **else**
9: $LNets[k] \leftarrow (P_k, T'_k)$
10: **end if**
11: **else**
12: **choose** P'_k such that $P_k \subset P'_k \subseteq {}^{\bullet N}T_k$
13: $LNets[k] \leftarrow (P'_k, T_k)$
14: **end if**
15: **return** true

found in these subnets can interfere. For the same interference reason, merging is also needed when one of the subnets contains a transition whose postset contains a place involved in the submarking of m (the reachability objective) in another subnet. The fact that two subnets may interfere is formalised through the notion of consistency in Definition 11.

Definition 11. *The list of subnets* $LNets = [(P_1, T_1), ..., (P_n, T_n)]$ *is* consistent *if*

1. $\forall k \neq \ell, (P_k \cap P_\ell) = \emptyset$, *and*
2. $\forall k \neq \ell, {T_k}^{N\bullet} \cap supp(m_{|P_\ell}) = \emptyset$.

Remark that the definition of consistency is completely syntactic and can therefore be checked effectively.

3.3 Proof of the algorithm

We now prove the correctness of Algorithm 1. Propositions 1 and 2 together prove the soundness of Algorithm 1, while proposition 3 proves its completeness. The proofs of these propositions are based on lemma for which proofs are presented in this part.

Proposition 1. *If Algorithm 1 returns false, then* m *is not reachable in* N.

Proof. The only way for Algorithm 1 to return false is at line 11. It implies that the previous call to the Concretise function (Algorithm 2) returned false. This can only occur at line 7 of Algorithm 2.

In this case, it must not be possible to choose T' such that $T \subset T' \subseteq supp(m')^{N\bullet} \cup {}^{\bullet N}P$ (line 5). In other words, the subnet $LNets[k]$ considered is m'-complete (Definition 9) and T-complete (Definition 6). Moreover, the current

marking m' must not be reachable in the subnet $LNets[k]$ (line 4) and is a submarking of m (line 3). Hence, by applying Lemma 1 below, m is not reachable in N. □

Lemma 1. *Let N' be a subnet of a Petri net N. Assume that N' is T-complete. Let m' be a marking of N', that is not reachable. If N' is m'-complete, then no marking m of N such that m' is the submarking of m in N' is reachable in N.*

Proof. Let m be a reachable marking of N, with $m_0 \xrightarrow{\omega} m$. Denote by n the number of transitions in ω. We prove by induction on n that m' is reachable in N'. By the contrapositive, this proves the Lemma.

Induction hypothesis for all n, if m is reachable in N and there is a sequence ω of n transitions such that $m_0 \xrightarrow{\omega} m$, then the submarking m' of m in N' is reachable in N'.

Initialisation when $n = 0$, the only possible m is m_0 (it must be reachable with 0 transitions). Thus, simply take $m' = m'_0$, which is, by construction the submarking of m_0 in N' and is obviously reachable in N'.

Induction Let us consider some $n > 0$ and assume that the induction hypothesis is verified for $n-1$. First, remark that ω can always be split in two parts: ω' of length $n-1$ and t_n (a single transition) such that $m_0 \xrightarrow{\omega'} \tilde{m} \xrightarrow{t_n} m$ in N for some marking $\tilde{m}$. From the induction hypothesis, the submarking $\tilde{m}'$ of $\tilde{m}$ is reachable in N'. Four cases are then possible. (1) ${t_n}^{N\bullet} \cap P' = \emptyset$ and ${}^{\bullet N}t_n \cap supp(m') = \emptyset$, in which case $\tilde{m}' = m'$ and so m' is reachable in N'. (2) ${t_n}^{N\bullet} \cap P' \neq \emptyset$ and ${}^{\bullet N}t_n \cap supp(m') = \emptyset$, in which case, as N' is $T - complete$, t_n must be in T'. Moreover, as t_n is fireable in N from $\tilde{m}$, it must be fireable as well in N' from $\tilde{m}'$. The effects of t_n on the places of P' are the same in N and N', so $\tilde{m}' \xrightarrow{t_n} m'$ in N'. Hence, m' is reachable in N'. (3) ${t_n}^{N\bullet} \cap P' = \emptyset$ and ${}^{\bullet N}t_n \cap supp(m') \neq \emptyset$, in which case, as N' is m'-complete, t_n must be in T'. Then, using the same arguments as in case (2), m' is reachable in N'. (4) ${t_n}^{N\bullet} \cap P' \neq \emptyset$ and ${}^{\bullet N}t_n \cap supp(m') \neq \emptyset$, in which case, as N' is T-complete and m'-complete, t_n must be in T'. Again, using the same arguments as in case (2), m' is reachable in N'. In each case, m' is reachable, and so the induction hypothesis is also verified for n, which concludes the induction. □

Proposition 2. *If Algorithm 1 returns true, then m is reachable in N.*

Proof. The only way for Algorithm 1 to return true is at line 22. This implies that it goes out of the while loop. It means that both *Complete* and *Consistent* are true (line 5). So, each element of the list $LNets$ is complete according to Definition 10 (line 6). Hence, for any k, $LNets[k]$ is P-complete and the submarking m_k of m in $LNets[k]$ is reachable in $LNets[k]$. By Lemma 2, the partial marking of N whose support is exactly the same as the support of m_k is reachable in N, using the same sequence of transitions ω_k as in $LNets[k]$. Moreover, the list $LNets$ is consistent according to Definition 11 (line 15). Hence, for any k, ℓ the

sets of places of $LNets[k]$ and $LNets[\ell]$ are disjoint (part 1. of Definition 11), as $LNets[k]$ is P-complete, this implies that no transition from $LNets[k]$ can reduce the marking of a place from $LNets[\ell]$. Moreover, no transition from $LNets[k]$ can increase the marking of a place from the support of m_ℓ (part 2. of Definition 11). As a consequence, the concatenation of all the ω_k allows to reach the objective marking m in N. □

Lemma 2. *Let N' be a P-complete subnet of a Petri net N. Let m' be a partial marking of N' and let m be a partial marking of N so that $supp(m) = supp(m')$ and for all $p \in supp(m)$, $m(p) = m'(p)$. If there exists a sequence of transitions ω such that $m'_0 \xrightarrow{\omega} m'$ in N', then $m_0 \xrightarrow{\omega} m$ in N.*

Proof. We proceed by induction.

Induction hypothesis. For all n, if a partial marking m' is reachable in N' by a sequence ω of n transitions, then there exists a partial marking m so that $supp(m) = supp(m')$, for all $p \in supp(m)$, $m(p) = m'(p)$ and m is reachable in N by the sequence ω.

Initialisation. When $n = 0$, any partial marking m' reachable by an empty sequence of transitions must realize m'_0, hence the partial marking m such that $\forall p \in P', m(p) = m'(p)$ and $\forall p \in P \backslash P', m(p) = \star$ must also be realized by m_0 and is thus reachable. This marking m is obviously such that $supp(m) = supp(m')$, which concludes the initialisation.

Induction. Let consider some $n > 0$ and assume that the induction hypothesis is verified for $n-1$. First remark that ω can always be split in two parts: ω' of length $n-1$ and t_n (a single transition) such that $m'_0 \xrightarrow{\omega'} \tilde{m}'_c \xrightarrow{t_n} m'_c$ in N' for some marking $\tilde{m}'_c$ (not a partial one) and some marking m'_c that realizes m'. From the induction hypothesis the only partial marking $\tilde{m}$ so that $supp(\tilde{m}) = supp(\tilde{m'}_c)$ is reachable in N by the sequence ω', so there exists $\tilde{m}_c$ a marking that realizes $\tilde{m}$ and is reached by ω'. Moreover, as N' is P-complete, if t_n is fireable from $\tilde{m}'_c$ in N', then t_n is fireable from $\tilde{m}_c$ in N (all the preconditions of t_n appear in N'). Firing t_n in N leads to a marking m_c so that $\forall p \in P', m_c(p) = m'_c(p)$, by definition of a subnet. Hence, taking for m the partial marking such that $\forall p \in supp(m'), m(p) = m_c(p) = m'_c(p) = m'(p)$ and $\forall p \in P \backslash supp(m'), m(p) = \star$ concludes the induction. □

Proposition 3. *Algorithm 1 always terminates and returns true or false.*

In order to prove Proposition 3 we define a relation over the lists $LNets$ involved in an execution of Algorithm 1. We show that this relation is an order relation and use this fact to conclude about the termination of the algorithm.

Definition 12. *Let $LNets_1$ and $LNets_2$ be two lists involved in an execution of Algorithm 1. We write $LNets_1 <_\ell LNets_2$ if and only if:*

- *$length(LNets_1) < length(LNets_2)$ or*

– $length(LNets_1) = length(LNets_2)$ *and* $\exists 1 \leq k \leq length(LNets_2)$ *such that* $\forall 1 < i < k, LNets_1[i] = LNets_2[i]$ *and* $LNets_1[k] <_n LNets_2[k]$,

where, for two subnets N_1 *and* N_2 *of a net* N*, we have* $N_1 <_n N_2$ *if and only if:*

– $P_1 \supset P_2$ *or*
– $P_1 = P_2$ *and* $T_1 \supset T_2$.

If $LNets_1 <_\ell LNets_2$ *or* $LNets_1 = LNets_2$ *we write* $LNets_1 \leq_\ell LNets_2$.

Lemma 3. *The relation* $\leq_\ell$ *of Definition 12 is an order relation.*

Proof. We prove that $\leq_\ell$ is reflexive, antisymmetric, and transitive.

Reflexive. This is a direct consequence of the fact that equality is reflexive.

Antisymmetric. Assume that $LNets_1 \leq_\ell LNets_2$ and $LNets_2 \leq_\ell LNets_1$. Suppose that $LNets_1 <_\ell LNets_2$. If $length(LNets_1) < length(LNets_2)$, then $length(LNets_1) \neq length(LNets_2)$ and $length(LNets_2) < length(LNets_1)$ cannot be true, so neither $LNets_2 <_\ell LNets_1$ nor $LNets_2 = LNets_1$ can be true, so $LNets_2 \leq_\ell LNets_1$ cannot be true. If $length(LNets_1) = length(LNets_2)$ and $LNets_1[k] < LNets_2[k]$ for some k with $LNets_1[i] = LNets_2[i]$ for any $i < k$, then either (1) $P_1 \subset P_2$ or (2) $P_1 = P_2$ and $T_1 \subset T_2$. In case (1), then $P_2 \neq P_1$ and $P_2 \subset P_1$ cannot be true, moreover $length(LNets_2) < length(LNets_1)$ cannot be true. So $LNets_2 <_\ell LNets_1$ nor $LNets_2 = LNets_1$ can be true, so $LNets_2 \leq_\ell LNets_1$ cannot be true. In case (2), then $P_2 = P_1$ but $T_2 \neq T_1$ and $T_2 \subset T_1$ cannot be true, moreover $length(LNets_2) < length(LNets_1)$ cannot be true. So $LNets_2 <_\ell LNets_1$ nor $LNets_2 = LNets_1$ can be true, so $LNets_2 \leq_\ell LNets_1$ cannot be true. In all cases if $LNets_1 <_\ell LNets_2$ then $LNets_2 \leq_\ell LNets_1$ cannot be true. Thus, as $LNets_1 \leq_\ell LNets_2$, one necessarily gets $LNets_1 = LNets_2$. This proves that $\leq_\ell$ is antisymmetric.

Transitive. Assume that $LNets_1 \leq_\ell LNets_2$ and $LNets_2 \leq_\ell LNets_3$. We show that $LNets_1 \leq_\ell LNets_3$. If $LNets_1 = LNets_2$ or $LNets_2 = LNets_3$, this is clearly true. Thus, assume $LNets_1 <_\ell LNets_2$ and $LNets_2 <_\ell LNets_3$. If $length(LNets_1) < length(LNets_2)$, then, as $length(LNets_2) \leq length(LNets_3)$, one gets $length(LNets_1) < length(LNets_3)$, thus $LNets_1 \leq_\ell LNets_3$. In the case where $length(LNets_1) = length(LNets_2)$, two cases are possible : (1) $length(LNets_2) < length(LNets_3)$, then $length(LNets_1) < length(LNets_3)$ is clearly true, and so $LNets_1 \leq_\ell LNets_3$, (2) $length(LNets_2) = length(LNets_3)$. In this second case, there exists k such that $LNets_1[k] <_n LNets_2[k]$ with $\forall i < k, LNets_1[i] = LNets_2[i]$ and k' such that $LNets_2[k'] <_n LNets_3[k']$ with $\forall i < k', LNets_2[i] = LNets_3[i]$. We need to distinguish three cases: (a) k < k', (b) k = k', and (c) k > k'. In case (a), one gets $LNets_1[k] <_n LNets_3[k]$ and $\forall i < k, LNets_1[i] = LNets_3[i]$, thus $LNets_1 \leq_\ell LNets_3$. In case (b), one gets $\forall i < k, LNets_1[i] = LNets_3[i]$ and $LNets_1[k] <_n LNets_2[k] <_n LNets_3[k]$. The transitivity of $<_n$ (immediately obtained by transitivity of $\subset$) is sufficient to

conclude that $LNets_1[k] <_n LNets_3[k]$, and thus $LNets_1 \leq_\ell LNets_3$. In case (c), one gets $LNets_1[k'] <_n LNets_3[k']$ and $\forall i < k', LNets_1[i] = LNets_3[i]$, thus $LNets_1 \leq_\ell LNets_3$. □

Proof (of Proposition 3). The only return statements in Algorithm 1 are at line 22 and line 11. At line 22 the algorithm returns true and at line 11 it returns false. This implies that the only possible return values are true and false. It remains to prove that the algorithm terminates.

We prove that the successive values of $LNets$ in Algorithm 1 are strictly decreasing with respect to the order relation $\leq_\ell$. As there exists a minimal element (the empty list) with respect to this relation, this suffices to prove the termination.

Assume that an iteration of the main loop (while loop at line 5) starts. This results in, at least, one call to *Concretise* or one call to *Merge*. Thus, if we show that $LNets$ strictly decreases with respect to $\leq_\ell$ after a call to either of these functions, the termination is given by the above argument.

A call to *Merge* strictly decreases the length of $LNets$. So, by the first point of Definition 12, $LNets$ strictly decreases with respect to $\leq_\ell$.

A call to *Concretise* does not modify the length of $LNets$ and modifies exactly one element e of $LNets$ (or returns false, in which case Algorithm 1 terminates). The modified element is chosen so that its set of transitions is strictly increased (line 5 of Algorithm 2) or its set of places is strictly increased (line 12 of Algorithm 2). In either case, the modified element e' is such that $e' <_n e$. This ensures that $LNets$ strictly decreases with respect to $\leq_\ell$ and concludes the proof. □

4 Lazy reachability analysis with inhibitor arcs

We now transpose the previous results from Petri nets to Petri nets with inhibitor arcs. In such nets, places – when marked – can prevent transitions from being fired. The results of the previous section were correct in all Petri nets, bounded or not. In this section, this will no longer be the case as reachability is known for not being decidable in unbounded Petri nets with inhibitor arcs [20,1].

4.1 From Petri nets to Petri nets with inhibitor arcs

We start by formally defining Petri nets with inhibitor arcs.

Definition 13 (Petri net with inhibitor arcs). *A* Petri net with inhibitor arcs *is a tuple* $N_I = (P, T, F, I, m_0)$ *where* (P, T, F, m_0) *is a Petri net and* $I : P \times T \to \mathbb{N} \cup \{\infty\}$ *is an* inhibition function.

Markings, presets and postsets are defined similarly in Petri nets with and without inhibitor arcs. In a Petri net with inhibitor arcs N_I, for any $t \in T$, we define ${}^\circ t = \{p \in P \ : \ I(p,t) \neq \infty\}$ the *inhibition set* of t. We extend this notion to sets T of transitions by union of inhibition sets. A transition

$t \in T$ is *fireable* from a marking m if and only if $\forall p \in {}^\bullet t, m(p) \geq F(p,t)$ and $\forall p \in t^\circ, m(p) < I(p,t)$. The result of firing t is similar as for Petri nets without inhibitor arcs, only the fireability condition changes. Reachability is thus also similarly defined in these two kinds nets.

Definition 14 (Subnet with inhibitor arcs). *A* subnet N'_I *of a Petri net with inhibitor arcs* $N_I = (P, T, F, I, m_0)$ *is a tuple* (P', T', F', I', m'_0) *such that* (P', T', F', m'_0) *is a subnet of* (P, T, F, m_0) *and* $I' = I_{|P',T'}$.

Given a subnet N'_I of a Petri net with inhibitor arcs N_I, and for any $t \in T'$, we define ${}^{\circ_{N_I}}t = \{p \in P \ : \ I(p,t) \neq \infty)\}$ (that is, intuitively, the inhibition set taken in N_I rather than in N'_I).

The notions of P-completeness, T-completeness, partial marking, reachability of partial markings, and m-completeness remain the same in presence of inhibitor arcs. However, we introduce two other notions of completeness with respect to a net N_I for a subnet N'_I. The notion of PI-completeness expresses that N'_I contains all the places from N_I that may inhibit transitions in N'_I.

Definition 15 (PI-completeness). *A subnet* N'_I *of a net* N_I *is said to be* PI-complete *when* $\forall t \in T', {}^{\circ_{N_I}}t \subseteq P'$.

The notion of TI-completeness expresses that N'_I contains all the transitions from N_I that may remove tokens from places that inhibit transitions in N'_I.

Definition 16 (TI-completeness). *A subnet* N'_I *of a net* N_I *is said to be* TI-complete *when* $\forall p \in {}^\circ T', p^{N_I\bullet} \subseteq T'$.

4.2 An algorithm for lazy reachability analysis with inhibitor arcs

The basic principles of our lazy reachability analysis algorithm for Petri nets with inhibitor arcs are the same as for the case where there are no inhibitor arcs. In fact, the main algorithm that we use is still Algorithm 1 – we simply rename N as N_I to make it clear that it has inhibitor arcs – and the merging does not change. However, we use a different concretisation function (Algorithm 3) as well as the following definitions for completeness and consistency.

Definition 17. *Let* $N_I = (P, T, F, I, m_0)$ *be a Petri net with inhibitor arcs and* m *a marking. A subnet* N'_I *of* N_I *is* complete *with respect to* N_I *and* m *if (*N'_I *is P-complete, PI-complete and) the submarking* m' *is reachable in* N'_I.

Since we still need to check reachability, we must now assume that N_I is bounded, though we do not need to know the bound. Even when N_I is bounded, its subnets may be unbounded. In the concretisation function, we will nonetheless build our subnets so that they are bounded by construction.

Definition 18. *The list of subnets* $LNets = [(P_1, T_1), ..., (P_n, T_n)]$ *is* consistent *if*

1. $\forall k \neq \ell, (P_k \cap P_\ell) = \emptyset$, *and*

Algorithm 3 Auxiliary function $Concretise(LNets, m)$ for Algorithm 1

choose k such that $LNets[k]$ is not complete
$(P_k, T_k) \leftarrow LNets[k]$
$m' \leftarrow m_{|P_k}$
choose T'_k such that $T_k \subset T'_k \subseteq {}^{\bullet_{N_I}} P_k \cup supp(m')^{N_I \bullet} \cup ({}^{\circ} T_k)^{N_I \bullet}$
if not possible then
    return false
else
    $P'_k \leftarrow {}^{\bullet_{N_I}} T'_k \cup {}^{\circ_{N_I}} T'_k$
    $LNets[k] \leftarrow (P'_k, T'_k)$
end if
return true

2. $\forall k \neq \ell, {T_k}^{N\bullet} \cap supp(m_{|P_\ell}) = \emptyset$.
3. $\forall k \neq \ell, {T_k}^{N\bullet} \cap {}^{\circ} T_\ell = \emptyset$.

The main difference with the concretisation function that was used for Petri nets with no inhibitor arcs is that one does not distinguish between the case where transitions should be added and the case where places should be added. Indeed, if one allows for adding transitions without their preconditions, this may result in unbounded subnets. In presence of inhibitor arcs this prevents for checking reachability. One can remark, however, that if all the preconditions of all transitions of a subnet are also part of this subnet, then this subnet is necessarily bounded (if the original net was bounded). This is expressed by Proposition 4 below. Thus, in Algorithm 3, one adds transitions as before to the considered subnet but then one always adds all the preconditions of the newly added transitions.

Remark 1. This explains the parenthesis in Definition 17: all the subnets considered are always P-complete and PI-complete.

Finally, notice that transitions that may remove tokens from inhibition places are also considered when adding transitions, as they can enable new transitions firings.

Proposition 4. *Let $N_I = (P, T, F, I, m_0)$ be a bounded Petri net with inhibitor arcs. Let $N'_I = (P', T', F', I', m_{0|P'})$ be a subnet of N_I. If $P' \supseteq {}^{\bullet_{N_I}} T'_k \cup {}^{\circ_{N_I}} T'_k$, then N'_I is bounded.*

Proof. Since N_I is bounded, let k be the corresponding bound. Assume N'_I is not k-bounded. Then there exists a transition firing sequence $\omega = t_1, \ldots, t_n$, some marking m', and some place $p \in P'$ such that in N'_I, we have $m_0 \xrightarrow{m'}$ and $m'(p) > k$.

We prove by induction on the length n of ω that it is also fireable in N_I and that if m (resp. m') is the marking obtained in N_I (resp. in N'_I) after firing ω, then $m_{|P'} = m'$.

First suppose $n = 0$, then the property holds trivially. Now assume it holds for some sequence $t_1, \ldots, t_n$, with $n \geq 0$, and consider an additional transition $t_{n+1} \in T'$. Let m_n (resp. m'_n) be the marking obtained in N_I (resp. N'_I) after firing $t_1, \ldots, t_n$. By the induction hypothesis $m_{n|P} = m'_n$. By construction the preset and inhibitor preset of t_{n+1} in N_I are included in P' and therefore since t_{n+1} is fireable in N'_I from m'_n, it is fireable in N_I from m_n and the effect of those firings on the places in P' is the same.

Since N_I is k-bounded, ω cannot be fireable in N_I so we have a contradiction and N'_I is therefore bounded. □

4.3 Proof of the algorithm

We now prove the correctness of Algorithm 1 when used with the concretisation function of Algorithm 3 on a bounded Petri net with inhibitor arcs.

First, remark that completeness of the algorithm is achieved essentially for the same reasons as in the previous case.

Proposition 5. *Algorithm 1 always terminates and returns true or false when used with the concretisation function of Algorithm 3 on a bounded Petri net with inhibitor arcs.*

Proof. The proof of Proposition 3 also works here as the order relation of Definition 12 does not depends on the arcs of the nets but only on their places and transitions. The only difference in the proof is to remark that a call to *Concretise* always increases the set of transitions. □

It remains to prove to soundness of the algorithm, which is achieved by proving Propositions 6 and 7.

Proposition 6. *If Algorithm 1 used with the concretisation function of Algorithm 3 on a bounded Petri net N_I with inhibitor arcs returns false, then m is not reachable in N_I.*

Proof. As before, the only way for Algorithm 1 to return false is at line 11. It implies that the previous call to the Concretise function (Algorithm 3) returned false. This can only occur at line 6 of Algorithm 3.

In this case, it must not be possible to choose T'_k such that $T_k \subset T'_k \subseteq {}^{\bullet^{N_I}}P_k \cup supp(m')^{N_I\bullet} \cup ({}^{\circ}T_k)^{N_I\bullet}$ (line 4). In other words, the subnet $LNets[k]$ considered is m'-complete (Definition 9), T-complete (Definition 6), and TI-complete (Definition 16). Moreover, the current marking m' must not be reachable in the subnet $LNets[k]$ due to Remark 1 and is a submarking of m. Hence, by applying Lemma 4 below, m is not reachable in N_I. □

Lemma 4. *Let N'_I be a subnet of a bounded Petri net with inhibitor arcs N_I. Assume that N'_I is T-complete and TI-complete. Let m' be a marking of N'_I, that is not reachable. If N'_I is m'-complete, then no marking m of N_I such that m' is the submarking of m in N'_I is reachable in N_I.*

Proof. Let m be a reachable marking of N_I, with $m_0 \xrightarrow{\omega} m$. Denote by n the number of transitions in ω. We can prove by induction on n that m' is reachable in N'_I. By the contrapositive, this proves the Lemma.

The induction is in fact similar to the one used for proving Lemma 1. The only difference is in the induction step where – in cases (2), (3), and (4) – one could imagine that t_n would be fireable in N_I but not in N'_I, due to inhibitor arcs. However, the fact N'_I is TI-complete prevents this: any place p that could inhibit t_n must have its full postset in N'_I and so, if, at some point in ω, p is marked, and later it is unmarked by some transition t, then t must be a transition from N'_I. □

Proposition 7. *If Algorithm 1 used with the concretisation function of Algorithm 3 on a bounded Petri net N_I with inhibitor arcs returns true, then m is reachable in N_I.*

Proof. The only way for Algorithm 1 to return true is at line 22. This implies that it goes out of the while loop. It means that both *Complete* and *Consistent* are true (line 5). So, each element of the list $LNets$ is complete according to Definition 17 (line 6). Hence, for any k, $LNets[k]$ is P-complete, PI-complete, and the submarking m_k of m in $LNets[k]$ is reachable in $LNets[k]$. By Lemma 5, the partial marking of N_I whose support is exactly the same as the support of m_k is reachable in N_I, using the same sequence of transitions ω_k as in $LNets[k]$. Moreover, the list $LNets$ is consistent according to Definition 18 (line 15). Hence, for any k, ℓ the sets of places of $LNets[k]$ and $LNets[\ell]$ are disjoint (part 1. of Definition 18), as $LNets[k]$ is P-complete, this implies that no transition from $LNets[k]$ can reduce the marking of a place from $LNets[\ell]$. Moreover, no transition from $LNets[k]$ can increase the marking of a place from the support of m_ℓ (part 2. of Definition 18) or the marking of a place that inhibits a transition of $LNets[\ell]$ (part 3. of Definition 18). As a consequence, the concatenation of all the ω_k allows to reach the objective marking m in N. □

Lemma 5. *Let N'_I be a P-complete and PI-complete subnet of a bounded Petri net with inhibitor arcs N_I. Let m' be a partial marking of N'_I and let m be a partial marking of N_I so that $supp(m) = supp(m')$ and for all $p \in supp(m)$, $m(p) = m'(p)$. If there exists a sequence of transitions ω such that $m'_0 \xrightarrow{\omega} m'$ in N'_I, then $m_0 \xrightarrow{\omega} m$ in N_I.*

Proof. The only difference with Lemma 2 is that some places in N_I may inhibit a transition of ω. This is prevented by the fact that N'_I is PI-complete: all these place must also exist in N'_I as well. A similar induction proof can thus be performed. □

5 Experimental evaluation

We have implemented the algorithms in the tool Romeo [16][5].

[5] 64bits Linux binaries for Romeo and converters from pnml (MCC) to cts (Romeo), and full results are at `http://lara.rts-software.org/`

Note that since Romeo deals with a very expressive formalism, encompassing inhibitor arcs, we have only implemented the concretisation function described in Section 4, even if it is less efficient than the one in Section 3 for the nets used in the experiments, which do not contain inhibitor arcs.

We implement line 4 in Algorithm 3 by always choosing the biggest possible T'_k. Actually to account for the added expressiveness of Romeo, where transitions can modify the marking in an arbitrarily complex way, we add even a bit more: every transition that may modify the marking in P_k, i. e., ${}^{\bullet_{N_I}}P_k \cup P_k{}^{N_I\bullet}$.

For reachability, we perform a simple explicit state exploration with no particular optimization.

Romeo has then been run in a setting as close as we could of the ReachabilityCardinality category of the 2020 edition of the model checking contest [10], and we compare it with the results of the other tools, which we do not recall here for the sake of space but which are fully available in [10]. For each model of the contest, we gave Romeo one hour to solve the same 16 formulas that the contestant had to solve during the contest. The machine we used is not as powerful as the machines used for running the contest (it has four Intel Xeon E5-2620 processors and 128GB of memory), but we did not run Romeo on a virtual machine (during the MCC it would have been the case).

On a factual point of view, Romeo, using only the algorithm presented here, has thus been faced to 1016 different models, among which it fully solved 120 (that is, it solved the 16 formulas corresponding to the model within one hour) and partially solved 288 (that is, solved at least 1 of the formulas within one hour but not the 16 of them). In total, Romeo solved 2654 formulas among 16256. We have also run our algorithm on the 2018 version of the contest, thus on a subset of the models but with different formulas, with similar overall results.

While this overall result is not outstanding, the details are a lot more interesting. First, each result that Romeo returned was the same as the result obtained by the majority of the tools during the actual contest, and hence we can assume with some confidence that all those results are correct.

Second, there is also a lot of room left to improve the algorithm itself, by choosing more cleverly which places to add, which transitions, how much of each partial net to compute, etc. And actually, since we require nothing in terms of exploration order, the technique we propose here should easily be combinable with, e.g., stubborn sets [14] for much better results.

Finally, and most significantly, Romeo managed to completely solve (all 16 formulas) one model (`GPPP-PT-C0010N1000000000`) that no other tool could handle (they solved at most the first formula). We have had the same results for the 2018 formulas and at that time the other tools had solved none of them. This is very relevant because most of the best performing tools actually have a portfolio approach, in which several techniques are tried in parallel. It thus seems that this model is particularly difficult for current state-of-the-art tools, none of the currently implemented approaches is efficient to handle it, and hardly any progress has been made on it for the last four years (the model was introduced in the 2016 edition).

In contrast, the algorithm we have proposed here performs very well on that model and is thus likely to be a worthy addition to the portfolio approach.

6 Conclusion

In this paper, we have presented an algorithm for reachability analysis in possibly unbounded Petri nets and in bounded Petri Nets with inhibitor arcs. This algorithm heuristically performs reachability queries on subnets of the original net, in a lazy manner: it works on subnets of increasing size, trying to answer as soon as possible. We have proven that, in each case (unbounded Petri nets, bounded Petri nets with inhibitor arcs), the algorithm terminates, always answers, and always gives correct answers.

We have implemented the approach in the tool Romeo and performed a large scale experimental evaluation based on the models from the 2018 edition of the *model-checking contest*, showing that on all the models we indeed answered correctly. Moreover, it revealed that our implementation can solve problems that state-of-the-art model-checking tools cannot handle. While for many other problems, those tools outperform our implementation, we believe our algorithm is still a good candidate for inclusion in a portfolio approach.

Future work consists in incorporating the rest of the features of Roméo in the lazy framework: timing parameters, cost optimization, properties beyond reachability, control, etc.

References

1. S. Akshay, Supratik Chakraborty, Ankush Das, Vishal Jagannath, and Sai Sandeep. On Petri nets with hierarchical special arcs. In *CONCUR*, pages 40:1–40:17, 2017.
2. Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on UPPAAL. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 200–236, 2004.
3. Blai Bonet, Patrik Haslum, Sarah Hickmott, and Sylvie Thiébaux. Directed unfolding of Petri nets. *ToPNOC*, 1(1):172–198, 2008.
4. Thomas Chatain and Loïc Paulevé. Goal-driven unfolding of Petri nets. In *CONCUR*, pages 18:1–18:16, 2017.
5. Jean-Michel Couvreur and Yann Thierry-Mieg. Hierarchical decision diagrams to exploit model structure. In Farn Wang, editor, *FORTE*, pages 443–457, 2005.
6. Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. In *TACAS*, pages 87–106, 1996.
7. G. J. Holzmann and D. Peled. An improvement in formal verification. In *FORTE*, pages 197–211, 1994.
8. Loïg Jezequel and Didier Lime. Lazy reachability analysis in distributed systems. In *CONCUR*, pages 17:1–17:14, 2016.
9. Loïg Jezequel and Didier Lime. Let's be lazy, we have time - or, lazy reachability analysis for timed automata. In *FORMATS*, pages 247–263, 2017.
10. F. Kordon, H. Garavel, L. M. Hillah, F. Hulin-Hubard, E. Amparore, B. Berthomieu, S. Biswal, D. Donatelli, F. Galla, G. Ciardo, S. Dal Zilio, P.

G. Jensen, C. He, D. Le Botlan, S. Li, A. Miner, J. Srba, and . Thierry-Mieg. Complete Results for the 2020 Edition of the Model Checking Contest. http://mcc.lip6.fr/2020/results.php, June 2020.
11. Fabrice Kordon, Hubert Garavel, Lom-Messan Hillah, Emmanuel Paviot-Adet, Loïg Jezequel, César Rodríguez, and Francis Hulin-Hubard. MCC'2015 - the fifth model checking contest. *ToPNOC*, 11:262–273, 2016.
12. S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *STOC*, pages 267–281. ACM, 1982.
13. Jean-Luc Lambert. A structure to decide reachability in Petri nets. *TCS*, 99(1):79–104, 1992.
14. A. Lehmann, N. Lohmann, and K. Wolf. Stubborn sets for simple linear time properties. In *ICATPN*, pages 228–247, 2012.
15. Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *LICS*, pages 56–67. IEEE Computer Society, 2015.
16. Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In *TACAS*, pages 54–57, 2009.
17. Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.
18. Kenneth McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *CAV*, pages 164–177, 1993.
19. A. Miner and J. Babar. Meddly: Multi-terminal and edge-valued decision diagram library. In *QEST*, pages 195–196, 2010.
20. Klaus Reinhardt. Reachability in Petri nets with inhibitor arcs. *ENTCS*, 223:239–264, 2008.
21. M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, SE-10(4):352–357, 1984.
22. Karsten Wolf. Running LoLA 2.0 in a model checking competition. *ToPNOC*, 11:274–285, 2016.