Property Directed Reachability for Generalized Petri Nets

Nicolas Amat[1], Silvano Dal Zilio[1], and Thomas Hujsa[1]

LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
namat@laas.fr

Abstract. We propose a semi-decision procedure for checking generalized reachability properties, on generalized Petri nets, that is based on the Property Directed Reachability (PDR) method. We actually define three different versions, that vary depending on the method used for abstracting possible witnesses, and that are able to handle problems of increasing difficulty. We have implemented our methods in a model-checker called SMPT and give empirical evidences that our approach can handle problems that are difficult or impossible to check with current state of the art tools.

1 Introduction

We propose a new semi-decision procedure for checking reachability properties on generalized Petri nets, meaning that we impose no constraints on the weights of the arcs and do not require a finite state space. We also consider a generalized notion of reachability, in the sense that we can not only check the reachability of a given state, but also if it is possible to reach a marking that satisfies a combination of linear constraints between places, such as $(p_0+p_1 = p_2+2) \wedge (p_1 \leqslant p_2)$ for example. Another interesting feature of our approach is that we are able to return a "certificate of invariance", in the form of an *inductive linear invariant* [26], when we find that a constraint is true on all the reachable markings. To the best of our knowledge, there is no other tool able to compute such certificates in the general case.

Our approach is based on an extension of the Property Directed Reachability (PDR) method, originally developed for hardware model-checking [8,9], to the case of Petri nets. We actually define three variants of our algorithm—two of them completely new when compared to our previous work [1]—that vary based on the method used for generalizing possible witnesses and can handle problems of increasing difficulty.

Reachability for Petri nets is an important and difficult problem with many practical applications: obviously for the formal verification of concurrent systems, but also for the study of diverse types of protocols (such as biological or business processes); the verification of software systems; the analysis of infinite state systems; etc. It is also a timely subject, as shown by recent publications on this subject [7,15], but also with the recent progress made on settling its theoretical

complexity [12,13], which asserts that reachability is Ackermann-complete, and therefore inherently more complex than, say, the coverability problem.

A practical consequence of this "inherent complexity", and a general consensus, is that we should not expect to find a one-size-fits-all algorithm that could be usable in practice. A better strategy is to try to improve the performances on some cases—for example by developing new tools, or optimizations, that may perform better on some examples—or try to improve "expressiveness"—by finding algorithms that can manage new cases, that no other tool can handle.

This wisdom is illustrated by the current state of the art at the Model Checking Contest (MCC) [3], a competition of model-checkers for Petri nets that includes an examination for the reachability problem. Albeit strongly oriented towards the analysis of bounded nets. As a matter of fact, the top three tools in recent competitions—ITS-Tools [30], LoLA [34], and Tapaal [14]—all rely on a portfolio approach. Methods that have been proposed in this context include the use of symbolic techniques, such as k-induction [31]; abstraction refinement [10]; the use of standard optimizations with Petri nets, like stubborn sets or structural reductions; the use of the "state equation"; reduction to integer linear programming problems; etc.

The results obtained during the MCC highlight the very good performances achieved when putting all these techniques together, on bounded nets, with a collection of randomly generated properties. Another interesting feedback from the MCC is that simulation techniques are very good at finding a *counter-example* when a property is not an invariant [7,31].

In our work, we seek improvements in terms of both *performance* and *expressiveness*. We also target what we consider to be a difficult, and less studied area of research: procedures that can be applied when a property is an invariant and when the net is unbounded, or its state space cannot be fully explored. We also focus on the verification of "genuine" reachability constraints, which are not instances of a coverability problem. These properties are seldom studied in the context of unbounded nets. Interestingly enough, our work provides a simple explanation of why coverability problems are also "simpler" in the case of PDR; what we will associate with the notion of *monotonic formulas*.

Concerning performances, we propose a method based on a well-tried symbolic technique, PDR, that has proved successful with unbounded model-checking and when used together with SMT solvers [11,22]. Concerning expressiveness, we define a small benchmark of "difficult nets": a set of synthetic examples, representative of patterns that can make the reachability problem harder.

Outline and Contributions. We define background material on Petri nets in Sect. 2, where we use Linear Integer Arithmetic (LIA) formulas to reason about nets. Section 3 describes our decision method, based on PDR and SMT solvers, for checking the satisfiability of linear invariants over the reachable states of a Petri net. Our method builds sequences of incremental invariants using both a property that we want to disprove, and a stepwise approximation of the reachability relation. It also relies on a generalization step where we can

abstract possible "bad states" into clauses that are propagated in order to find a counter-example, or to block inconsistent states.

We describe a first generalization method, based on the upset of markings, that is able to deal with coverability properties. We propose a new, dual variant based on the concept of *hurdles* [21], that is without restrictions on the properties. In this method, the goal is to block bad sequences of transitions instead of bad states. We show how this approach can be further improved by defining a notion of saturated transition sequence, at the cost of adding universal quantification in our SMT problems.

We have implemented our approach in an open-source tool, called SMPT, and compare it with other existing tools. In this context, one of our contributions is the definition of a set of difficult nets, that characterizes classes of difficult reachability problems.

2 Petri Nets and Linear Reachability Constraints

Let $\mathbb{N}$ denote the set of natural numbers and $\mathbb{Z}$ the set of integers. Assuming P is a finite, totally ordered set $\{p_1, \ldots, p_n\}$, we denote by $\mathbb{N}^P$ the set of mappings from $P \to \mathbb{N}$ and we overload the addition, subtraction and comparison operators $(=, \geq, \leq)$ to act as their component-wise equivalent on mappings. A QF-LIA formula F, with support in P, is a Boolean combination of atomic propositions of the form $\alpha \sim \beta$, where $\sim$ is one of $=, \leq$ or $\geq$ and α, β are *linear expressions*, that is, linear combinations of elements in $\mathbb{N} \cup P$. We simply use the term *linear constraint* to describe F.

A *Petri net* N is a tuple $(P, T, \mathbf{pre}, \mathbf{post})$ where $P = \{p_1, \ldots, p_n\}$ is a finite set of places, T is a finite set of transitions (disjoint from P), and $\mathbf{pre} : T \to \mathbb{N}^P$ and $\mathbf{post} : T \to \mathbb{N}^P$ are the pre- and post-condition functions (also called the flow functions of N). A state m of a net, also called a *marking*, is a mapping of $\mathbb{N}^P$. We say that the marking m assigns $m(p_i)$ tokens to place p_i. A marked net (N, m_0) is a pair composed from a net and an initial marking m_0.

A transition $t \in T$ is *enabled* at marking $m \in \mathbb{N}^P$ when $m \geqslant \mathbf{pre}(t)$. When t is enabled at m, we can fire it and reach another marking $m' \in \mathbb{N}^P$ such that $m' = m - \mathbf{pre}(t) + \mathbf{post}(t)$. We denote this transition $m \xrightarrow{t} m'$. The difference between m and m' is a mapping $\Delta(t) = \mathbf{post}(t) - \mathbf{pre}(t)$ in $\mathbb{Z}^P$, also called the *displacement* of t.

By extension, we say that a *firing sequence* $\sigma = t_1 \ldots t_k \in T^*$ can be fired from m, denoted $m \overset{\sigma}{\Rightarrow} m'$, if there exist markings $m_0, \ldots, m_k$ such that $m = m_0$, $m' = m_k$ and $m_i \xrightarrow{t_{i+1}} m_{i+1}$ for all $i < k$. We can also simply write $m \to^\star m'$. In this case, the displacement of σ is the mapping $\Delta(\sigma) = \Delta(t_1) + \cdots + \Delta(t_k)$. We denote by $R(N, m_0)$ the set of markings reachable from m_0 in N. A marking m is k-bounded when each place has at most k tokens. By extension, we say that a marked Petri net (N, m_0) is bounded when there is k such that all reachable markings are k-bounded.

While reachable states are computed by adding a linear combination of "displacements" (vectors in $\mathbb{Z}^P$), the set $R(N, m_0)$ is not necessarily semilinear or,

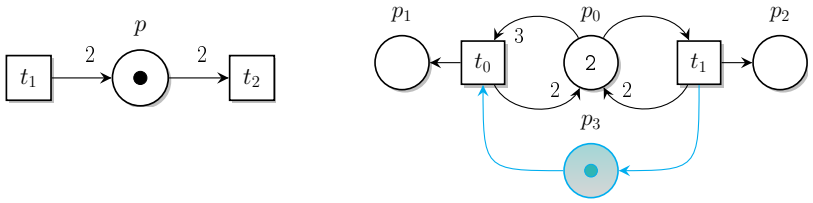

Fig. 1: Two examples of Petri nets: Parity (left) and PGCD (right).

equivalently, definable using Presburger arithmetic [20,26]. This is a consequence of the constraint that transitions must be enabled before firing. But there is still some structure to the set $R(N, m_0)$, like for instance the following monotonicity constraint:

$$\forall m \in \mathbb{N}^P \,.\, m_1 \overset{\sigma}{\Rightarrow} m_2 \;\text{ implies }\; m_1 + m \overset{\sigma}{\Rightarrow} m_2 + m \tag{H1}$$

We have other such results, such as with the notion of *hurdle* [21]. Just as $\mathbf{pre}(t)$ is the smallest marking for which a given transition t is enabled, there is a smallest marking at which a given firing sequence σ is fireable. This marking, denoted by $H(\sigma)$, has a simple inductive definition:

$$H(t) = \mathbf{pre}(t) \quad \text{and} \quad H(\sigma_1 \cdot \sigma_2) = \max\left(H(\sigma_1), H(\sigma_2) - \Delta(\sigma_1)\right) \tag{H2}$$

Given this notion of hurdles, we obtain that $m \overset{\sigma}{\Rightarrow} m'$ if and only if (1) the sequence σ is enabled: $m \geqslant H(\sigma)$, and (2) $m' = m + \Delta(\sigma)$. We use this result in the second variant of our method.

We can go a step further and characterize a necessary and sufficient condition for firing the sequence $\sigma.\sigma^k$, meaning firing the same sequence more than once. Given $\Delta(\sigma)$, a place p with a negative displacement (say $-d$) means that we "loose" d token each time we fire σ. Hence we should budget d tokens in p for each new iteration. On the opposite, nothing is needed for places with a positive displacement, which accrue tokens.

Therefore we have $m \overset{\sigma}{\Rightarrow} \overset{\sigma^k}{\Longrightarrow} m'$ if and only if (1) $m \geqslant H(\sigma) + k \cdot \max(\mathbf{0}, -\Delta(\sigma))$, and (2) $m' = m + (k+1) \cdot \Delta(\sigma)$. Equivalently, if we denote by m^+ the "positive" part of mapping m, such that $m^+(p) = 0$ when $m(p) \leqslant 0$ and $m^+(p) = m(p)$ otherwise, we have:

$$H(\sigma^{k+1}) = \max\left(H(\sigma), H(\sigma) - k \cdot \Delta(\sigma)\right) = H(\sigma) + k \cdot (-\Delta(\sigma))^+ \tag{H3}$$

2.1 Examples

We give two simple examples of unbounded nets in Fig. 1, which are both part of our benchmark. Parity has a single place, hence its state space can be interpreted as a subset of $\mathbb{N}$: with an initial marking of 1, this is exactly the set of odd numbers (and therefore state 0 is not reachable). We are in a special case where the set $R(N, m_0)$ is semilinear. For instance, it can be seen as solution to

the constraint $\exists k.(p = 2k + 1)$, or equivalently $p \equiv 1 \pmod 2$. But it cannot be expressed with a linear constraint involving only the variable p without quantification or modulo arithmetic. This example can be handled by most of the tools used in our experiments, e.g. with the help of k-induction.

In PGCD, transitions t_0/t_1 can decrement/increment the marking of p_0 by 1. Nonetheless, with this initial state, it is the case that the number of occurrences of t_0 is always less than the one of t_1 in any feasible sequence σ. Hence the two predicates $p_0 \geq 2$ and $p_2 \geq p_1$ are valid invariants. (Since some tools do not accept literals of the form $p \geq q$, we added the "redundant" place p_3 so we can restate our second invariant as $p_3 \geq 1$.) These invariants cannot be proved by reasoning only on the displacements of traces (using the state equation) and are already out of reach for LoLA or Tapaal.

2.2 Linear Reachability Formulas

We can revisit the semantics of Petri nets using linear predicates. In the following, we use $\boldsymbol{p}$ for the vector $(p_1, \ldots, p_n)$, and $F(\boldsymbol{p})$ for a formula with variables in P. We also simply use $F(\boldsymbol{\alpha})$ for the substitution $F\{p_1 \leftarrow \alpha_1\} \ldots \{p_n \leftarrow \alpha_n\}$, with $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ a sequence of linear expressions. We say that a mapping m of $\mathbb{N}^P$ is a *model* of F, denoted $m \models F$, if the ground formula $F(m) = F(m(p_1), \ldots, m(p_n))$ is true. Hence we can also interpret F as a predicate over markings. Finally, we define the semantics of F as the set $[\![F]\!] = \{m \in \mathbb{N}^P \mid m \models F\}$.

As usual, we say that a predicate F is *valid*, denoted $\models F$, when all its interpretations are true ($[\![F]\!] = \mathbb{N}^P$); and that F is *unsatisfiable* (or simply `unsat`), denoted $\not\models F$, when $[\![F]\!] = \emptyset$.

We can define many properties on the markings of a net N using this For instance, we can model the set of markings m such that some transition t is enabled using predicate ENBL_t (see Equation (2) below). We can also define a linear predicate to describe the relation between the markings before and after some transition t fires. To this end, we use a vector $\boldsymbol{p'}$ of "primed variables" $(p'_1, \ldots, p'_n)$, where p'_i will stand for the marking of place p_i after a transition is fired. With this convention, formula $\mathrm{FIRE}_t(\boldsymbol{p}, \boldsymbol{p'})$ is such that $\mathrm{FIRE}_t(m, m')$ entails $m \xrightarrow{t} m'$ or $m = m'$ when t is enabled at m. With all these notations, we can define a predicate $\mathrm{T}(\boldsymbol{p}, \boldsymbol{p'})$ that "encodes" the effect of firing at most one transition in the net N.

$$\mathrm{GEQ}_m(\boldsymbol{p}) \stackrel{\text{def}}{=} \bigwedge_{i \in 1..n} (p_i \geqslant m(p_i)) \tag{1}$$

$$\mathrm{ENBL}_t(\boldsymbol{p}) \stackrel{\text{def}}{=} \bigwedge_{i \in 1..n} (p_i \geqslant \mathbf{pre}(t)(p_i)) \;=\; \mathrm{GEQ}_{H(t)}(\boldsymbol{p}) \tag{2}$$

$$\Delta_t(\boldsymbol{p}, \boldsymbol{p'}) \stackrel{\text{def}}{=} \bigwedge_{i \in 1..n} (p'_i = p_i + \mathbf{post}(t)(p_i) - \mathbf{pre}(t)(p_i)) \tag{3}$$

$$\mathrm{EQ}(\boldsymbol{p}, \boldsymbol{p'}) \stackrel{\text{def}}{=} \bigwedge_{i \in 1..n} (p'_i = p_i) \tag{4}$$

$$\mathrm{FIRE}_t(\boldsymbol{p}, \boldsymbol{p'}) \stackrel{\text{def}}{=} \mathrm{EQ}(\boldsymbol{p}, \boldsymbol{p'}) \vee (\mathrm{ENBL}_t(\boldsymbol{p}) \wedge \Delta_t(\boldsymbol{p}, \boldsymbol{p'})) \tag{5}$$

$$\mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \stackrel{\text{def}}{=} \mathrm{EQ}(\boldsymbol{p}, \boldsymbol{p'}) \vee \bigvee_{t \in T} (\mathrm{ENBL}_t(\boldsymbol{p}) \wedge \Delta_t(\boldsymbol{p}, \boldsymbol{p'})) \tag{6}$$

In our work, we focus on the verification of *safety* properties on the reachable markings of a marked net (N, m_0). Examples of properties that we want to check include: checking if some transition t is enabled (commonly known as *quasi-liveness*); checking if there is a deadlock; checking whether some linear invariant between place markings is true; ... All properties that can be expressed using a linear predicate.

Definition 1 (Linear Invariants and Inductive Predicates). *A linear predicate F is an invariant on (N, m_0) if and only if we have $m \models F$ for all $m \in R(N, m_0)$. It is inductive if for all markings m we have $m \models F$ and $m \to m'$ entails $m' \models F$.*

It is possible to characterize inductive predicates using our logical framework. Indeed, F is inductive if and only if the QF-LIA formula (i) $F(\boldsymbol{p}) \wedge T(\boldsymbol{p}, \boldsymbol{p'}) \wedge \neg F(\boldsymbol{p'})$ is `unsat`. Also, an inductive formula is an invariant when (ii) $m_0 \models F$, or equivalently $\models F(m_0)$. As a consequence, a sufficient condition for a predicate F to be invariant is to have both conditions (i) and (ii); conditions that can be checked using a SMT solver. Unfortunately, the predicates that we need to check are often not inductive. In this case, the next best thing is to try to build an inductive invariant, say R, such that $[\![R]\!] \subseteq [\![F]\!]$ (or equivalently $R \wedge \neg F$ `unsat`). This predicate provides a certificate of invariance that can be checked independently.

Lemma 1 (Certificate of Invariance). *A sufficient condition for F to be invariant on (N, m_0) is to exhibit a linear predicate R that is (i) initial: $R(m_0)$ `valid`; (ii) inductive: $R(\boldsymbol{p}) \wedge T(\boldsymbol{p}, \boldsymbol{p'}) \wedge \neg R(\boldsymbol{p}\,')$ `unsat`; and (iii) that entails F, for instance: $R \wedge \neg F$ `unsat`.*

This result is in line with a property proved by Leroux [26], which states that when a final configuration m is not reachable there must exist a Presburger inductive invariant that contains m_0 but does not contain m. This result does not explain how to effectively compute such an invariant. Moreover, in our case, we provide a method that works with general linear predicates, and not only with single configurations. On the other side of the coin, given the known results about the complexity of the problem, we do not expect our procedure to be complete in the general case.

In the next section, we show how to (potentially) find such certificates using an adaptation of the PDR method. An essential component of PDR is to abstract a "scenario" leading to the model of some property F—say a transition $m \stackrel{\sigma}{\Rightarrow} m'$ with $m' \models F$—into a predicate that contains m (and potentially many more similar scenarios). More generally, a *generalization* of the trio (m, σ, F) is a predicate G satisfied by m such that $m_1 \models G$ entails that there is $m_1 \to^\star m_2$ with $m_2 \models F$.

We can use properties (H1)–(H3), defined earlier, to build generalizations.

Lemma 2 (Generalization). *Assume we have a scenario such that* $m \overset{\sigma}{\Rightarrow} m'$ *and* $m' \models F$. *We have three possible generalizations of the trio* (m, σ, F).

(G1) *If property* F *is monotonic, then* $m_1 \models \mathrm{GEQ}_m(\boldsymbol{p})$ *implies there is* $m_2 \geqslant m'$ *such that* $m_1 \overset{\sigma}{\Rightarrow} m_2$ *and* $m_2 \models F$.

(G2) *If* $m_1 \models \mathrm{GEQ}_{H(\sigma)}(\boldsymbol{p}) \wedge F(\boldsymbol{p} + \Delta(\sigma))$ *then* $m_1 \overset{\sigma}{\Rightarrow} m_2$ *and* $m_2 \models F$.

(G3) *Assume* a, b *are mappings of* $\mathbb{N}^P$ *such that* $a = H(\sigma)$ *and* $b = (-\Delta(\sigma))^+$, *with the notations used in (H3). Then*

$$m_1 \models \exists k. \left(\begin{array}{c} \left[\bigwedge_{i \in 1..n} (p_i \geqslant a(i) + k \cdot b(i))\right] \\ \wedge F(\boldsymbol{p} + (k+1) \cdot \Delta(\sigma)) \end{array} \right) \quad \textit{implies} \quad \left\{ \begin{array}{l} \exists k. m_1 \overset{\sigma^{k+1}}{\Longrightarrow} m_2 \\ \textit{and } m_2 \models F \end{array} \right.$$

Proof. Each property is a direct result of properties (H1) to (H3). □

Property (G3) is the first and only instance of linear formula using an extra variable, k, that is not in P. The result is still a linear formula though, since we never need to use the product of two variables. This generalization is used when we want to "saturate the sequence σ". This is the only situation where we may need to deal with quantified LIA formulas. Another solution would be to replace each quantification with the use of modulo arithmetic, but this operation may be costly and could greatly increase the size of our formulas. It would also not cut down the complexity of the SMT problems.

3 Property Directed Reachability

Some symbolic model-checking procedure, such as BMC [6] or k-induction [28], are a good fit when we try to find counter-examples on infinite-state systems. Unfortunately, they may perform poorly when we want to check an invariant. In this case, adaptations of the PDR method [8,9] (also known as IC3, for "Incremental Construction of Inductive Clauses for Indubitable Correctness") have proved successful.

We assume that we start with an initial state m_0 satisfying a linear property, $\mathbb{I}$, and that we want to prove that property $\mathbb{P}$ is an invariant of the marked net (N, m_0). (We use blackboard bold symbols to distinguish between parameters of the problem, and formulas that we build for solving it.) When checking for the reachability from the initial state, we can simple choose $\mathbb{I}$ such that $[\![\mathbb{I}]\!] = \{m_0\}$.

We define $\mathbb{F} = \neg\mathbb{P}$ as the "set of feared events"; such that $\mathbb{P}$ is not an invariant if we can find m in $R(N, m_0)$ such that $m \models \mathbb{F}$. To simplify the presentation, we assume that $\mathbb{F}$ is a conjunction of literals (a cube), meaning that $\mathbb{P}$ is a clause. In practice, we assume that $\mathbb{F}$ is in Disjunctive Normal Form.

PDR is a combination of induction, over-approximation, and SAT or SMT solving. The goal is to build an incremental sequence of predicates $F_0, \ldots, F_k$ that are "inductive relative to stepwise approximations": such that $m \models F_i$ and $m \to m'$ entails $m' \models F_{i+1}$, but not $m' \models \mathbb{F}$. The method stops when it finds a counter-example, or when we find that one of the predicates F_i is inductive.

Function prove($\mathbb{I}$, $\mathbb{F}$: linear predicates)

Result: $\bot$ if $\mathbb{F}$ is reachable ($\mathbb{P} = \neg\mathbb{F}$ is not an invariant), otherwise $\top$

```
1  if sat(𝕀(p) ∧ T(p, p') ∧ 𝔽(p')) then
2  |  return ⊥
3  k ← 1, F_0 ← 𝕀, F_1 ← ℙ
4  while ⊤ do
5  |  if not strengthen(k) then
6  |  |  return ⊥
7  |  propagateClauses(k)
8  |  if CL(F_i) = CL(F_{i+1}) for some 1 ⩽ i ⩽ k then
9  |  |  return ⊤
10 |  k ← k + 1
```

We adapt the PDR approach to Petri nets, using linear predicates and SMT solvers for the QF-LIA and LIA logics in order to learn, generalize, and propagate new clauses. The most innovative part of our approach is the use of specific "generalization algorithms" that take advantage of the Petri nets theory, like the use of hurdles for example.

3.1 Algorithm

Our implementation follows closely the algorithm for IC3 described in [9]. We only give a brief sketch of the OARS construction.

The main function, `prove`, computes an *Over Approximated Reachability Sequence* (OARS) $(F_0, \ldots, F_k)$ of linear predicates, called *frames*, with variables in $\boldsymbol{p}$. An OARS meets the following constraints: (1) it is monotonic: $F_i \wedge \neg F_{i+1}$ `unsat` for $0 \leqslant i < k$; (2) it contains the initial states: $\mathbb{I} \wedge \neg F_0$ `unsat`; (3) it does not contain feared states: $F_i \wedge \mathbb{F}$ `unsat` for $0 \leqslant i \leqslant k$; and (4) it satisfies *consecution*: $F_i(\boldsymbol{p}) \wedge \mathrm{T}(\boldsymbol{p}, \boldsymbol{p'}) \wedge \neg F_{i+1}(\boldsymbol{p'})$ `unsat` for $0 \leqslant i < k$.

By construction, each frame F_i in the OARS is defined as a set of clauses, $\mathrm{CL}(F_i)$, meaning that F_i is built as a formula in CNF: $F_i = \bigwedge_{cl \in \mathrm{CL}(F_i)} cl$. We also enforce that $\mathrm{CL}(F_{i+1}) \subseteq \mathrm{CL}(F_i)$ for $0 \leqslant i < k$, which means that the monotonicity property between frames is trivially ensured.

The body of function `prove` contains a *main iteration* (line 4) that increases the value of k (the number of levels of the OARS). At each step, we enter a second, minor iteration (line 2 in function `strengthen`), where we generate new minimal inductive clauses that will be propagated to all the frames. Hence both the length of the OARS, and the set of clauses in its frames, increase during computation.

The procedure stops when we find an index i such that $F_i = F_{i+1}$. In this case we know that F_i is an inductive invariant satisfying $\mathbb{P}$. We can also stop during the iteration if we find a counter-example (a model m of $\mathbb{F}$). In this case, we can also return a trace leading to m.

Function strengthen(k : current level)

```
1 try:
2 |  while (m --t--> m') |= F_k(p) ∧ T(p,p') ∧ F(p') do
3 |  |  s ← generalizeWitness(m, t, F)
4 |  |  n ← inductivelyGeneralize(s, k - 2, k)
5 |  |  pushGeneralization({(s, n+1)}, k)
6 |  return ⊤
7 catch counter example:
8 |  return ⊥
```

Procedure propagateClauses(k: level)

```
1 for i ← 1 to k do
2 |  foreach cl ∈ CL(F_i) do
3 |  |  if ⊭ F_i(p) ∧ T(p,p') ∧ ¬cl(p') then
4 |  |  |  CL(F_{i+1}) ← CL(F_i) ∪ {cl}
```

When we start the first minor iteration, we have $k = 1$, $F_0 = \mathbb{I}$ and $F_1 = \mathbb{P}$. If we have $F_k(\boldsymbol{p}) \wedge T(\boldsymbol{p}, \boldsymbol{p'}) \wedge \mathbb{F}(\boldsymbol{p})$ `unsat`, it means that $\mathbb{P}$ is inductive, so we can stop and return that $\mathbb{P}$ is an invariant. Otherwise, we proceed with the strengthen phase, where each model of $F_k(\boldsymbol{p}) \wedge T(\boldsymbol{p}, \boldsymbol{p'}) \wedge \mathbb{F}(\boldsymbol{p})$ becomes a potential counter-example, or *witness*, that we need to "block" (line 3–5 of function `strengthen`).

Instead of blocking only one witness, we first generalize it into a predicate that abstracts similar dangerous states (see the call to `generalizeWitness`). This is done by applying one of the three generalization results in Lemma 2. We give more details about this step later. By construction, each generalization is a cube s (a conjunction of literals). Hence, when we block it, we learn new clauses from $\neg s$ that can be propagated to the previous frames.

Before pushing a new clause, we test whether s is reachable from previous frames. We take advantage of this opportunity to find if we have a counter-example and, if not, to learn new clauses in the process. This is the role of functions `pushGeneralization` and `inductivelyGeneralize`.

We find a counter example (in the call to `inductivelyGeneralize`) if the generalization from a witness found at level k, say s, reaches level 0 and $F_0(\boldsymbol{p}) \wedge T(\boldsymbol{p}, \boldsymbol{p'}) \wedge s(\boldsymbol{p'})$ is satisfiable (line 1 in `inductivelyGeneralize`). Indeed, it means that we can build a trace from $\mathbb{I}$ to $\mathbb{F}$ by going through $F_1, \ldots, F_k$.

Procedure generateClause(s : cube, i: level, k: level)

```
1 cl ← ¬unsat_core(¬s(p) ∧ F_i(p) ∧ T(p,p') ∧ s(p'))
2 for j ← 1 to i+1 do
3 |  CL(F_j) ← CL(F_j) ∪ {cl}
```

Function inductivelyGeneralize(s : cube, min: level, k: level)

```
1 if min < 0 and sat(F_0(p) ∧ T(p, p') ∧ s(p')) then
2 |   raise Counterexample
3 for i ← max(1, min + 1) to k do
4 |   if sat(F_i(p) ∧ T(p, p') ∧ ¬s(p) ∧ s(p')) then
5 |   |   generateClause(s, i-1, k)
6 |   |   return i − 1
7 generateClause(s, k, k)
8 return k
```

Function pushGeneralization($states$: set of (state, level), k: level)

```
1  while ⊤ do
2  |   (s, n) ← from states minimizing n
3  |   if n > k then
4  |   |   return
5  |   if (m -t-> m') ⊨ F_n(p) ∧ T(p, p') ∧ s(p') then
6  |   |   p ← generalizeWitness(m, t, s)
7  |   |   l ← inductivelyGeneralize(p, n - 2, k)
8  |   |   states ← states ∪ {(p, l + 1)}
9  |   else
10 |   |   l ← inductivelyGeneralize(s, n, k)
11 |   |   states ← states \ {(s, n)} ∪ {(s, l + 1)}
```

The method relies heavily on checking the satisfiability of linear formulas in QF-LIA, which is achieved with a call to a SMT solver. In each function call, we need to test if predicates of the form $F_i \wedge T \wedge G$ are `unsat` and, if not, enumerate its models. To accelerate the strengthening of frames, we also rely on the unsat core of properties in order to compute a *minimal inductive clause* (MIC).

Our approach is parametrized by a generalization function (`generalizeWitness`) that is crucial if we want to avoid enumerating a large, potentially unbounded, set of witnesses. This can be the case, for example, in line 5 of `pushGeneralization`. In this particular case, we find a state m at level n (because $m \models F_n$), and a transition t that leads to a problematic clause in F_{n+1}. Therefore we have a sequence σ of size $k - n + 1$ such that $m \overset{\sigma}{\Rightarrow} m'$ and $m' \models \mathbb{F}$. We consider three possible methods for generalizing the trio $(m, \sigma, \mathbb{F})$, that corresponds to property (G1)–(G3) in Lemma 2.

3.2 State-based Generalization

A special case of the reachability problem is when the predicate $\mathbb{F}$ is monotonic,, meaning that $m_1 \models \mathbb{F}$ entails $m_1 + m_2 \models \mathbb{F}$ for all markings m_1, m_2. A sufficient (syntactic) condition is for $\mathbb{F}$ to be a positive formula with literals of the form $\sum_{i \in I} p_i \geq a$. This class of predicates coincide with what is called a *coverability property*, for which there exists specialized verification methods (see e.g. [18,19]).

By property (G1), If we have to block a witness m such that $m \overset{\sigma}{\Rightarrow} m'$ and $m' \models \mathbb{F}$, we can as well block all the states greater than m. Hence we can choose the predicate GEQ_m to generalize m. This is a very convenient case for verification and one of the optimizations used in previous works on PDR for Petri nets [1,16,23,24]. First, the generalization is very simple and we can easily compute a MIC when we block predicate GEQ_m in a frame. Also, we can prove the completeness of the procedure when $\mathbb{F}$ is monotonic. An intuition is that it is enough, in this case, to check the property on the minimal coverability set of the net, which is always finite [18]. The procedure is also complete for finite transition systems. These are the only cases where we have been able to prove that our method always terminates.

3.3 Transition-based Generalization

We propose a new generalization based on the notion of hurdles. This approach can be used when $\mathbb{F}$ is not monotonic, for example when we want to check an invariant that contains literals of the form $p = k$ (e.g. the reachability of a fixed marking) or $p \geqslant q$.

Assume we need to block a witness of the from $m \overset{\sigma}{\Rightarrow} m' \models s$. Typically, s is a cube in $\mathbb{F}$, or a state resulting from a call to `pushGeneralization`. By property (G2), we can as well block all the states satisfying $G_\sigma(\boldsymbol{p}) \overset{\text{def}}{=} \text{GEQ}_{H(\sigma)}(\boldsymbol{p}) \wedge s(\boldsymbol{p} + \Delta(\sigma))$. This generalization is interesting when property s does not constraint all the places, or when we have few equality constraints. In this case G_σ may have an infinite number of models. It should be noted that using the duality between "feasible traces" and hurdles is not new. For example, it was used recently [19] to accelerate the computation of coverability trees. Nonetheless, to the best of our knowledge, this is the first time that this generalization method has been used with PDR.

3.4 Saturated Transition-based Generalization

We still assume that we start from a witness $m \overset{\sigma}{\Rightarrow} m' \models s$. Our last method relies on property (G3) and allows us to consider several iterations of σ. If we fix the value of k, then a possible generalization is $G^k_\sigma \overset{\text{def}}{=} \left(\bigwedge_{i \in 1..n}(p_i \geqslant a(i) + k \cdot b(i))\right) \wedge s(\boldsymbol{p} + (k+1) \cdot \Delta(\sigma))$, where a, b are the mappings of $\mathbb{N}^P$ defined in Lemma 2. (Notice that $G^1_\sigma = G_\sigma$.) More generally the predicate $G^{\leqslant k}_\sigma = G^1_\sigma \vee \cdots \vee G^k_\sigma$ is a valid generalization for the witness (m, σ, s), in the sense that if $m_1 \models G^{\leqslant k}_\sigma$ then there is a trace $m_1 \to^\star m_2$ such that $m_2 \models s$. At the cost of using existential quantification (and therefore a "top-level" universal quantification when we negate the predicate to block it in a frame), we can use the more general predicate $G^\star_\sigma \overset{\text{def}}{=} \exists k. G^k_\sigma$, which is still linear and has its support in P.

We know examples of invariants where the PDR method does not terminate except when using saturation. A simple example is the net Parity, used as an example in Sect. 2, with the invariant $\mathbb{P} = (p \geqslant 1)$. In this case, $\mathbb{F} = \neg\mathbb{P} = (p = 0)$. Hence we are looking for witnesses such that $m \to^\star 0$. The simplest example is $2 \xrightarrow{t_2} 0$, which corresponds to the "blocking clause" $p \neq 2$. In this case, we

have $H(t_2) = 2$ and $\Delta(t_2) = -2$. Hence the transition-based generalization is $(p \geq 2) \wedge (p - 2 = 0) \equiv (p = 2)$, which does not block new markings. At this point, we try to block $(p = 0) \vee (p = 2)$. The following minor iteration of our method will consider the witness $4 \xRightarrow{t_2.t_2} 0$, etc. Hence after k minor iterations, we have $F_k \equiv (p \neq 0) \wedge (p \neq 2) \wedge \cdots \wedge (p \neq 2k)$. If we saturate t_2, we find in one step that we should block $\exists k.(p - 2 \cdot (k + 1) = 0)$. This is enough to prove that $(p \geqslant 1)$ is an invariant as soon as the initial marking is an odd number.

This example proves that PDR is not complete, without saturation, in the general case. We conjecture that it is also the case with saturation. Even though example Parity is extremely simple, it is also enough to demonstrate the limit of our method without saturation. Indeed, when we only allow unquantified linear predicates with variables in P, it is not possible to express all the possible semilinear sets in $\mathbb{N}^P$. (We typically miss some periodic sets.) In practice, it is not always useful to saturate a trace and, in our implementation, we use heuristics to limit the number of quantifications introduced by this operation. Actually, nothing prevents us from mixing our different kinds of generalization together, and there is still much work to be done in order to find good tactics in this case.

4 Experimental Results

We have implemented our complete approach in a tool, called SMPT (for Satisfiability Modulo P/T Nets), and made our code freely available under the GPLv3 license. The software, scripts and data used to perform our analyses are available on Github (htttps://github.com/nicolasAmat/SMPT) and are archived in Zenodo [2]. The tool supports the declaration of reachability constraints expressed using the same syntax as in the Reachability examinations of the Model Checking Contest (MCC). For instance, we use PNML as the input format for nets. SMPT relies on a SMT solver to answer `sat` and `unsat-core` queries. It interacts with SMT solvers using the SMT-LIBv2 format, which is a well-supported interchange format. We used the z3 solver for all the results presented in this section.

4.1 Evaluation on Expressiveness

It is difficult to find benchmarks with unbounded Petri nets. To quote Blondin et al. [7], "due to the lack of tools handling reachability for unbounded state spaces, benchmarks arising in the literature are primarily coverability instances". It is also very difficult to randomly generate a true invariant that does not follow, in an obvious way, from the state equation. For this reason, we decided to propose our own benchmark, made of five synthetic examples of nets, each with a given invariant. This benchmark is freely available and presented as an archive similar to instances of problems used in the MCC.

Our benchmark is made of deceptively simple nets that have been engineered to be difficult or impossible to check with current techniques. We already depicted our two first examples in Fig. 1. We display all our other examples in Figs. 2, 3 and 4.

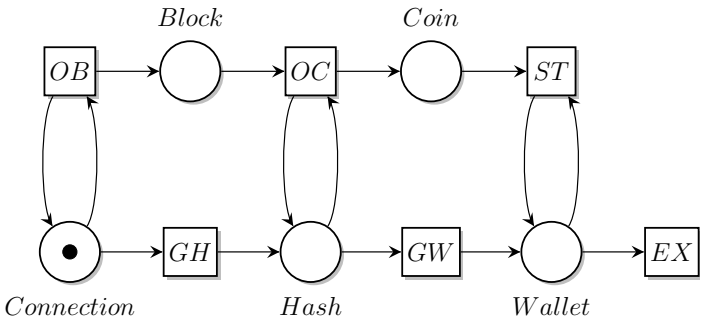

Fig. 2: CryptoMiner with $\mathbb{P} = \neg(Block = 4 \wedge Connection = 1 \wedge Coin = 10)$

Instance	SMPT	ITS-Tools	LoLA	Tapaal
Murphy	**0.75** *	TLE	TLE	TLE
PGCD	**0.11** *	139.08	TLE	TLE
CryptoMiner	0.19 *	5.92	TLE	**0.18**
Parity	0.40 *	3.36	**0.01**	4.16
Process	83.39	TLE	**0.03**	0.18

Table 1: Computation time on our synthetic examples (time in seconds).

We give a brief description of the nets composing our "reachability" benchmark (except for Parity and PGCD from Fig. 1 already described previously). Each of our example is quite small, with less than 10 places or transitions, and is representative of patterns that can make the reachability problem harder: the use of self-loops; dead transitions that cannot be detected with the state equation; weights that are relatively prime; etc. Also, most of our examples can be turned into families of nets using parameters such has the initial marking, weights on the arcs, or by adding copies of a sub-net.

- **CryptoMiner** describes the, simplified, daily schedule of someone mining bitcoins. The net is composed of two disjoint state machines synchronized by self-loops (trivial cycles of weight 1). Removing the self-loops do not modify the incidence matrix, and so do not change the solutions of the state equation. The difficulty when analysing this net lies in the presence of constraints that cannot be derived from the state equation alone. For instance, the presence of tokens in *Coin* implies *Connection* empty.
- **Process** is a net composed of three subnets coupled by self-loops on the places p_2, p_3 and p_4. The component at the bottom includes a dead transition (t_6); it will never be enabled although the state equation ensures at least one possibility of firing it. Like with our previous example, reasoning only on the state equation is not enough to capture the exact behaviour of this net.

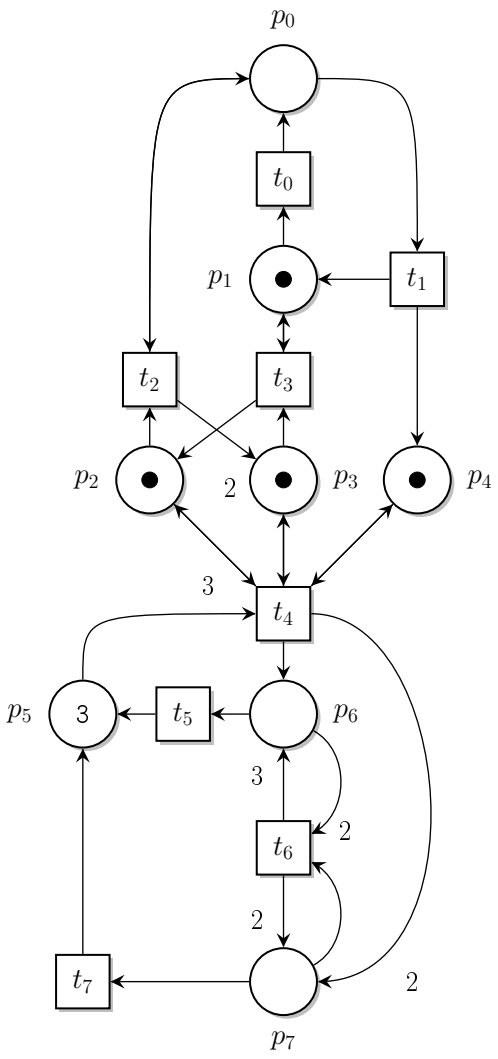

Fig. 3: Process with $\mathbb{P} = (p_2 + p_3 + p_4 \geqslant 1 \wedge p_7 \leqslant 2)$

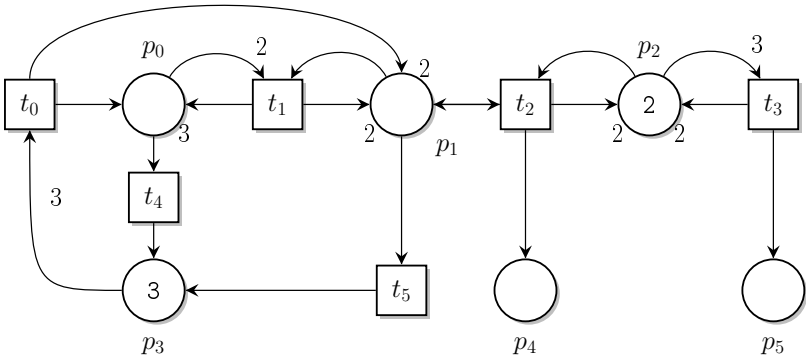

Fig. 4: Murphy with $\mathbb{P} = (p_1 \leqslant 2 \wedge p_4 \geqslant p_5)$

For instance, the state equation allows to get 3 tokens in p_7, which would contradict our invariant.

– **Murphy** is a net combining PGCD with the "bottom component" of net Process.

We compared SMPT against ITS-Tools, LoLA, and Tapaal and give our results in Table 1. All results are computed using 4 cores, a limit of 16 GB of RAM, and a timeout of 1 h. A result of TLE stands for "Time Limit Exceeded". For SMPT, we marked with an asterisk (*) the results computed using our saturation-based generalization. Our results show that SMPT is able to answer on several classes of examples that are out of reach for some, or all the other tools; often by orders of magnitude.

We also experimented with two other tools for reachability recently presented at TACAS: KReach [15], that provides a complete implementation of Kosaraju's original decision procedure, and FastForward [7], a tool for efficiently finding counter-examples in unbounded Petri nets (but that may report that an invariant is true in some cases). We do not include these tools in our findings since they were unable to answer any of our problems. (But input files for our benchmark, for both tools, are available in our artifact.)

4.2 Computing Certificate of Invariance

A distinctive feature of SMPT is the ability to output a linear inductive invariant for reachability problems: when we find that $\mathbb{P}$ is invariant, we are also able to output an inductive formula $\mathbb{C}$, of the form $\mathbb{P} \wedge G$, that can be checked independently with a SMT solver. We can find the same capability in the tool Petrinizer [16] in the case of coverability properties.

To get a better sense of this feature, we give the actual outputs computed with SMPT on the two nets of Fig. 1. The invariant for the net Parity is $\mathbb{P}_1 = (p_0 \geqslant 1)$, and for PGCD it is $\mathbb{P}_2 = (p_1 \leqslant p_2)$

The certificate for property $\mathbb{P}_1$ on Parity is $\mathbb{C}_1 \equiv (p_0 \geqslant 1) \wedge \forall k.((p_0 < 2\,k+2) \vee (p_0 \geqslant 2\,k+3))$, which is equivalent to $(p_0 \geqslant 1) \wedge (\forall k \geqslant 1).(p_0 \neq 2.k)$, meaning the marking of p_0 is odd. This invariant would be different if we changed the initial marking to an even number.

```
[PDR] Certificate of invariance
# (not (p0 < 1))
# (forall (k1) ((p0 < (2 + (k1 * 2))) or (p0 + (-2 * (k1 + 1))) >= 1))
```

The certificate for property $\mathbb{P}_2$ on PGCD is $\mathbb{C}_2 \equiv (p_1 \leqslant p_2) \wedge \forall k.((p_0 < k+3) \vee (p_2 - p_1 \geqslant k+1))$ and may seem quite inscrutable. It happens actually that the saturation "learned" the invariant $p_0 + p_1 = p_2 + 2$ and was able to use this information to strengthen property $\mathbb{P}_2$ into an inductive invariant.

```
[PDR] Certificate of invariance
# (not (p1 > p2))
# (forall (k1) ((p0 < (3 + (k1 * 1))) or ((p1 + (1 * (k1 + 1))) <= p2))
```

4.3 Evaluation on Performance

Since it is not sufficient to use only a small number of hand-picked examples to check the performance of a tool, we also provide results obtained on a set of 30 problems (a net together with an invariant) that are borrowed from test cases used by the tool Sara [32,33] (examples test$\{3, 4, 12\}$) and a similar software, called Reach, that is part of the Tina toolbox [5] (examples 1, 3u, ..., zz). Most of these problems can be easily answered, but are interesting to test our reliability on a relatively even-handed benchmark.

Our benchmark also include 6 examples of bounded nets obtained by limiting the number of times we can fire transitions in the nets PGCD and CryptoMiner. (This is achieved by adding a new place that loses a token when a transition is fired.)

The experiments were performed with the same conditions as previously, but with a timeout of only 255s. We display our results in the chart of Fig. 5, which gives the number of feasible problems, for each tool, when we change the timeout value. We also provide the computation times, for the same dataset, in Table 2. We observe that our performances are on par with Tapaal, which is the fastest among our three reference tools on this benchmark.

Our tool is actually quite mature. In particular, a preliminary version of SMPT [1] (without many of the improvements described in this work) participated in the 2021 edition of the MCC, where we ranked fourth, out of five competitors, and achieved a reliability in excess of 99.9%.

Even if it was with a previous version of our tool, there are still lessons to be learned from these results. In particular, it can inform us on the behavior of SMPT on a very large and diverse benchmark of bounded nets, with a majority of reachability properties that are not invariants.

We can compare our results with those of LoLA, that fared consistently well in the reachability category of the MCC. LoLA is geared towards model checking of finite state spaces, but it also implements semi-decision procedures for the

Instance	SMPT	ITS-Tools	LoLA	Tapaal
1	**0.15**	0.78	5.01	0.17
3u	1.84*	0.80	**0.01**	0.16
5pi	6.86	0.88	**0.01**	0.17
6pi	0.21	0.88	**0.01**	0.16
7pi	**0.15**	0.78	5.00	0.16
Crypto	0.20*	4.94	TLE	**0.16**
Crypto-10000	0.25*	4.88	**0.02**	0.16
Crypto-50	0.22*	1.04	**0.01**	0.18
Crypto-500	0.24*	4.57	**0.01**	0.17
PGCD-10000	**0.14***	142.63	TLE	96.59
PGCD-50	0.10*	0.87	**0.01**	0.17
PGCD-500	0.11*	1.13	**0.08**	0.30
b	**0.09**	0.79	5.02	0.16
kw2	0.18	0.78	5.01	**0.16**
mtx	0.60	0.86	0.00	**0.16**
nope	**0.12**	0.78	5.01	0.16
nope2	**0.10**	0.76	5.01	0.17
test12	0.10	0.76	5.00	**0.05**
test3	**0.15**	0.91	5.02	0.18
test4	0.15	0.82	**0.01**	0.17
u	**0.09**	0.77	5.00	0.17
w	**0.10**	0.79	5.02	0.16
w1	0.10	0.75	5.01	**0.05**
w2	**0.10**	0.80	5.00	0.17
wb	0.29*	0.80	5.00	**0.17**
we	**0.16**	0.78	5.01	0.17
x	1.24*	0.84	**0.01**	0.16
z	**0.10**	1.22	5.00	0.30
ze	0.71	0.88	5.03	**0.17**
zz	**0.12***	1.64	5.01	0.25

Table 2: Computation times with existing benchmarks (time in seconds)

unbounded case. Out of 45 152 reachability queries at the MCC in 2021 (one instance of a net with one formula), LoLA was able to solve 85% of them (38 175 instances) and SMPT only 52% (23 375 instances); it means approximately $\times 1.6$ more instances solved using LoLA than using SMPT. Most of the instances solved with SMPT have also been solved by LoLA; but still 1 631 instances are computed only with our tool, meaning we potentially increase the number of computed queries by 4%. This is quite an honorable result for SMPT, especially when we consider the fact that we use a single technique, with only a limited number of optimizations.

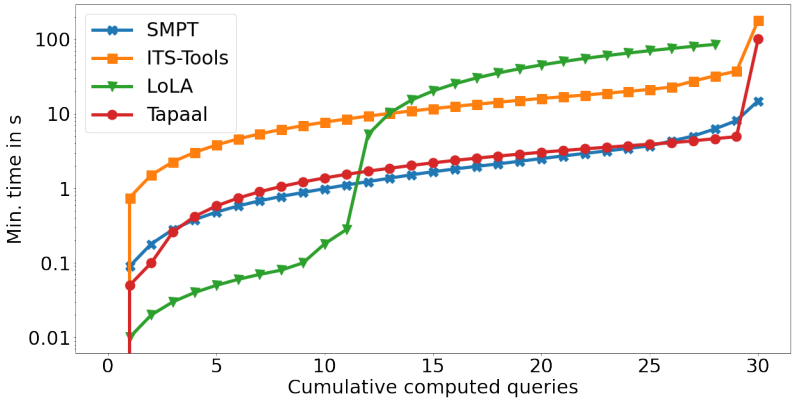

Fig. 5: Minimal timeout to compute a given number of queries.

5 Conclusion and Related Works

One of the most important results in concurrency theory is the decidability of reachability for Petri nets or, equivalently, for Vector Addition Systems with States (VASS) [25]. Even if this result is based on a constructive proof, and its "construction" streamlined over time [26], the classical Kosaraju-Lambert-Mayr-Sacerdote-Tenney approach does not lead to a workable algorithm. It is in fact a feat that this algorithm has been implemented at all, see e.g. the tool KReach [15]. While the (very high) complexity of the problem means that no single algorithm could work efficiently on all inputs, it does not prevent the existence of methods that work well on some classes of problems. For example, several algorithms are tailored for the discovery of counter-examples. We mention the tool FastForward [7] in our experiments, that explicitly targets the case of unbounded nets.

We propose a method that works as well on bounded as on unbounded ones; that behaves well when the invariant is true; and that works with "genuine" reachability properties, and not only with coverability. But there is of course no panacea. Our approach relies on the use of linear predicates, which are incrementally strengthened until we find an invariant based on: the transition relation of the net; the property we want to prove (it is "property-directed"); and constraints on the initial states. This is in line with a property proved by Leroux [26], which states that when a final configuration is not reachable then "*there exist checkable certificates of non-reachability in the Presburger arithmetic.*" Our extension of PDR provides a constructive method for computing such certificates, when it terminates. For our future works, we would like to study more precisely the completeness of our approach and/or its limits.

This is not something new. There are many tools that rely on the use of integer programming techniques to check reachability properties. We can mention

the tool Sara [33], that is now integrated inside LoLA and can answer reachability problems on unbounded nets; or libraries like Fast [4], designed for the analysis of systems manipulating unbounded integer variables. An advantage of our method is that we proceed in a lazy way. We never explicitly compute the structural invariants of a net, never switch between a Presburger formula and its representation as a semilinear set (useful when one wants to compute the "Kleene closure" of a linear constraint), ... and instead let a SMT solver work its magic.

We can also mention previous works on adapting PDR/IC3 to Petri nets. A first implementation of SMPT was presented in [1], where we focused on the integration of structural reductions with PDR. This work did not use our abstraction methods based on hurdles and saturation, which are new. We can find other related works, such as [16,23,24]. Nonetheless they all focus on coverability properties. Coverability is not only a subclass of the general reachability problem, it has a far simpler theoretical complexity (EXPSPACE vs NONELEMENTARY). It is also not expressive enough for checking the absence of deadlocks or for complex invariants, for instance involving a comparison between the marking of two places, such as $p < q$. The idea we advocate is that approaches based on the generalization of markings are not enough. This is why we believe that abstractions (G2) and (G3) defined in Lemma 2 are noteworthy.

We can also compare our approach with tools oriented to the verification of bounded Petri nets; since many of them integrate methods and semi-decision procedures that can work in the unbounded case. The best performing tools in this category are based on a portfolio approach and mix different methods. We compared ourselves with three tools: ITS-Tools [30], Tapaal [14] and LoLA [34], that have in common to be the top trio in the Model Checking Contest [3]. (And can therefore accept a common syntax to describe nets and properties.) Our main contribution in this context, and one of our most complex results, is to provide a new benchmark of nets and properties that can be used to evaluate future reachability algorithms "for expressiveness".

The methods closest to ours in these portfolios are Bounded Model Checking and k-induction [28], which are also based on the use of SMT solvers. We can mention the case of ITS-Tools [31], that can build a symbolic over-approximation of the state space, represented as set of constraints. This approximation is enough when it is included in the invariant that we check, but inconclusive otherwise. A subtle and important difference between PDR and these methods is that PDR needs only $2n$ variables (the $\boldsymbol{p}$ and $\boldsymbol{p'}$), whereas we need n fresh variables at each new iteration of k-induction (so kn variables in total). This contributes to the good performances of PDR since the complexity of the SMT problems are in part relative to the number of variables involved. Another example of over-approximation is the use of the so-called "state equation method" [27], that can strengthen the computations of inductive invariants by adding extra constraints, such as place invariants [29], siphons and traps [16,17], causality constraints, etc. We plan to exploit similar constraints in SMPT to better refine our invariants.

To conclude, our experiments confirm what we already knew: we always benefit from using a more diverse set of techniques, and are still in need of new techniques, able to handle new classes of problems. For instance, we can attribute the good results of TAPAAL, in our experiments, to their implementation of a Trace Abstraction Refinement (TAR) techniques, guided by counter-examples [10]. The same can be said with LoLA, that also uses a CEGAR-like method [33]. We believe that our approach could be a useful addition to these techniques.

Acknowledgements. We would like to thank Alex Dixon, Philip Offtermatt and Yann Thierry-Mieg for their support when evaluating their respective tools. Their assistance was essential in improving the quality of our experiments.

References

1. Amat, N., Berthomieu, B., Dal Zilio, S.: On the combination of polyhedral abstraction and SMT-based model checking for Petri nets. In: International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets). LNCS, vol. 12734. Springer (2021). https://doi.org/10.1007/978-3-030-76983-3_9
2. Amat, N., Dal Zilio, S., Hujsa, T.: SMPT (Jan 2022). https://doi.org/10.5281/zenodo.5863379
3. Amparore, E., Berthomieu, B., Ciardo, G., Dal Zilio, S., Gallà, F., Hillah, L.M., Hulin-Hubard, F., Jensen, P.G., Jezequel, L., Kordon, F., Le Botlan, D., Liebke, T., Meijer, J., Miner, A., Paviot-Adet, E., Srba, J., Thierry-Mieg, Y., van Dijk, T., Wolf, K.: Presentation of the 9th edition of the model checking contest. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Springer (2019). https://doi.org/10.1007/978-3-662-58381-4_9
4. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: acceleration from theory to practice. International Journal on Software Tools for Technology Transfer **10**(5) (2008). https://doi.org/10.1007/s10009-008-0064-3
5. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool TINA–construction of abstract state spaces for Petri nets and time Petri nets. International journal of production research **42**(14) (2004)
6. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic Model Checking without BDDs. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, Springer (1999). https://doi.org/10.1007/3-540-49059-0_14
7. Blondin, M., Haase, C., Offtermatt, P.: Directed reachability for infinite-state systems. In: Tools and Algorithms for the Construction and Analysis of Systems. LNCS, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_1
8. Bradley, A.R.: SAT-Based Model Checking without Unrolling. In: Verification, Model Checking, and Abstract Interpretation (VMCAI), LNCS, vol. 6538. Springer (2011). https://doi.org/10.1007/978-3-642-18275-4_7
9. Bradley, A.R.: Understanding IC3. In: Theory and Applications of Satisfiability Testing (SAT), LNCS, vol. 7317. Springer (2012). https://doi.org/10.1007/978-3-642-31612-8_1
10. Cassez, F., Jensen, P.G., Larsen, K.G.: Refinement of trace abstraction for real-time programs. In: International Workshop on Reachability Problems. Springer (2017). https://doi.org/10.1007/978-3-319-67089-8_4

11. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: IC3 modulo theories via implicit predicate abstraction. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_4
12. Czerwiński, W., Lasota, S., Lazić, R., Leroux, J., Mazowiecki, F.: The reachability problem for Petri nets is not elementary. Journal of the ACM (JACM) **68**(1) (2020). https://doi.org/10.1016/0304-3975(79)90041-0
13. Czerwinski, W., Orlikowski, L.: Reachability in vector addition systems is Ackermann-complete. CoRR **abs/2104.13866** (2021), https://arxiv.org/abs/2104.13866
14. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K.Y., Møller, M.H., Srba, J.: TAPAAl 2.0: Integrated development environment for timed-arc Petri nets. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_36
15. Dixon, A., Lazić, R.: Kreach: A tool for reachability in Petri nets. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 12078. Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_22
16. Esparza, J., Ledesma-Garza, R., Majumdar, R., Meyer, P., Niksic, F.: An SMT-Based Approach to Coverability Analysis. In: Computer Aided Verification (CAV). LNCS (2014). https://doi.org/10.1007/978-3-319-08867-9_40
17. Esparza, J., Melzer, S.: Verification of safety properties using integer programming: Beyond the state equation (2000). https://doi.org/10.1023/A:1008743212620
18. Finkel, A.: The minimal coverability graph for Petri nets. In: International Conference on Application and Theory of Petri Nets. Springer (1991). https://doi.org/10.1007/3-540-56689-9_45
19. Finkel, A., Haddad, S., Khmelnitsky, I.: Commodification of accelerations for the Karp and Miller construction. Discret. Event Dyn. Syst. **31**(2) (2021). https://doi.org/10.1007/s10626-020-00331-z
20. Ginsburg, S., Spanier, E.: Semigroups, Presburger formulas, and languages. Pacific journal of Mathematics **16**(2) (1966). https://doi.org/10.2140/pjm.1966.16.285
21. Hack, M.H.T.: Decidability questions for Petri Nets. Ph.D. thesis, Massachusetts Institute of Technology (1976)
22. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: International Conference on Theory and Applications of Satisfiability Testing (SAT). Springer (2012). https://doi.org/10.1007/978-3-642-31612-8_13
23. Kang, J., Bai, Y., Jiao, L.: Abstraction-based incremental inductive coverability for Petri nets. In: International Conference on Applications and Theory of Petri Nets and Concurrency. LNCS, vol. 12734. Springer (2021). https://doi.org/10.1007/978-3-030-76983-3_19
24. Kloos, J., Majumdar, R., Niksic, F., Piskac, R.: Incremental, inductive coverability. In: Computer Aided Verification (CAV). Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_10
25. Kosaraju, S.R.: Decidability of reachability in vector addition systems. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing. ACM (1982). https://doi.org/10.1145/800070.802201
26. Leroux, J.: The general vector addition system reachability problem by Presburger inductive invariants. In: 2009 24th Annual IEEE Symposium on Logic In Computer Science. IEEE (2009). https://doi.org/10.1109/LICS.2009.10
27. Murata, T.: State equation, controllability, and maximal matchings of petri nets. IEEE Transactions on Automatic Control **22**(3) (1977). https://doi.org/10.1109/TAC.1977.1101509

28. Sheeran, M., Singh, S., Stålmarck, G.: Checking Safety Properties Using Induction and a SAT-Solver. In: Formal Methods in Computer-Aided Design. LNCS, Springer, Berlin, Heidelberg (2000). https://doi.org/10.1007/3-540-40922-X_8
29. Silva, M., Terue, E., Colom, J.M.: Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In: Advanced Course on Petri Nets. Springer (1998). https://doi.org/10.1007/3-540-65306-6_19
30. Thierry-Mieg, Y.: Symbolic Model-Checking Using ITS-Tools. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_20
31. Thierry-Mieg, Y.: Structural reductions revisited. In: Application and Theory of Petri Nets and Concurrency. LNCS, vol. 12152. Springer (2020). https://doi.org/10.1007/978-3-030-51831-8_15
32. Wimmel, H.: Sara: Structures for automated reachability analysis (2013), https://github.com/nlohmann/service-technology.org/tree/master/sara
33. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. Logical Methods in Computer Science **8** (2012). https://doi.org/10.2168/LMCS-8(3:27)2012
34. Wolf, K.: Petri net model checking with LoLA 2. In: Application and Theory of Petri Nets and Concurrency. Springer (2018). https://doi.org/10.1007/978-3-319-91268-4_18