A Polyhedral Abstraction for Petri nets and its Application to SMT-Based Model Checking

Nicolas Amat, Bernard Berthomieu, Silvano Dal Zilio

A Polyhedral Abstraction for Petri nets and its Application to SMT-Based Model Checking

Nicolas Amat

LAAS-CNRS

Université de Toulouse, CNRS, INSA, Toulouse, France

Bernard Berthomieu

LAAS-CNRS

Université de Toulouse, CNRS, Toulouse, France

Silvano Dal Zilio

LAAS-CNRS

Université de Toulouse, CNRS, Toulouse, France

Abstract. We define a new method for taking advantage of net reductions in combination with a SMT-based model checker. Our approach consists in transforming a reachability problem about some Petri net, into the verification of an updated reachability property on a reduced version of this net. This method relies on a new state space abstraction based on systems of constraints, called polyhedral abstraction.

We prove the correctness of this method using a new notion of equivalence between nets. We provide a complete framework to define and check the correctness of equivalence judgements; prove that this relation is a congruence; and give examples of basic equivalence relations that derive from structural reductions.

Our approach has been implemented in a tool, named SMPT, that provides two main procedures: Bounded Model Checking (BMC) and Property Directed Reachability (PDR). Each procedure has been adapted in order to use reductions and to work with arbitrary Petri nets. We tested SMPT on a large collection of queries used in the Model Checking Contest. Our experimental results show that our approach works well, even when we only have a moderate amount of reductions.

Address for correspondence: Nicolas Amat, nicolas.amat@laas.fr, LAAS-CNRS, Vertics,
7 ave. Colonel Roche, 31031 Toulouse cedex 4, France

1. Introduction

A significant focus in model checking research is finding algorithmic solutions to avoid the "state explosion problem", that is finding ways to analyse models that are out of reach using current verification methods. To overcome this problem, it is often useful to rely on symbolic representation of the state space (like with decision diagrams) or on an abstraction of the problem, for instance with the use of logical approaches like SAT solving. We can also benefit from optimizations related to the underlying model. When analysing Petri nets, for instance, a valuable technique relies on the transformation and decomposition of nets, a method pioneered by Berthelot [1] and known as *structural reduction*.

We recently proposed a new abstraction technique based on reductions [2, 3]. The idea is to compute reductions of the form (N, E, N'), where: N is an initial net (that we want to analyse); N' is a residual net (hopefully much simpler than N); and E is a system of linear constraints. The idea is to preserve enough information in E so that we can rebuild the reachable markings of N knowing only the ones of N'. In a nutshell, we capture and abstract the effect of reductions using a set of linear constraints between the places of N and N'.

In this paper, we show that this approach works well when combined with SMT-based verification. In particular, it provides an elegant way to integrate reductions into known verification procedures. To support this statement, we provide a full theoretical framework based on the definition of a new equivalence relation between Petri nets (Sect. 3) and show how to use it for checking safety and invariant properties (Sect. 4). Our method does not impose restrictions on the syntax of nets, such as constraints on the weights of arcs or bounds on the marking of places.

We have previously applied this technique in a symbolic model checker, called TEDD, that uses Set Decision Diagrams [4] in order to generate an abstract representation for the state space of a net N. In practice, we can often reduce a Petri net N with n places (from a high dimensional space) into a residual net N' with far fewer places, say n' (in a lower-dimensional space). Hence, with our approach, we can represent the state space of N as the "inverse image", by the linear system E, of a subset of vectors of dimension n'. This technique can result in a very compact representation of the state space. We observed this effect during the recent editions of the Model Checking Contest (MCC) [5], where our tool finished at the first place for three consecutive years in the *State Space* category. In this paper, we show that we can benefit from the same "dimensionality reduction" effect when using automatic deduction procedures. Actually, since we are working with (possibly unbounded) vectors of integers, we need to consider SMT instead of SAT solvers. We show that it is enough in our case to use solvers for the theory of Quantifier-Free formulas on Linear Integer Arithmetic, what is known as QF-LIA in SMT-LIB [6].

To adapt our approach with the theory of SMT solving, we define an abstraction based on Boolean combinations of linear constraints between integer variables (representing the marking of places). This results in a new relation $N \rhd_E N'$, which is the counterpart of the tuple (N, E, N') in a SMT setting. We named this relation a *polyhedral abstraction* in reference to "polyhedral models" used in program optimization and static analysis [7, 8]. Indeed, like in these works, we propose an algebraic representation of the relation between a model and its state space based on the sets of solutions to systems of constraints. We should also often use the term *E-abstraction equivalence* to emphasize the importance of the linear system E. One of our main results is that, given a relation $N \rhd_E N'$, we can

derive a formula $\tilde{E}$ such that F is an invariant for N if and only if $\tilde{E} \wedge F$ is an invariant for the net N'. Since the residual net may be much simpler than the initial one, we expect that checking the invariant $\tilde{E} \wedge F$ on N' is more efficient than checking F on N.

Our approach has been implemented and computing experiments show that reductions are effective on a large benchmark of queries. We provide a prototype tool, called SMPT, that includes an adaptation of two procedures, Bounded Model Checking (BMC) [9] and Property Directed Reachability (PDR) [10, 11]. Each of these methods has been adapted in order to use reductions and to work with arbitrary Petri nets. We tested SMPT on a large collection of queries (13 265 test cases) used during the 2020 edition of the Model Checking Contest and participated, with our tool, in the two reachability competitions in the MCC'2021. Our experimental results show that our approach works well, even when we only have a moderate amount of reductions.

Outline and contributions. The paper is organized as follows. We start by defining the notations used in our work in Sect. 2, where we rely on a presentation of Petri net semantics that emphasizes the relationship with the QF-LIA theory. In Sect. 3, we define our notion of polyhedral abstraction and prove several of its properties. We give a description of some of the structural reductions used in our approach and show how they correspond to axioms of our polyhedral abstraction equivalence. We also prove that polyhedral abstractions are preserved by composition and transitivity, which gives a simple way to check the equivalence between two complex nets. We use these results in Sect. 4 and 5 to describe an adaptation of two SMT-based, model checking algorithms for Petri nets that can take advantage of reductions and prove their correctness. Before concluding, we report on experimental results on an extensive collection of nets and queries. Our results are quite promising. For example, on our benchmark, we observe that we are able to compute twice as many results using reductions than without.

Many results and definitions were already presented in a shorter version of the paper [12]. This extended version contains several additions that improve on the two main contributions of our work.

Concerning our definition of a polyhedral abstraction for Petri nets, we describe more precisely the reduction rules used in our approach and give more detailed proofs and definitions about the properties of our equivalence. With these additions, we give a stand-alone definition of E-abstraction equivalence that provides a more algebraic (and therefore less monolithic) approach than the one used previously in our work about computing the number of reachable states [3]. We believe that our equivalence could be reused in other settings.

Concerning our application to SMT-based model checking. We give a detailed description of our adaptation of the PDR procedure for model checking Petri nets in the case of coverability properties.

2. Petri nets and linear arithmetic constraints

Some familiarity with Petri nets is assumed from the reader. We recall some basic terminology. Throughout the text, comparison $(=, \geqslant)$ and arithmetic operations $(-, +)$ are extended pointwise to functions and tuples.

Definition 2.1. A *Petri net* N is a tuple $(P, T, \mathrm{Pre}, \mathrm{Post})$ where:

- $P = \{p_1, \ldots, p_n\}$ is a finite set of places,
- $T = \{t_1, \ldots, t_k\}$ is a finite set of transitions (disjoint from P),
- Pre : $T \to (P \to \mathbb{N})$ and Post : $T \to (P \to \mathbb{N})$ are the pre- and post-condition functions (also called the flow functions of N).

2.1. States

A state m of a net, also called a *marking*, is a mapping $m : P \to \mathbb{N}$ which assigns a number of *tokens*, $m(p)$, to each place p in P. A marked net (N, m_0) is a pair composed from a net and an initial marking m_0.

A marking m is k-bounded when each place has at most k tokens; property $\bigwedge_{p \in P} m(p) \leq k$ is true. Likewise, a marked Petri net (N, m_0) is bounded when there is k such that all reachable markings are k-bounded. A net is *safe* when it is 1-bounded. In our work, we consider *generalized* Petri nets (in which net arcs may have weights larger than 1) and we do not restrict ourselves to bounded nets.

2.2. Behaviour

A transition $t \in T$ is *enabled* at marking $m \in \mathbb{N}^P$ when $m(p) \geq \text{Pre}(t, p)$ for all places p in P. (We can also simply write $m \geqslant \text{Pre}(t)$, where $\geqslant$ stands for the component-wise comparison of markings.) A marking $m' \in \mathbb{N}^P$ is reachable from a marking $m \in \mathbb{N}^P$ by firing transition t, denoted $m \xrightarrow{t} m'$, if: (1) transition t is enabled at m; and (2) $m' = m - \text{Pre}(t) + \text{Post}(t)$.

By extension, we say that a *firing sequence* $\varrho = t_1 \ldots t_n \in T^*$ can be fired from m, denoted $m \stackrel{\varrho}{\Rightarrow} m'$, if there exist markings $m_0, \ldots, m_n$ such that $m = m_0$, $m' = m_n$ and $m_i \xrightarrow{t_{i+1}} m_{i+1}$ for all i in the range $0..n-1$.

We denote $R(N, m_0)$ the set of markings reachable from m_0 in N:

$$R(N, m_0) \triangleq \{m \mid \exists \varrho.\ m_0 \stackrel{\varrho}{\Rightarrow} m\} \tag{1}$$

The *semantics* of a marked net is the Labeled Transition System (LTS), with nodes in $R(N, m_0)$ and edges between states (m, m') whenever $m \xrightarrow{t} m'$. We focus mostly on reachable states in our work and will therefore seldom refer to the LTS of the net.

2.3. Labels and observations

In the following, we will often consider that each transition is associated with a label (a symbol taken from an alphabet Σ). In this case, we assume that a net is associated with a labeling function $l : T \to \Sigma \cup \{\tau\}$, where τ is a special symbol for the silent action name. Every net has a default labeling function l_N such that $\Sigma = T$ and $l_N(t) = t$ for every transition $t \in T$.

We can extend the notion of labels to sequences of transitions in a straightforward way. Given a relabeling function, l, we can extend it into a function from $T^\star$ into $\Sigma^\star$ such that $l(\epsilon) = \epsilon$, $l(\tau) = \epsilon$ and $l(\varrho\, t) = l(\varrho)\, l(t)$. Given a sequence of labels σ in $\Sigma^\star$, we write $(N, m) \stackrel{\sigma}{\Rightarrow} (N, m')$ when there is a firing sequence ϱ in $T^\star$ such that $(N, m) \stackrel{\varrho}{\Rightarrow} (N, m')$ and $\sigma = l(\varrho)$. We say in this case that σ is an *observation sequence* of the marked net (N, m).

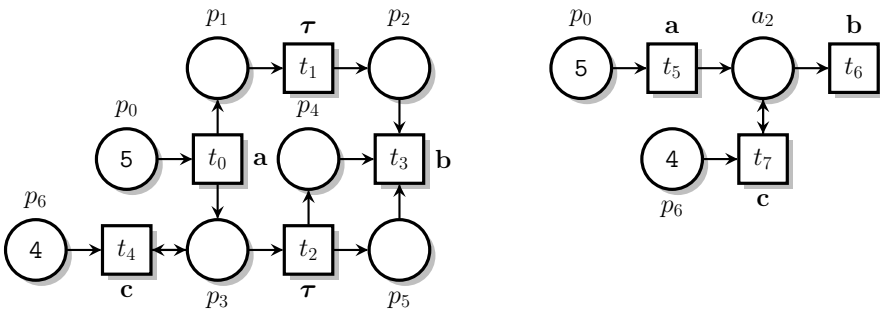


Figure 1: An example of Petri net, M_1 (left), and one of its polyhedral abstraction, M_2 (right), with $E_M \triangleq (p_5 = p_4) \wedge (a_1 = p_1 + p_2) \wedge (a_2 = p_3 + p_4) \wedge (a_1 = a_2)$.

2.4. Graphical notations

We use the standard graphical notation for nets, where places are depicted as circles and transitions as squares. For marking, we use the notation $p{*}k$ to state that place p has k tokens and we omit empty places. With the net displayed in Fig. 1 (left), and with our convention, the initial marking is $m_0 \triangleq p_0{*}5\ p_6{*}4$ (only 5 and 4 tokens in places p_0 and p_6). We have $m_0 \stackrel{\varrho}{\Rightarrow} m'_0$ with $\varrho \triangleq t_0\, t_0\, t_1\, t_1\, t_2\, t_3\, t_4$ and $m'_0 \triangleq p_0{*}3\ p_2{*}1\ p_3{*}1\ p_6{*}3$; and therefore $m_0 \xrightarrow{\mathbf{aabc}} m'_0$ when we only look at (observable) labels.

2.5. Properties

We can define many properties on the markings of a net N using Boolean combinations of linear constraints with integer variables. Assume that we have a marked net (N, m_0) with set of places $P = \{p_1, \ldots, p_n\}$. We can associate a marking m over P to the formula $\underline{m}(x_1, \ldots, x_n)$, below. In this context, an equation $x_i = k$ means that there must be k tokens in place p_i. Formula $\underline{m}$ is obviously a conjunction of literals, what is called a *cube* in [10].

$$\underline{m}(x_1, \ldots, x_n) \triangleq (x_1 = m(p_1)) \wedge \cdots \wedge (x_k = m(p_k)) \tag{2}$$

In the remainder, we use the notation $\phi(\vec{x})$ for the declaration of a formula ϕ with variables in $\vec{x}$, instead of the more cumbersome notation $\phi(x_1, \ldots, x_n)$. We also simply use $\phi(\vec{v})$ instead of $\phi\{x_1 \leftarrow v_1\} \ldots \{x_n \leftarrow v_n\}$, for the substitution of $\vec{x}$ with $\vec{v}$ in ϕ. We often use place names as variables (or parameters) and use $\vec{p}$ for the vector $(p_1, \ldots, p_n)$. We also often use $\underline{m}$ instead of $\underline{m}(\vec{p})$.

Definition 2.2. (Model of a formula)
We say that a marking m is a model of (or m *satisfies*) property ϕ, denoted $m \models \phi$, when formula $\phi(\vec{x}) \wedge \underline{m}(\vec{x})$ is satisfiable. In this case ϕ may use variables that are not necessarily in P.

We can use this approach to reframe many properties on Petri nets. For instance the notion of safe markings, described previously: a marking m is safe when $m \models \mathrm{BND}_1(\vec{x})$, where BND_k is a predicate in QF-LIA defined as:

$$\mathrm{BND}_k(\vec{x}) \triangleq \bigwedge_{i \in 1..n} (x_i \leqslant k) \tag{3}$$

Likewise, the property that transition t is enabled corresponds to the predicate ENBL_t below, in the sense that t is enabled at m when $m \models \mathrm{ENBL}_t(\vec{x})$.

$$\mathrm{ENBL}_t(\vec{x}) \triangleq \bigwedge_{i \in 1..n} (x_i \geqslant \mathrm{Pre}(t, p_i)) \tag{4}$$

Another example is the definition of *deadlocks*, which are characterized by formula $\mathrm{DEAD}(\vec{x}) \triangleq \bigwedge_{t \in T} \neg \mathrm{ENBL}_t(\vec{x})$. We give other examples in Sect. 5, when we encode the transition relation of a Petri net using formulas.

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

Definition 2.3. (Invariant and reachable properties)
Property ϕ is an invariant on (N, m_0) if and only if we have $m \models \phi$ for all $m \in R(N, m_0)$. We say that ϕ is reachable when there exists $m \in R(N, m_0)$ such that $m \models \phi$.

In our experiments, we consider the two main kinds of *reachability formulas* used in the MCC: $\mathrm{AG}\,\phi$ (true only when ϕ is an invariant), and $\mathrm{EF}\,\phi$ (true when ϕ is reachable), where ϕ is a Boolean combination of atomic properties (it has no modalities). At various times, we will use the fact that ϕ is invariant if and only if its negation is not reachable: $\mathrm{EF}\,\neg\phi$ is false.

3. Polyhedral abstraction and E-equivalence

We define a new notion, called E-*abstraction*, that is used to state a correspondence between the set of reachable markings of two Petri nets "modulo" some system of linear constraints E. Basically, we have that (N_2, m_2) is an abstraction of (N_1, m_1) when, for every sequence $m_1 \stackrel{\sigma_1}{\Longrightarrow} m_1'$ in N_1, there must exist a sequence $m_2 \stackrel{\sigma_2}{\Longrightarrow} m_2'$ in N_2 such that $E \wedge \underline{m_1'} \wedge \underline{m_2'}$ is satisfiable. We also ask that there exists such sequence for every marking m_2' such that $E \wedge \underline{m_1'} \wedge \underline{m_2'}$ is satisfiable. Therefore, knowing E, we can compute the reachable markings of N_1 from those of $\overline{N_2}$.

We also ask for the observation sequences, σ_1 and σ_2 in this case, to be equal. With the addition of this constraint, we prove that the reflexive and symmetric closure of an E-abstraction is also a congruence, which we call an E-equivalence (defined formally in Sect. 3.1).

We can illustrate these notions using the two nets M_1, M_2 in Fig. 1 and the linear constraint $E_M \triangleq (p_5 = p_4) \wedge (a_1 = p_1 + p_2) \wedge (a_2 = p_3 + p_4) \wedge (a_1 = a_2)$. Recall that marking $m_1' \triangleq p_0{*}3\ p_2{*}1\ p_3{*}1\ p_6{*}3$ is reachable in M_1. We also have that $E_M \wedge \underline{m_1'}$ entails $(p_0 = 3) \wedge (p_6 = 3) \wedge (a_2 = 1)$. Hence, if we prove that (M_1, m_1) is E_M-equivalent to $(\overline{M_2}, m_2)$, we can conclude that the marking $m_2' \triangleq a_2{*}1\ p_0{*}3\ p_6{*}3$ is reachable in M_2.

Conversely, we have several markings (exactly 4) in M_1 that correspond to the constraint $E_M \wedge \underline{m_2'} \equiv (p_5 = p_4) \wedge (p_1 + p_2 = 1) \wedge (p_3 + p_4 = 1) \wedge \underline{m_2'}$. All these markings are reachable in M_1 using the same observation sequence $\mathbf{a\,a\,b\,c}$. More generally, each marking m_2' of N_2 can be associated to

a convex set of markings of N_1, defined as the set of positive integer solutions of $E \wedge \underline{m'_2}$. Moreover, these sets form a partition of $R(N_1, m_1)$. This motivates our choice of calling this relation a *polyhedral abstraction*.

While our approach does not dictate a particular method for finding pairs of equivalent nets, we rely on an automatic approach based on the use of *structural net reductions*. When the net N_1 can be reduced, we will obtain a resulting net (N_2) and a condition (E) such that N_2 is a polyhedral abstraction of N_1. In this case, E will always be expressed as a conjunction of equality and inequation constraints between linear combinations of integer variables (the marking of places). This is why we should often use the term *reduction constraints* when referring to E. Our goal is to transform any reachability problem on the net N_1 into a reachability problem on the (reduced) net N_2, which is typically much easier to check.

3.1. Solvable systems and E-equivalence

Before defining our equivalence more formally, we need to introduce some constraints on the condition, E, used to correlate the markings of two different nets. We say that a pair of markings (m_1, m_2) are *compatible* (over respective sets of places P_1 and P_2) when they have the same number of tokens on their shared places, meaning $m_1(p) = m_2(p)$ for all p in $P_1 \cap P_2$. This is a necessary and sufficient condition for formula $\underline{m_1} \wedge \underline{m_2}$ to be satisfiable. When this is the case, we denote $m_1 \uplus m_2$ the unique marking in $(P_1 \cup P_2)$ defined by:

$$(m_1 \uplus m_2)(p) = \begin{cases} m_1(p) & \text{if } p \in P_1, \\ m_2(p) & \text{otherwise.} \end{cases} \tag{5}$$

Equipped with this notion, we can say that two markings m_1, m_2 (defined over the set of places P_1, P_2 of two nets N_1 and N_2) are "a solution" of constraints E when they are compatible with each other and $E \wedge \underline{m_1 \uplus m_2}$ is satisfiable.

This leads to the notion of *solvable system*, such that every reachable marking of N_1 can be paired with at least one reachable marking of N_2 to form a solution of E; and reciprocally.

Definition 3.1. (Solvable system of reduction constraints)
A system of linear constraints, E, is solvable for N_1, N_2 if and only if for all reachable markings m_1 in N_1 there exists at least one marking m_2 of N_2, compatible with m_1, such that $m_1 \uplus m_2 \models E$, and vice versa for every reachable marking m_2 in N_2.

In the following, when we use an E-abstraction equivalence between two marked nets (N_1, N_2), we ask that condition E be *solvable for* N_1, N_2 (see condition A2). While this property is not essential for most of our results, it simplifies our presentation and it will always be true for the reduction constraints generated with our method. On the other hand, we do not prohibit to use variables in E that are not in $P_1 \cup P_2$. Actually, such a situation will often occur in practice, when we start to chain several reductions.

We define our notion of E-abstraction as an equivalence relation between the markings reached using equal "observation sequences". An E-abstraction equivalence (shortened as E-equivalence) is an abstraction in both directions.

Definition 3.2. (E-abstraction and E-abstraction equivalence)
Assume $N_1 = (P_1, T_1, \mathrm{Pre}_1, \mathrm{Post}_1)$ and $N_2 = (P_2, T_2, \mathrm{Pre}_2, \mathrm{Post}_2)$ are two Petri nets with respective labeling functions l_1, l_2, over the same alphabet Σ, and E a system of linear constraints. We say that the marked net (N_2, m_2) is an E-abstraction of (N_1, m_1), denoted $(N_1, m_1) \sqsupseteq_E (N_2, m_2)$, if and only if:

(A1) the initial markings are compatible with E, meaning $m_1 \uplus m_2 \models E$.

(A2) for all firing sequences $(N_1, m_1) \stackrel{\varrho_1}{\Rightarrow} (N_1, m'_1)$ in N_1, there is at least one marking m'_2 over P_2 such that $m'_1 \uplus m'_2 \models E$ (e.g. solvable), and for all markings m'_2 over P_2 such that $m'_1 \uplus m'_2 \models E$ there must exist a firing sequence $\varrho_2 \in T_2^\star$ such that $(N_2, m_2) \stackrel{\varrho_2}{\Rightarrow} (N_2, m'_2)$ and $l_1(\varrho_1) = l_2(\varrho_2)$.

We say that (N_1, m_1) is E-equivalent to (N_2, m_2), denoted $(N_1, m_1) \rhd_E (N_2, m_2)$, when we have both $(N_1, m_1) \sqsupseteq_E (N_2, m_2)$ and $(N_2, m_2) \sqsupseteq_E (N_1, m_1)$.

Notice that condition (A2) is defined only for observation sequences starting from the initial marking of N_1. Hence the relation is usually not true on every pair of matched markings; it is not a bisimulation. Also, condition (A2) can be defined in an alternative way using observation sequences.

(A2) for all observation sequences σ such that $(N_1, m_1) \stackrel{\sigma}{\Rightarrow} (N_1, m'_1)$, there is at least one marking m'_2 over P_2 such that $m'_1 \uplus m'_2 \models E$, and for all markings m'_2 over P_2 such that $m'_1 \uplus m'_2 \models E$ we must have $(N_2, m_2) \stackrel{\sigma}{\Rightarrow} (N_2, m'_2)$.

By definition, relation $\rhd_E$ is symmetric. We deliberately use a "comparison symbol" for our equivalence, $\rhd$, in order to stress the fact that we expect the fact that N_2 is a reduced version of N_1. In particular, we expect that $|P_2| \leq |P_1|$.

3.2. Basic properties of polyhedral abstraction

We prove that we can use E-equivalence to check the reachable markings of N_1 simply by looking at the reachable markings of N_2. We give a first property that is useful in the context of bounded model checking, when we try to find a counter-example to a property by looking at firing sequences with increasing length. Our second property is useful for checking invariants, and is at the basis of our implementation of the PDR method for Petri nets.

Lemma 3.3. (Reachability checking)
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$. Then for all m'_1 in $R(N_1, m_1)$ there is m'_2 in $R(N_2, m_2)$ such that $m'_1 \uplus m'_2 \models E$.

Proof:
Since m'_1 is reachable, there must be a firing sequence ϱ_1 in N_1 such that $(N_1, m_1) \stackrel{\varrho_1}{\Rightarrow} (N_1, m'_1)$. By condition (A2), there must be some marking m'_2 over P_2, compatible with m'_1, such that $m'_1 \uplus m'_2 \models E$ and $(N_2, m_2) \stackrel{\varrho_2}{\Rightarrow} (N_2, m'_2)$ (for some firing sequence ϱ_2). Therefore we have m'_2 reachable in N_2 such that $m'_1 \uplus m'_2 \models E$. □

Lemma 3.3 can be used to find a counter-example m'_1, to some property F in N_1, just by looking at the reachable markings of N_2. Indeed, it is enough to find a marking m'_2 reachable in N_2 such that $m'_2 \models E \wedge \neg F$. This is the result we use in our implementation of the BMC method.

Our second property can be used to prove that every reachable marking of N_2 can be traced back to at least one marking of N_1 using the reduction constraints. (While this mapping is surjective, it is not a function, since a state in N_1 could be associated with multiple states in N_2.)

Lemma 3.4. (Invariance checking)
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$. Then for all pairs of markings m'_1, m'_2 of N_1, N_2 such that $m'_1 \uplus m'_2 \models E$ and $m'_2 \in R(N_2, m_2)$ it is the case that $m'_1 \in R(N_1, m_1)$.

Proof:
Take m'_1, m'_2 a pair of markings in N_1, N_2 such that $m'_1 \uplus m'_2 \models E$ and $m'_2 \in R(N_2, m_2)$. Hence there is a firing sequence ϱ_2 such that $(N_2, m_2) \stackrel{\varrho_2}{\Longrightarrow} (N_2, m'_2)$. By condition (A2), since $m'_1 \uplus m'_2 \models E$, there must be a firing sequence in N_1, say ϱ_1, such that $(N_1, m_1) \stackrel{\varrho_1}{\Longrightarrow} (N_1, m'_1)$. Hence $m'_1 \in R(N_1, m_1)$. □

Using Lemma 3.4, we can easily extract an invariant on N_1 from an invariant on N_2. Basically, if property $E \wedge F$ is an invariant on N_2 (where F is a formula whose variables are in P_1) then we can prove that F is an invariant on N_1. This property (the *invariant conservation* theorem of Sect. 4) ensures the soundness of the model checking technique implemented in our tool.

3.3. Composition laws

We prove that polyhedral abstraction is a transitive relation (Th 3.5) that is also closed by synchronous composition (Th 3.7) and relabeling (Th 3.8). These results can be used as a set of "algebraic laws" allowing us to derive complex equivalence assertions from much simpler instances, or *axioms*, inside arbitrary contexts. We give an example of such reasoning in Sect. 3.5.

Before defining our composition laws, we start by describing sufficient conditions in order to safely compose equivalence relations. The goal here is to avoid inconsistencies that could emerge if we inadvertently reuse the same variable in different reduction constraints.

The *fresh variables* in an equivalence statement $\mathrm{EQ} : (N_1, m_1) \rhd_E (N_2, m_2)$ are the variables occurring in E but not in $P_1 \cup P_2$. (These variables can be safely "alpha-converted" in E without changing any of our results.) We say that a net N_3 is *compatible* with respect to EQ when $(P_1 \cup P_2) \cap P_3 = \emptyset$ and there are no fresh variables of EQ that are also places in P_3. Likewise we say that the equivalence statement $\mathrm{EQ}' : (N_2, m_2) \rhd_{E'} (N_3, m_3)$ is *compatible* with EQ when $P_1 \cap P_3 \subseteq P_2$ and the fresh variables of EQ and EQ′ are disjoint.

The composition laws stated in the following theorems are useful to build larger equivalences from simpler axioms (reduction rules). We show some examples of reductions in the next section and how they occur in the example of Fig. 1.

3.3.1. Preservation by chaining

We prove that we can chain equivalences together in order to derive more general reduction rules. When doing so, we need to combine constraints together.

Theorem 3.5. Assume we have two compatible equivalence statements $(N_1, m_1) \rhd_E (N_2, m_2)$ and $(N_2, m_2) \rhd_{E'} (N_3, m_3)$, then $(N_1, m_1) \rhd_{E \wedge E'} (N_3, m_3)$.

Proof:
First, we use the fact system $E \wedge E'$ is solvable for N_1, N_3. This is a consequence of the compatibility assumption, since no fresh variable in E can clash with a fresh variable in E'. For similar reason, we have that $m_1 \uplus m_2 \models E$ and $m_2 \uplus m_3 \models E'$ entails $m_1 \uplus m_3 \models E \wedge E'$. Indeed we even have the stronger property that $\underline{m_1} \wedge \underline{m_2} \wedge \underline{m_3} \wedge E \wedge E'$ is satisfiable. From this, we obtain condition (A1) and the first constraint of condition (A2).

For the second constraint of condition (A2), we assume that σ is an observation sequence such that $(N_1, m_1) \stackrel{\sigma}{\Rightarrow} (N_1, m_1')$. Hence, using the fact that $(N_1, m_1) \rhd_E (N_2, m_2)$, we have $(N_2, m_2) \stackrel{\sigma}{\Rightarrow} (N_2, m_2')$ for every marking m_2' of N_2 such that $m_1' \uplus m_2' \models E$. Using a similar property from $(N_2, m_2) \rhd_{E'} (N_3, m_3)$, we have $(N_3, m_3) \stackrel{\sigma}{\Rightarrow} (N_3, m_3')$ for every marking m_3' of N_3 such that $m_2' \uplus m_3' \models E$. The result follows from the observation that, since E and E' are both solvable and the nets are compatible, for all markings m_1'' of N_1, if a marking m_3'' of N_3 satisfies $m_1'' \uplus m_3'' \models E \wedge E'$ then there must be a marking m_2'' of N_2 such that both $m_1'' \uplus m_2'' \models E$ and $m_2'' \uplus m_3'' \models E'$. □

3.3.2. Preservation by synchronous composition

Our next result relies on the classical synchronous product operation between labeled Petri nets [13]. Assume $N_1 = (P_1, T_1, \mathrm{Pre}_1, \mathrm{Post}_1)$ and $N_2 = (P_2, T_2, \mathrm{Pre}_2, \mathrm{Post}_2)$ are two labeled Petri nets with respective labeling functions l_1 and l_2 on the respective alphabets Σ_1 and Σ_2. We can assume, without loss of generality, that the sets P_1 and P_2 are disjoint.

We introduce a new symbol, $\circ$, used to build (structured) names for transitions that are not synchronized. The *synchronous product* between N_1 and N_2, denoted as $N_1 \| N_2$, is the net $(P_1 \cup P_2, T, \mathrm{Pre}, \mathrm{Post})$ with labelling function l where T is the smallest set containing:

- transition $(t, \circ)$ if $l_1(t) \notin \Sigma_2$, such that $l((t, \circ)) = l_1(t)$;
- transition $(\circ, t)$ if $l_2(t) \notin \Sigma_1$, such that $l((\circ, t)) = l_2(t)$;
- and transition (t_1, t_2) if $l_1(t_1) = l_2(t_2)$, such that $l((t_1, t_2)) = l_1(t_1)$.

The flow functions of $N_1 \| N_2$ are such that $\mathrm{Pre}((t_1, t_2), p) = \mathrm{Pre}_1(t_1, p)$ if $p \in P_1$ and $t_1 \neq \circ$, or $\mathrm{Pre}_2(t_2, p)$ if $p \in P_2$ and $t_2 \neq \circ$ (and 0 in all the other cases). Similarly for Post.

To simplify our proofs, we define a notion of projection over firing sequences of $N_1 \| N_2$, that is two functions $\varrho \cdot 1$ and $\varrho \cdot 2$ such that $\epsilon \cdot i = \epsilon$ and $(\varrho\, t) \cdot i = (\varrho \cdot i)\ (t \cdot i)$ for all $i \in 1..2$, where $(t_1, \circ) \cdot 1 = t$, and $(\circ, t_2) \cdot 1 = \epsilon$, and $(t_1, t_2) \cdot 1 = t_1$ (and symmetrically with $\cdot 2$ on the second component of each transition pair).

Projections can be used to extract from a firing sequence of $N_1 \| N_2$, the transitions that were fired from the left ($\cdot 1$) and right ($\cdot 2$) components of the synchronous product.

We also need to define a dual relation, denoted $\varrho_1 \| \varrho_2$, that defines the (potential) "zip merge" of firing sequences in $T_1^\star \times T_2^\star$ into firing sequences of $N_1 \| N_2$, when the two sequences can synchronize. When defined, $\varrho_1 \| \varrho_2$ is the smallest set of sequences of $N_1 \| N_2$ satisfying the following inductive rules. In particular, we say that ϱ_1 and ϱ_2 can be synchronized when $\varrho_1 \| \varrho_2 \neq \emptyset$.

- $\epsilon \| \epsilon = \{\epsilon\}$
- $(t_1\ \varrho_1) \| \epsilon = \begin{cases} \{(t_1, \circ)\ \varrho \mid \varrho \in (\varrho_1 \| \epsilon)\} & \text{if all the transitions in } t_1\ \varrho_1 \text{ have labels in } \Sigma_1 \setminus \Sigma_2, \\ \emptyset & \text{otherwise.} \end{cases}$
- $\epsilon \| (t_2\ \varrho_2) = \begin{cases} \{(\circ, t_2)\ \varrho \mid \varrho \in (\epsilon \| \varrho_2)\} & \text{if all the transitions in } t_2\ \varrho_2 \text{ have labels in } \Sigma_2 \setminus \Sigma_1, \\ \emptyset & \text{otherwise.} \end{cases}$
- $(t_1\ \varrho_1) \| (t_2\ \varrho_2) = \begin{cases} \{(t_1, t_2)\ \varrho \mid \varrho \in (\varrho_1 \| \varrho_2)\} & \text{if } l_1(t_1) = l_2(t_2), \\ \{(t_1, \circ)\ \varrho \mid \varrho \in \varrho_1 \| (t_2\ \varrho_2)\} & \text{if } l_1(t_1) \in \Sigma_1 \setminus \Sigma_2, \\ \{(\circ, t_2)\ \varrho \mid \varrho \in (t_1\ \varrho_1) \| \varrho_2\} & \text{if } l_2(t_2) \in \Sigma_2 \setminus \Sigma_1, \\ \emptyset & \text{otherwise.} \end{cases}$

We can also project the reachable markings of a synchronous product over reachable markings of each of its components. Since the places in N_1 and N_2 are disjoint, we can always see a marking m in $N_1 \| N_2$ as the disjoint union of two (necessarily compatible) markings m_1, m_2 from N_1, N_2. In this case we simply write $m = m_1 \| m_2$.

More generally, we extend this product operation to marked nets and write $(N_1, m_1) \| (N_2, m_2)$ for the marked net $(N_1 \| N_2, m_1 \| m_2)$. The following result underscores the equivalence between the semantics (the Labeled Transition System) of $N_1 \| N_2$ and the product of the LTS of its components.

Lemma 3.6. (Projection and product of sequences)
Assume there is a firing sequence $(N_1 \| N_2, m_1 \| m_2) \stackrel{\varrho}{\Rightarrow} (N_1 \| N_2, m_1' \| m_2')$ on the synchronous product $N_1 \| N_2$. Then the projections $\varrho \cdot 1$ and $\varrho \cdot 2$ are firing sequences of their respective components, $(N_i, m_i) \stackrel{\varrho \cdot i}{\Longrightarrow} (N_i, m_i')$ for all $i \in 1..2$, such that $\varrho \cdot 1$ and $\varrho \cdot 2$ can be synchronized: $\varrho \cdot 1 \| \varrho \cdot 2 \neq \emptyset$. Conversely, if $(N_i, m_i) \stackrel{\varrho_i}{\Rightarrow} (N_i, m_i')$ for all $i \in 1..2$ and $\varrho \in (\varrho_1 \| \varrho_2)$ then $(N_1 \| N_2, m_1 \| m_2) \stackrel{\varrho}{\Rightarrow} (N_1 \| N_2, m_1' \| m_2')$.

Proof:
See for instance Proposition 2.1 in [13]. □

We can now prove that E-abstraction equivalence is stable by synchronous composition. Note that it is enough to prove the results on E-abstraction, since the equivalence is symmetric.

Theorem 3.7. (Composability)
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$ and that (M, m) is compatible with respect to this equivalence, then $(N_1, m_1) \| (M, m) \rhd_E (N_2, m_2) \| (M, m)$.

Proof:
By hypothesis system E is solvable for N_1, N_2. Hence, since M is compatible, no place in the net M can occur in one of the constraints of E. Therefore E is also solvable for the pair of nets $(N_1 \| M)$ and $(N_2 \| M)$. Likewise, the initial markings $(m_1 \| m)$ and $(m_2 \| m)$ are compatible together and $(m_1 \| m) \uplus (m_2 \| m) \models E$ (the constraints in m have no effect on the constraints of E). Therefore condition (A1) is valid for the marked nets $(N_1, m_1) \| (M, m)$ and $(N_2, m_2) \| (M, m)$, and we obtain the first constraint of condition (A2).

We are left with proving the second constraint of condition (A2). Assume we have a firing sequence ϱ in $N_1 \| M$. By our projection property (Lemma 3.6) it must be the case that $(N_1 \| M, m_1 \| m) \stackrel{\varrho}{\Rightarrow} (N_1 \| M, m_1' \| m')$ with $(N_1, m_1) \stackrel{\varrho \cdot 1}{\Longrightarrow} (N_1, m_1')$. We also have that $(M, m) \stackrel{\varrho \cdot 2}{\Longrightarrow} (M, m')$ such that $(\varrho \cdot 1) \| (\varrho \cdot 2) \neq \emptyset$.

By condition (A2) on the abstraction between N_1 and N_2, it must be the case that $(N_2, m_2) \stackrel{\varrho_2}{\Rightarrow} (N_2, m_2')$, for some firing sequence ϱ_2 of N_2, for all markings m_2' of N_2 such that $m_1' \uplus m_2' \models E$. Moreover the observable sequence obtained from ϱ_2 and $\varrho \cdot 1$ are the same: $l_1(\varrho \cdot 1) = l_2(\varrho_2)$ $(\star)$, which means also that $(\varrho_2) \| (\varrho \cdot 2) \neq \emptyset$. Hence, using the second direction in Lemma 3.6, we can find a firing sequence in $\varrho_2 \| (\varrho \cdot 2)$, say ϱ', such that $(N_2 \| N_3, m_2 \| m_3) \stackrel{\varrho'}{\Rightarrow} (N_2 \| M, m_2' \| m')$. Like in the proof of condition (A1), we obtain that $(m_1' \| m') \uplus (m_2' \| m') \models E$ from the fact that $m_1' \uplus m_2' \models E$, and E is solvable, and M is compatible.

We are left to prove that ϱ and ϱ' have the same observation sequences. This is a consequence of the fact that $l_1(\varrho \cdot 1) = l_2(\varrho_2)$ (property $\star$ above); and the fact that, by construction of ϱ', we have $\varrho' \cdot 1 = \varrho_2$ and $\varrho' \cdot 2 = \varrho \cdot 2$. □

3.3.3. Preservation by relabeling

Another standard operation on labeled Petri nets is *relabeling*, denoted as $N[a/b]$, that apply a substitution to the labeling function of a net. Assume l is the labeling function over the alphabet Σ. We denote $l[a/b]$ the labeling function on $(\Sigma \setminus \{a\}) \cup \{b\}$ such that $l[a/b](t) = b$ when $l(t) = a$ and $l[a/b](t) = l(t)$ otherwise. Then $N[a/b]$ is the same as net N but equipped with labeling function $l[a/b]$. Relabeling has no effect on the marking of a net. The relabeling law is true even in the case where b is the silent action τ. In this case we say that we *hide* action a from the net.

We prove that E-abstraction equivalence is also preserved by relabeling and hiding.

Theorem 3.8. If $(N_1, m_1) \rhd_E (N_2, m_2)$ then $(N_1[a/b], m_1) \rhd_E (N_2[a/b], m_2)$.

Proof:
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$. Condition (A1) does not depend on the labels and therefore it is also true between $N_1[a/b], E$ and $N_2[a/b]$. For condition (A2), we simply use the fact that for any firing sequences ϱ_1 and ϱ_2, $l_1(\varrho_1) = l_2(\varrho_2)$ implies $l_1[a/b](\varrho_1) = l_2[a/b](\varrho_2)$. □

3.4. Reduction rules

We define a simplified set of relations that can act as "axioms" in a system for deriving E-abstraction equivalences. Each of these axioms derives from a standard *structural reduction rule* (see e.g [1, 3]), where labeled transitions play the role of interfaces with a possible outside "context".

Each rule is defined by a triplet (N_1, E, N_2) such that $(N_1, m_1) \rhd_E (N_2, m_2)$. A rule also defines possible values for the initial markings, which can be expressed using integer parameters, and may also include a condition that should be true initially.

Each of our rules corresponds to instances of the reduction system that was defined in our previous work on "counting reachable markings" [3] (we give a precise reference in each case). Hence they also correspond to instances of reduction rules implemented in our tool, called REDUCE, that can automatically find occurrences of reductions in Petri nets and apply them recursively. We give more information about this tool and the relation with our approach in Sect. 3.5. This section also contains an example showing how to apply our reduction rules to derive the equivalence stated in Fig. 1.

We consider four general families of reductions: first rules for agglomerating places (like [CONCAT] and [AGG]); then rules based on a "place invariant" over the initial net (what we call a *redundancy rule* like [RED] and [SHORTCUT]); rules for garbage collecting dead places or transitions (like [DEADT] and [REDT]); and finally rules that can be used to abstract constant or "closed" places (like [CONSTANT] and [SOURCE]).

We give a detailed proof of correctness for our first "reduction axiom", rule [CONCAT], since it is representative of the complexity of checking simple instances of E-abstraction equivalence. We do not prove similar results for all the rules defined in this section but will only give one other example, for the redundancy rule [RED]. All the correctness proofs for the reduction rules given in this section are very similar to one of these two examples.

3.4.1. Rule [CONCAT]

Our first example is the prototypical example of net reduction, as defined in [1]. It also corresponds to the simplest example of agglomeration rule (see the rule for chaining in Fig. 6 of [3]).

Rule [CONCAT], Fig. 2, can be used to fuse together two places "connected only through a deterministic transition" (modeled as a silent transition in our approach). The constraints imposed for applying this rule is that place y_2, in the initial net, must be initially empty. We also have the condition that no transition with an observable label (hence no transition that can potentially be merged with an outside context with a synchronous composition) can add a token directly to place y_2. This condition is necessary to ensure the correctness of this rule, see Proposition 3.9.

Note that nets N_1 and N_2 are not bounded, since transition a can always be fired to increase the marking of places y_1 and x. Which means that we need to consider an unbounded number of firing sequences. Therefore it may not be possible to prove this result using an automatic verification method, such as model checking. Actually, we are working on the definition of an automatic proof method for certifying the correctness of reduction rules, which would be an interesting addition to our approach.

Proposition 3.9. (Correctness of rule [CONCAT])
We have $(N_1, m_1) \rhd_E (N_2, m_2)$, with E the system containing the single equation $x = y_1 + y_2$, and N_1, N_2 the nets depicted in Fig. 2.

Proof:
The constraints on the initial marking of the nets are such that $m_1(y_1) = m_2(x) = K \geqslant 0$ and

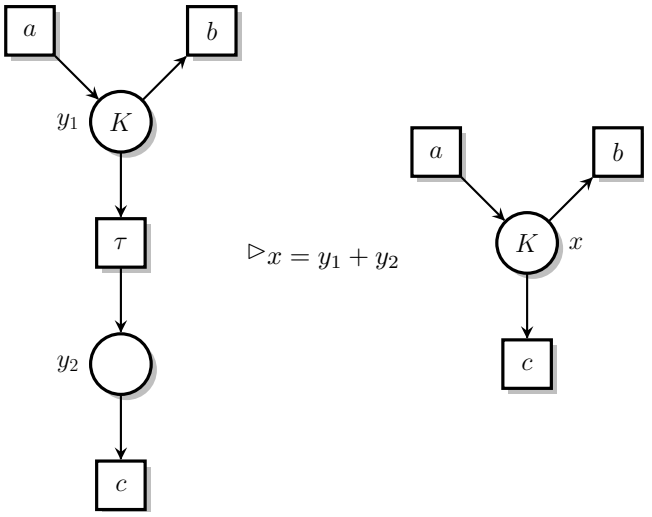


Figure 2: Rule [CONCAT].

$m_1(y_2) = 0$. To ease the presentation, we should use τ, a, b, c as the name of the transitions, and not only as labels.

We start by proving condition (A1) and the first constraint of condition (A2). By construction, we have $m_1 \uplus m_2 \models E$ and E is solvable for N_1, N_2. Indeed, equation $x = y_1 + y_2$ is always satisfiable when we fix either the values of variables y_1, y_2, or the value of x.

We now prove the second constraint of condition (A2) for the relation $(N_1, m_1) \sqsupseteq_E (N_2, m_2)$. Assume that $(N_1, m_1) \stackrel{\varrho_1}{\Longrightarrow} (N_1, m'_1)$ and that $m'_1 \uplus m'_2 \models E$. By definition of E, when m'_1 is fixed, there is a unique solution for m'_2 such that $m'_1 \uplus m'_2 \models E$; which is $m'_2(x) = m'_1(y_1) + m'_1(y_2)$. Take ϱ_2 the unique firing sequence of N_2 such that $l_1(\varrho_1) = l_2(\varrho_2)$ (basically ϱ_2 is obtained from ϱ_1 by erasing all occurrences of the silent transition). It is the case that $(N_2, m_2) \stackrel{\varrho_2}{\Longrightarrow} (N_2, m'_2)$, as needed.

We are left to prove the same constraint for the relation $(N_2, m_2) \sqsupseteq_E (N_1, m_1)$. Assume we have $(N_2, m_2) \stackrel{\varrho_2}{\Longrightarrow} (N_2, m'_2)$. We prove that there is a firing sequence ϱ_1 such that $(N_1, m_1) \stackrel{\varrho_1}{\Longrightarrow} (N_1, m'_1)$ and $l_1(\varrho_1) = l_2(\varrho_2)$, where m'_1 is the marking defined by $m'_1(y_1) = m'_2(x)$ and $m'_1(y_2) = 0$ (all the tokens are in y_1). We define ϱ_1 as the (unique) sequence obtained from ϱ_2 by adding one occurrence of the τ-transition before each occurrence of c in ϱ_2. Intuitively, we always keep all the tokens in place y_1 of N_1, except before firing a c; in which case we add a token to place y_2. We can prove, using an induction on the size of ϱ_2, that ϱ_1 is a legitimate firing sequence of (N_1, m_1) and that $(N_1, m_1) \stackrel{\varrho_1}{\Longrightarrow} (N_1, m'_1)$. □

3.4.2. Rule [AGG]

Our second example of rule is for the *agglomeration of places*, see Fig. 3, that can be used to simplify a "cluster of places" between which tokens can move freely from y_2 to y_1. This is an instance of the general "loop agglomeration" rule given in Fig. 7 of [3].

We could easily define a family of reduction rules similar to [AGG] and [CONCAT] but for longer

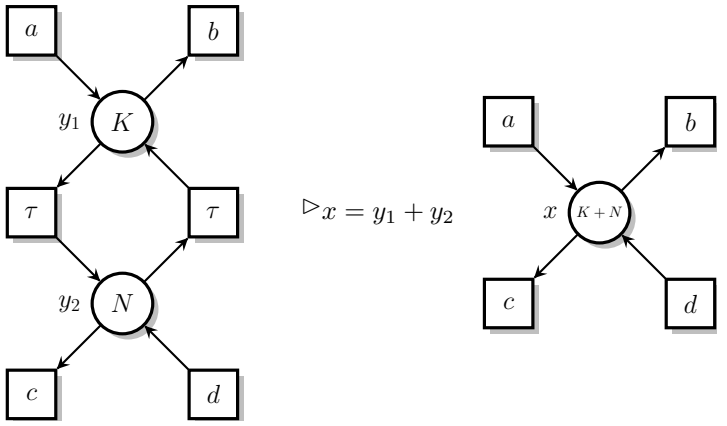


Figure 3: Rule [AGG].

"loops" or "chains" of places, or with the addition of weights on the arcs. For the sake of brevity, we only list one archetypal instance of each rule in this section.

3.4.3. Rules [RED] and [SHORTCUT]

Our next two rules, Fig. 4, are reductions that can be used to eliminate *redundant places*, meaning places whose marking derives from a place invariant (and the knowledge of the marking of other places).

In rule [RED] for instance, with the assumption that we have more tokens in place z than in y initially, it is always the case that $m(z) - m(y)$ is a constant for all the reachable states m. Hence we can safely eliminate z and keep the relevant information in our linear system E.

Rule [SHORTCUT] gives a more involved example, that relies on a condition involving more than two places; an invariant of the form $z = y_1 + y_2 + K$.

We give the proof of correctness for the equivalence corresponding to rule [RED]. The proofs for other redundant place elimination rules are all similar.

Proposition 3.10. (Correctness of rule [RED])
Assuming $K \leqslant N$, we have $(N_1, m_1) \rhd_E (N_2, m_2)$, with E the system containing the single equation $x = y + N - K$, and N_1, N_2 the nets depicted in Fig. 4 (above).

Proof:
Condition $K \leqslant N$ is necessary in order to have that $N - K \geqslant 0$, and therefore that the marking of y in N_2 is indeed non-negative.

By construction, we have $m_1 \uplus m_2 \models E$, satisfying condition (A1). The first constraint of condition (A2) follows from the fact that $z = y + N - K$ is an invariant on (N_1, m_1), meaning that for all firing sequence ϱ such that $(N_1, m_1) \stackrel{\varrho}{\Rightarrow} (N_1, m_1')$ we have $m_1'(z) = m_1'(y) + N - K$. This can be proved by a simple induction on the length of ϱ. Hence, E is satisfied for every reachable marking in (N_1, m_1), and so E is solvable.

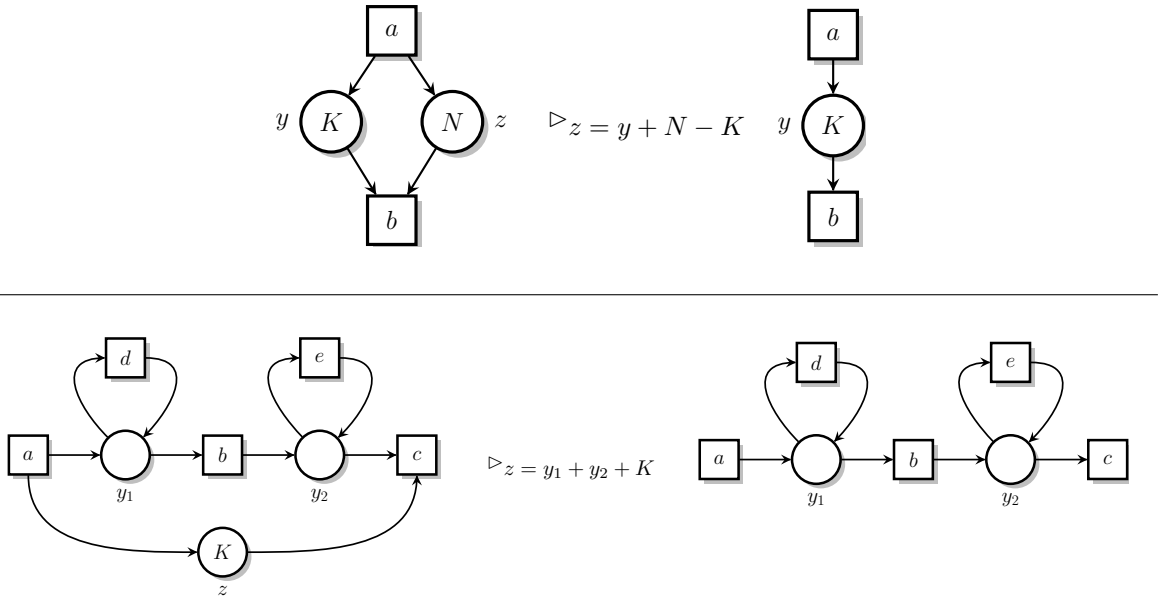


Figure 4: Rule [RED] (above), assuming $K \leqslant N$, and rule [SHORTCUT] (below).

We now prove the second constraint of condition (A2) for the relation $(N_1, m_1) \sqsupseteq_E (N_2, m_2)$. Assume that $(N_1, m_1) \stackrel{\varrho}{\Rightarrow} (N_1, m_1')$. We have that ϱ is also a firing sequence of (N_2, m_2) and, moreover, $(N_2, m_2) \stackrel{\varrho}{\Rightarrow} (N_2, m_2')$ such that $m_2'(y) = m_1'(y)$. The proof is similar in the other direction. □

3.4.4. Rules [REDT] and [DEADT]

We can use the same approach to simplify transitions in a net, rather than places. One such example is rule [REDT], to remove redundant transitions. Such rules are interesting because, when applied in collaboration with others, they can create new opportunities to apply reductions. We give an example of such mechanism in the example of Sect. 3.5.

Another example is the elimination of dead transitions, rule [DEADT], that can get rid of transitions that are "structurally dead". In this example, we know that place x will always stay empty since no transition can increase its marking. Hence the τ transition is dead and we can remove it without modifying the set of reachable markings nor the observation sequences.

3.4.5. Rules [CONSTANT] and [SOURCE]

Our last examples of rules illustrate the case of equivalences $(N_1, m_1) \triangleright_E (N_2, m_2)$ where the final Petri net is "empty" (denoted $\emptyset$). A Petri net with an empty set of places has only one marking; the empty mapping, the only function with domain $\emptyset \to \mathbb{N}$.

In this case the reachable markings of (N_1, m_1) are exactly defined by the non-negative solutions of system E.

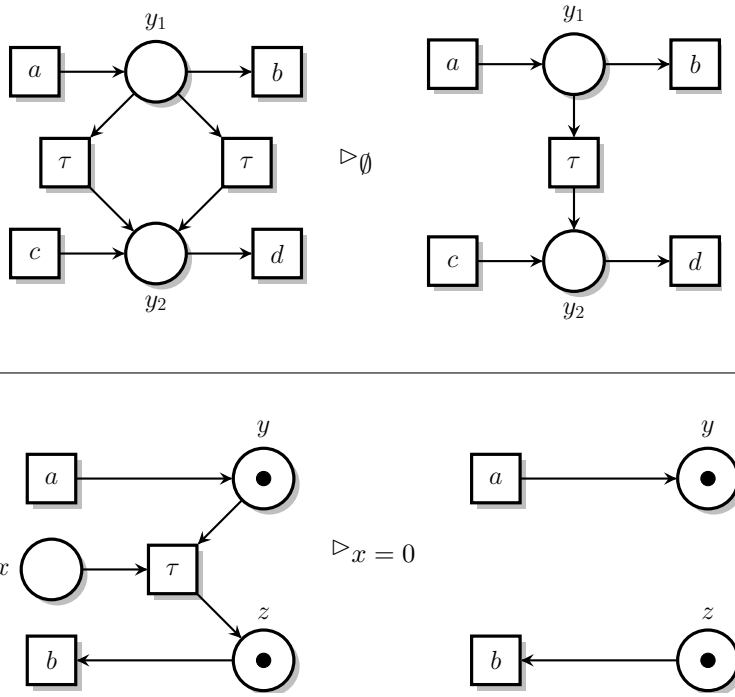


Figure 5: Rules [REDT] (above), and [DEADT] (below).

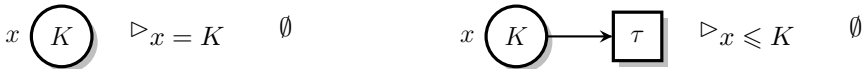


Figure 6: Rules [CONSTANT] (left) and [SOURCE] (right).

Such cases may occur in practice when we can apply several reductions in a row. We say that the initial net is "fully reducible". In example [SOURCE] for instance, we can abstract the state space of the initial net with the single constraint $x \leqslant K$.

We have other rules that allow us to fully reduce a net. For instance specific structural or behavioural restrictions, such as nets that are marked graphs or other cases where the set of reachable markings is exactly defined by the solutions of the state equation [14].

3.5. Deriving E-equivalences using reductions

We can compute net reductions by reusing a tool, called REDUCE, that was developed in our previous work [3] (see also Sect. 6). The tool takes a marked Petri net as input and returns a reduced net and a sequence of linear constraints. For example, given the net M_1 of Fig. 1, REDUCE returns net M_2 and equations $(p_5 = p_4), (a_1 = p_1 + p_2), (a_2 = p_3 + p_4)$, and $(a_1 = a_2)$, that corresponds to formula E_M in Fig. 1.

The tool works by applying successive reduction rules, in a compositional way. We give an ex-

ample of this mechanism in Fig. 7, where we show the four reduction steps involved in our running example.

The first step is a direct application of rule [RED] inside a larger context; in each case we use colours to emphasize the sub-net where the rule is applied. The two following ones are variations of rule [CONCAT]. Each rule introducing a fresh "place variable", a_1 and a_2. Finally, after simplification, we obtain a net with a new opportunity to apply a redundancy rule.

It is possible to prove that each reduction step computed by REDUCE, from a net (M_i, m_i) to (M_{i+1}, m_{i+1}) with constraints E_i, is such that $(M_i, m_i) \rhd_{E_i} (M_{i+1}, m_{i+1})$. From Th. 3.5, we have $(M_0, m_0) \rhd_E (M_n, m_n)$, i.e., the results computed by REDUCE always translate into valid polyhedral abstractions.

In conclusion, we can use REDUCE to compute polyhedral abstractions automatically. In the other direction, we can use our notion of equivalence to prove the correctness of new reduction patterns that could be added in the tool. While it is not always possible to reduce the complexity of a net using this approach, we observed in our experiments (Sect. 6) that, on a benchmark suite that includes almost $1\,000$ instances of nets, about half of them can be reduced by a factor of more than 30%.

4. SMT-based model checking using abstractions

We introduce a general method for combining polyhedral abstractions with SMT-based model checking procedures. Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$, where the nets N_1, N_2 have sets of places P_1, P_2 respectively. In the following, we use $\vec{p_1} \triangleq (p_1^1, \ldots, p_k^1)$ and $\vec{p_2} \triangleq (p_1^2, \ldots, p_l^2)$ for the places in P_1 and P_2. We also consider (disjoint) sequences of variables, $\vec{x}$ and $\vec{y}$, ranging over (the places of) N_1 and N_2. With these notations, we denote $\tilde{E}(\vec{x}, \vec{y})$ the formula obtained from E where place names in N_1 are replaced with variables in $\vec{x}$, and place names in N_2 are replaced with variables in $\vec{y}$. When we have the same place in both nets, say $p_i^1 = p_j^2$, we also add the constraint $(x_i = y_j)$ to $\tilde{E}$ in order to avoid shadowing variables. (Remark that $\tilde{E}(\vec{p_1}, \vec{p_2})$ is equivalent to E, since equalities $x_i = y_j$ become tautologies in this case.)

$$\tilde{E}(\vec{x}, \vec{y}) \triangleq E\{\vec{p_1} \leftarrow \vec{x}\}\{\vec{p_2} \leftarrow \vec{y}\} \wedge \bigwedge_{\{(i,j)|p_i^1 = p_j^2\}} (x_i = y_j) \tag{6}$$

Given a formula F, we denote $fv(F)$ the set of free variables contained in it. Assume F_1 is a property that we want to study on N_1, without loss of generality we can enforce the condition $(fv(F_1) \setminus P_1) \cap (fv(E) \setminus P_1) = \emptyset$ (meaning we can always rename the variables in F_1 and E that are not places in N_1). This condition ensures that the property checked on the initial net does not inadvertently contain new variables introduced during the reduction.

Definition 4.1. **(E-transform formula)**
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$ and take F_1 a property with variables in P_1 such that $(fv(F_1) \setminus P_1) \cap (fv(E) \setminus P_1) = \emptyset$. Formula $F_2(\vec{y}) \triangleq \exists \vec{x}.\ \tilde{E}(\vec{x}, \vec{y}) \wedge F_1(\vec{x})$ is the E-transform of F_1.

The following property states that, to check an invariant F_1 on the reachable markings of N_1, it is enough to check the corresponding E-transform formula F_2 on the reachable markings of N_2.

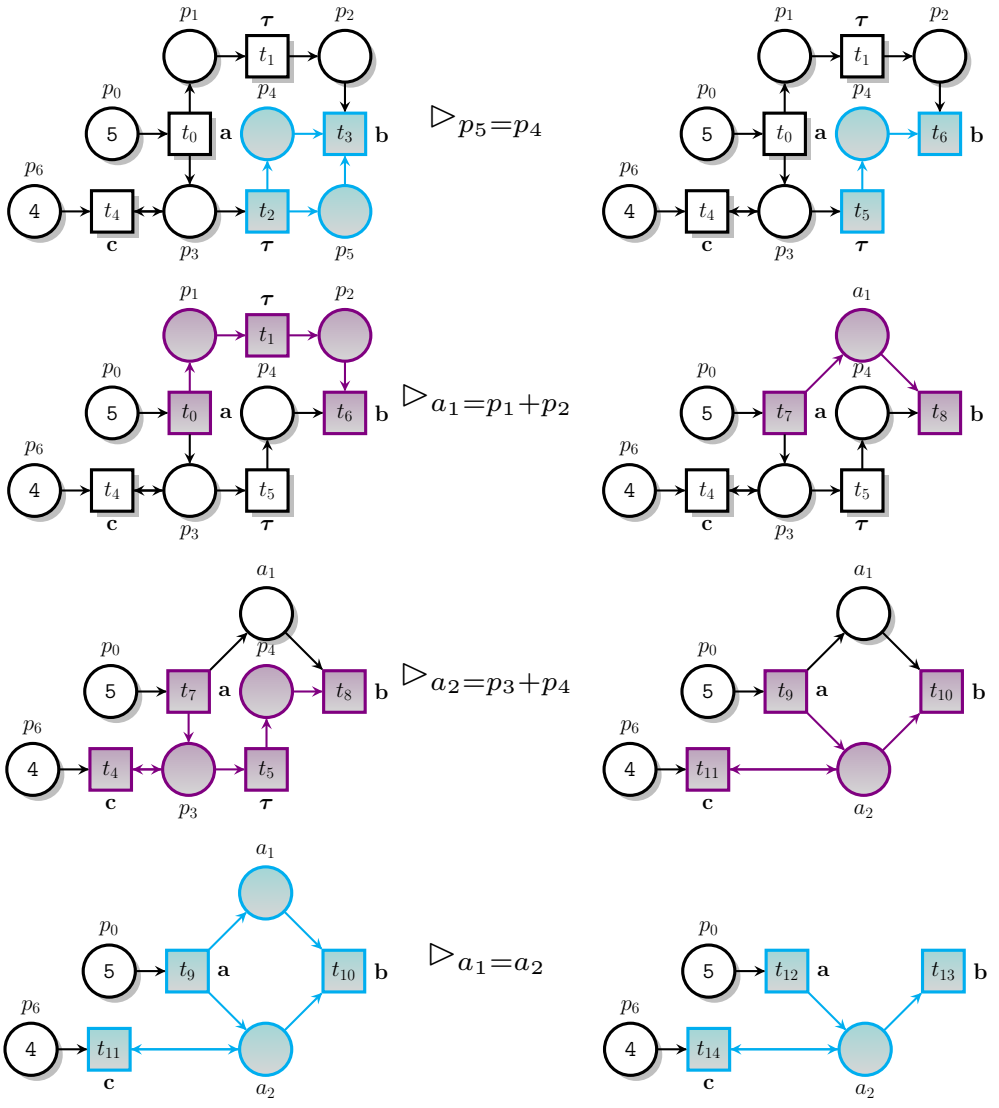


Figure 7: Example of sequence of four reductions leading from the net N_1 to N_2 from Fig. 1.

Theorem 4.2. (Invariant conservation)
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$ and that F_2 is the E-transform of formula F_1 on N_1. Then F_1 is an invariant on N_1 if and only if F_2 is an invariant on N_2.

Proof:
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$ and property F_1 is an invariant on N_1. Consider m'_2 a reachable marking in N_2. By definition of E-abstraction, we have at least one reachable marking m'_1 in N_1 such that $m'_1 \uplus m'_2 \models E$. Since F_1 is an invariant on N_1 we have $m'_1 \models F_1$. The condition $m'_1 \uplus m'_2 \models E$ is equivalent to $\underline{m'_1} \wedge \underline{m'_2} \wedge E$ satisfiable. By definition we have $\tilde{E}(\vec{p_1}, \vec{p_2}) \equiv E$, which implies $\underline{m'_1}(\vec{p_1}) \wedge \underline{m'_2}(\vec{p_2}) \wedge \tilde{E}(\vec{p_1}, \vec{p_2}) \wedge F_1(\vec{p_1})$ satisfiable, since the only variables that are both in F_1 and E must also be in N_1. Hence, m'_2 satisfies the E-transform formula of F_1. The proof is similar in the other direction. □

Since F_1 invariant on N_1 is equivalent to $\neg F_1$ not reachable, we can directly infer an equivalent conservation theorem for reachability: to find a model of F_1 in N_1, it is enough to find a model for $F_1(\vec{p_1}) \wedge \tilde{E}(\vec{p_1}, \vec{p_2})$ in N_2.

Theorem 4.3. (Reachability conservation)
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$ and that F_2 is the E-transform of formula F_1 on N_1. Then formula F_1 is reachable in N_1 if and only if F_2 is reachable in N_2.

Proof:
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$ and property F_1 is reachable in N_1. Hence, there exists a reachable marking m'_1 in N_1 such that $m'_1 \models F_1$. By definition of E-abstraction, we have at least one reachable marking m'_2 in N_1 such that $m'_1 \uplus m'_2 \models E$. The condition $m'_1 \uplus m'_2 \models E$ is equivalent to $\underline{m'_1}(\vec{p_1}) \wedge \underline{m'_2}(\vec{p_2}) \wedge E$ satisfiable. By definition we have $\tilde{E}(\vec{p_1}, \vec{p_2}) \equiv E$, which implies $\underline{m'_1} \wedge \underline{m'_2} \wedge \tilde{E}(\vec{p_1}, \vec{p_2}) \wedge \underline{F_1}(\vec{p_1})$ satisfiable, since the only variables that are both in F_1 and E must also be in N_1. Hence, m'_2 satisfies the E-transform formula of F_1. The proof is similar in the other direction. □

5. BMC and PDR implementation

We developed a prototype model checker that takes advantage of net reductions. The tool includes two main verification procedures that have been developed for generalized Petri nets. (No specific optimizations are applied when we know the net is safe, like for instance using Boolean formulas instead of QF-LIA.) These procedures correspond to instantiations of the BMC and PDR methods for checking general reachability properties on Petri nets. We sketch these two procedures below.

5.1. Encoding Petri nets semantics using Linear Integer Arithmetic

Our approach is based on a revisit of the semantics of Petri nets using Linear Integer Arithmetic formulas.

We already defined (Sect. 2) a helper formula, or *operator*, $\mathrm{ENBL}_t(\vec{x})$ such that $\mathrm{ENBL}_t(\vec{x}) \wedge \underline{m}(\vec{x})$ is true when t is enabled at m. We can define, in the same way, a linear predicate to describe the

relation between the markings before and after some transition t fires. To this end, we use a vector $\vec{x}'$ of "primed variables" $(x'_1, \ldots, x'_n)$, where x'_i will stand for the marking of place p_i after a transition is fired.

With this convention, formula $\mathrm{FIRE}_t(\vec{x}, \vec{x}')$ is such that $\mathrm{FIRE}_t(m, m')$ entails $m \xrightarrow{t} m'$ when t is enabled at m, or $m = m'$.

With all these notations, we can define a predicate $\mathrm{T}(\vec{x}, \vec{x}')$ that "encodes" the effect of firing at most one transition in the net N. By construction, formula $\underline{m}(\vec{x}) \wedge \mathrm{T}(\vec{x}, \vec{x}') \wedge \underline{m'}(\vec{x}')$ is true when $m \to m'$, or when $m = m'$.

$$\mathrm{ENBL}_t(\vec{x}) \triangleq \bigwedge_{i \in 1..n} (x_i \geqslant \mathrm{Pre}(t)(p_i)) \tag{7}$$

$$\Delta_t(\vec{x}, \vec{x}') \triangleq \bigwedge_{i \in 1..n} (x'_i = x_i + \mathrm{Post}(t, p_i) - \mathrm{Pre}(t, p_i)) \tag{8}$$

$$\mathrm{EQ}(\vec{x}, \vec{x}') \triangleq \bigwedge_{i \in 1..n} x_i = x'_i \tag{9}$$

$$\mathrm{FIRE}_t(\vec{x}, \vec{x}') \triangleq \mathrm{EQ}(\vec{x}, \vec{x}') \vee \big(\mathrm{ENBL}_t(\vec{x}) \wedge \Delta_t(\vec{x}, \vec{x}')\big) \tag{10}$$

$$\mathrm{T}(\vec{x}, \vec{x}') \triangleq \mathrm{EQ}(\vec{x}, \vec{x}') \vee \bigvee_{t \in T} (\mathrm{ENBL}_t(\vec{x}) \wedge \Delta_t(\vec{x}, \vec{x}')) \tag{11}$$

5.2. Bounded Model Checking (BMC)

BMC is an iterative method for exploring the state space of finite-state systems by unrolling their transitions [9]. The method was originally based on an encoding of transition systems into (a family of) propositional logic formulas and the use of SAT solvers to check these formulas for satisfiability [15]. There are several works that adapt BMC to Petri nets, such as [16]. More recently, this approach was extended to more expressive models, and richer theories, using SMT solvers [17]. Our goal is not to develop a state of the art BMC model checker for generalized Petri nets. Instead, we develop a textbook implementation that is enough to test the impact on performances when using reductions.

5.2.1. Description of the algorithm

In BMC, we try to find a reachable marking m that is a model for a given formula F, that usually models a set of "feared events". The algorithm (see function `BMC`) starts by computing a formula, say ϕ_0, representing the initial marking and checking whether $\phi_0 \wedge F$ is satisfiable (meaning F is initially true). If the formula is unsatisfiable, we compute a formula ϕ_1 representing all the markings reachable in one step, or less, from the initial marking and check $\phi_1 \wedge F$. This way, we compute a sequence of formulas $(\phi_i)_{i \in \mathbb{N}}$ until either $\phi_i \wedge F$ is satisfiable (in which case a counter-example is found) or we have $\phi_{i+1} \Rightarrow \phi_i$ (in which case we reach a fixed point and no counter-example exists). The BMC method is not complete since it is not possible, in general, to bound the number of iterations needed to give an answer. Also, when the net is unbounded, we may very well have an infinite sequence of formulas $\phi_0 \subsetneq \phi_1 \subsetneq \ldots$ However, in practice, this method can be very efficient to find a counter-example when it exists.

The crux of the method is to compute formulas ϕ_i that represent the set of markings reachable using firing sequences of length at most i. We show how we can build such formulas incrementally. We assume that we have a marked net (N, m_0) with places $P = \{p_1, \ldots, p_n\}$ and transitions $T =$

$\{t_1, \ldots, t_k\}$. In the remainder of this section, we build formulas that express constraints between markings m and m' such that $m \to m'$ in N. Hence we define formulas with $2\,n$ variables. We use the notation $\psi(\vec{x}, \vec{x}')$ as a shorthand for $\psi(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$.

Formula ϕ_i is the result of connecting i successive occurrences of formulas of the form $\mathrm{T}(\vec{x}_j, \vec{x}_{j+1})$. We define the formulas inductively, with a base case (ϕ_0) which states that only m_0 is reachable initially. To define the ϕ_i's, we assume that we have a collection of (pairwise disjoint) sequences of variables, $(\vec{x}_i)_{i\in\mathbb{N}}$. In the following listings, we use the auxiliary function `freshVariables` to iterate over this family of variable vectors.

$$\phi_0(N, m_0) \triangleq \underline{m_0}(\vec{x}_0) \qquad \phi_{i+1}(N, m_0) \triangleq \phi_i(N, m_0) \wedge \mathrm{T}(\vec{x}_i, \vec{x}_{i+1})$$

Function BMC(EF F: linear predicate)

Result: $\top$ if F is reachable (meaning $\neg F$ is not an invariant)

```
1 x⃗ ← freshVariables()
2 φ ← m_0(x⃗)
3 while unsat(m_0(x⃗) ∧ φ ∧ (¬F)(x⃗)) do
4 |   x⃗' ← freshVariables()
5 |   φ ← φ ∧ T(x⃗, x⃗')
6 |   x⃗ ← x⃗'
7 return ⊤
```

We can prove that this family of BMC formulas provide a way to check reachability properties, meaning that formula F is reachable in (N, m_0) if and only if there exists $i \geqslant 0$ such that $F(\vec{x}_i) \wedge \phi_i(N, m_0)$ is satisfiable. The approach we describe here is well-known (see for instance [9]). It is also quite simplified. Actual model checkers that rely on BMC apply several optimization techniques, such as compositional reasoning; acceleration methods; or the use of invariants on the underlying model to add extra constraints. We do not consider such optimizations here, on purpose, since our motivation is to study the impact of polyhedral abstractions. We believe that our use of reductions is orthogonal and does not overlap with many of these optimizations, in the sense that we do not preclude them, and that the performance gain we observe with reductions could not be obtained with these optimizations.

5.2.2. Combination with polyhedral abstraction

Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$. We denote $\mathrm{T}_1, \mathrm{T}_2$ the equivalent of formula T, above, for the nets N_1, N_2 respectively. We also use $\vec{x}$, $\vec{y}$ for sequences of variables ranging over (the places of) N_1 and N_2 respectively. We should use $\phi(N_1, m_1)$ for the family of formulas built using operator T_1 and variables $\vec{x}_0, \vec{x}_1, \ldots$ and similarly for $\phi(N_2, m_2)$, where we use T_2 and variables of the form $\vec{y}_i$.

The following property states that, to find a model of F in the reachable markings of N_1 (meaning EF F true), it is enough to find a model for its E-transform in N_2.

Theorem 5.1. (BMC with E-transform)
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$ and that F_2 is the E-transform of F_1. Formula F_1 is reachable in N_1 if and only if there exists $j \geqslant 0$ such that $F_2(\vec{y}_j) \wedge \phi_j(N_2, m_2)$ is satisfiable.

Proof:
Our proof relies on the property that BMC is sound and complete for finding a finite counter-example (see e.g.[18]): there is a firing sequence ϱ, of size less than i, such that $m_1 \stackrel{\varrho}{\Rightarrow} m'_1$ and $m'_1 \models F_1$— meaning property F_1 is reachable in N_1— if and only if $F_1 \wedge \phi_i(N_1, m_1)$. We can prove this property by induction on the value of i and use the fact that $m \Rightarrow m'$ or $m = m'$ in N_1 entails $\mathrm{T}_1(m, m')$.

By our *conservation of reachability* theorem (Th. 4.3), property F_1 is reachable in N_1 (say with a counter-example of size i) if and only if property F_2 is reachable in N_2 (say with a counter-example of size j). Therefore there exists i such that $F_1(\vec{x}_i) \wedge \phi_i(N_1, m_1)$ is satisfiable if and only if there exists j such that $F_2(\vec{y}_j) \wedge \phi_j(N_2, m_2)$ is satisfiable. □

We can give a stronger result, comparing the value of i and j, when the reductions used in proving the E-abstraction equivalence never introduce new transitions. This is the case, for example, with the reductions computed using the REDUCE tool. Indeed, in this case, we can show that we may find a witness of length i in N_1 (a firing sequence of length i showing that F_1 is reachable in N_1) when we find a witness of length $j \leqslant i$ in N_2. This is because, in this case, reductions may compact a sequence of several transitions into a single one or, at worst, not change it. Take the example of the [CONCAT] rule in Fig. 7. Therefore BMC benefits from reductions in two ways. First because we can reduce the size of formulas ϕ (which are proportional to the size of the net), but also because we can accelerate transition unrolling in the reduced net.

5.3. Property Directed Reachability (PDR)

While BMC is the right choice when we try to find counter-examples, it usually performs poorly when we want to check an invariant property, $\mathrm{AG}\, F$. There are techniques that are better suited to prove *inductive invariants* in a transition system; that is a property that is true initially and stays true after firing any transition.

In order to check invariants with SMPT, we have implemented a method called PDR [10, 11] (also known as IC3), which incrementally generates clauses that are inductive "relative to stepwise approximate reachability information". PDR is a combination of induction, over-approximation, and SAT solving. For SMPT, we developed a similar method that uses SMT solving, to deal with markings and transitions, and that can take advantage of polyhedral abstractions.

As with BMC, our focus is not to study the performances of PDR per se, but its combination with polyhedral abstractions. We give more details about our adaptation of PDR for generalized Petri nets in [19]. We keep enough information about our implementation in order to state an equivalent of Th. 5.1 for PDR.

We use similar notations as with BMC, but with a small difference. Indeed, since PDR "unrolls at most one transition" at a time, we only need two vectors of variables instead of a family $(\vec{x}_i)_{i \geqslant 0}$ like with BMC: we use unprimed variables ($\vec{x}$) to represent states before firing a transition and primed variables ($\vec{x}'$) to represent the reached states.

PDR requires to define a set of *safe states*, described as the models of some property F. It also requires a set of initial states, I. In our case $I \triangleq \underline{m_0}$.

The procedure is complete for finite transition systems, for instance with bounded Petri nets. We can also prove termination in the general case when property $\neg F$ is *monotonic*, meaning that $m \models \neg F$ implies that $m' \models \neg F$ for all markings m' that cover m (that is when $m' \geqslant m$, component-wise). 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 (see e.g. [20]).

A formula F is *inductive* [11] when $I(\vec{x}) \Rightarrow F(\vec{x})$ and $F(\vec{x}) \wedge \mathrm{T}(\vec{x}, \vec{x}') \Rightarrow F(\vec{x}')$ are tautologies. It is *inductive relative* to formula G if both $I(\vec{x}) \Rightarrow F(\vec{x})$ and $G(\vec{x}) \wedge F(\vec{x}) \wedge \mathrm{T}(\vec{x}, \vec{x}') \Rightarrow F(\vec{x}')$ are tautologies. With PDR we compute *Over Approximated Reachability Sequences* (OARS), meaning sequences of formulas $(F_0, \ldots, F_{k+1})$, with variables in $\vec{x}$, that are monotonic: $I(\vec{x}) \Rightarrow F_0(\vec{x})$, $F_i(\vec{x}) \Rightarrow F_{i+1}(\vec{x})$ for all $i \in 0..k$, and $F_{k+1} \Rightarrow F$; and satisfies *consecution*: $F_i(\vec{x}) \wedge \mathrm{T}(\vec{x}, \vec{x}') \Rightarrow F_{i+1}(\vec{x}')$ for all $i \leqslant k+1$. The formulas F_i change at each iteration of the procedure (each time we increase k). The procedure stops when we find an index i such that $F_i = F_{i+1}$. In this case we know that F is an invariant. We can also stop during the iteration if we find a counter-example.

5.3.1. Description of the algorithm

Our implementation follows closely the algorithm for IC3 described in [11]. We only give the pseudo-code for the four main functions (`prove`, `strengthen`, `inductivelyGeneralize` and `pushGeneralization`).

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

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 all $i \in 0..k$, 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 F. We can also stop during the iteration if we find a counter-example (a model m of $\neg F$). In this case, we can also return a trace leading to m.

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

Function prove(AG F: linear predicate)

Result: $\top$ if F is an invariant, otherwise $\bot$ (meaning $\neg F$ is reachable)

```
1  if sat(I(x⃗) ∧ T(x⃗, x⃗′) ∧ (¬F)(x⃗′)) then
2  |  return ⊥
3  k ← 1, F_0 ← I, F_1 ← F
4  while ⊤ do
5  |  if 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
```

Function strengthen(k : current level)

```
1  try:
2  |  while (m --t--> m′) ⊨ F_k(x⃗) ∧ T(x⃗, x⃗′) ∧ (¬F)(x⃗′) do
3  |  |  m̂ ← generalizeWitness(m)
4  |  |  n ← inductivelyGeneralize(m̂, k − 2, k)
5  |  |  pushGeneralization({(m̂, n + 1)}, k)
6  |  return ⊤
7  catch counter example:
8  |  return ⊥
```

Instead of blocking only one witness (and to overcome the problem with a potential infinite number of witnesses), we first generalize it into a predicate that abstracts similar dangerous states (see the calls to `generalizeWitness`). We define the formula $\underline{\hat{m}}(\vec{x}) \triangleq \bigwedge_{i\in 1..n}(x_i \geq m(p_i))$ that is valid for every marking that covers m; in the sense that $m' \models \underline{\hat{m}}$ only when $m' \geqslant m$. By virtue of the monotonicity of the flow function of Petri nets, when $\neg F$ is monotonic and m is a witness, we know that all models of $\underline{\hat{m}}$ are also witnesses. Hence we can improve the method by generating a *minimal inductive clause* (MIC) from $\neg\underline{\hat{m}}$ instead of $\neg\underline{m}$. Another benefit of this choice is that $\underline{\hat{m}}(\vec{x})$ is a conjunction of inequalities of the form $(x_j \geqslant k_i)$, which greatly simplifies the computation of the MIC. When F is anti-monotonic ($\neg F$ is monotonic), we can prove the completeness of the procedure using an adaptation of Dickson's lemma, which states that we cannot find an infinite decreasing chain of witnesses (but the number of possible witness may be extremely large). Hence, when we block it, we learn new clauses from $\neg\underline{\hat{m}}$ that can be propagated to the previous frames.

Before pushing a new clause, we test whether $\underline{\hat{m}}$ 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

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

1 **if** $min < 0$ ***and*** `sat`$(F_0(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge s(\vec{x}'))$ **then**
2 | **raise** $Counterexample$
3 **for** $i \leftarrow \max(1, min + 1)$ ***to*** k **do**
4 | **if** `sat`$(F_i(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge \neg s(\vec{x}) \wedge s(\vec{x}'))$ **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** $\top$ **do**
2 | $(s, n) \leftarrow$ from $states$ minimizing n
3 | **if** $n > k$ **then**
4 | | **return**
5 | **if** $(m \xrightarrow{t} m') \models F_n(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge s(\vec{x}')$ **then**
6 | | $\hat{m} \leftarrow$ `generalizeWitness(`m`)`
7 | | $l \leftarrow$`inductivelyGeneralize(`$\hat{m}, n - 2, k$`)`
8 | | $states \leftarrow states \cup \{(p, l + 1)\}$
9 | **else**
10 | | $l \leftarrow$ `inductivelyGeneralize(`*s, n, k*`)`
11 | | $states \leftarrow states \setminus \{(s, n)\} \cup \{(s, l + 1)\}$

the process. This is the role of functions `pushGeneralization` and `inductivelyGeneralize`.

The goal of `inductivelyGeneralize` is to strengthen the invariants in $(F_i)_{i \leqslant k}$ by learning new clauses and finding the smallest index in $1..k$ that can lead to the dangerous states $\hat{m}$. The goal of `pushGeneralization` is to apply inductive generalization, starting from the earliest possible level.

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

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 $F_i(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge G(\vec{x}')$ are unsatisfiable 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 MIC.

5.3.2. Combination with polyhedral abstraction

Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$ and that G_2 is the E-transform of formula G_1 on N_1. We also assume that G_1 and G_2 are anti-monotonic (meaning $\neg G_1$ and $\neg G_2$ monotonic), in order to ensure the termination of the PDR procedure. (We can prove that $\tilde{E}$ is monotonic for systems E computed with the REDUCE tool when the initial net does not use inhibitor arcs.) To check that formula G_1 is an invariant on N_1 (meaning $\mathrm{AG}\, G_1$ true), it is enough [10] to incrementally build OARS $(F_0, \ldots, F_{k+1})$ on N_1 until $F_i = F_{i+1}$ for some index $i \in 0..k$. In this context, $F_0 = \underline{m_1}$ and $F_{k+1} \Rightarrow G_1$. In a similar way than with our extension of BMC with reductions, a corollary of our *invariant conservation* theorem (Th. 4.2) is that, to check that G_1 is an invariant on N_1, it is enough to build OARS $(F'_0, \ldots, F'_{l+1})$ on N_2 where $F'_0 = \underline{m_2}$ and $F'_{l+1} \Rightarrow G_2$.

Theorem 5.2. (PDR with E-transform)
Assume $(N_1, m_1) \rhd_E (N_2, m_2)$ and that G_2 is the E-transform of G_1, both anti-monotonic formulas. Formula G_1 is an invariant on N_1 if and only if there exists $i \geqslant 0$ such that $F'_i = F'_{i+1}$ in the OARS built from net N_2 and formula G_2.

Proof:
Our proof relies on the property that PDR is sound and complete for finite-state systems, see for instance Th. 1 in [10]: G_1 is an invariant on (N_1, m_1) if and only if there exists $i \geqslant 0$ such that $F_i = F_{i+1}$. Since we deal with monotonic formulas (in this case the feared states $\neg G_1$ and $\neg G_2$), the set of markings satisfying a frame F_i is always upward-closed (if $m \models F_i$ and $m' \geqslant m$ then also $m' \models F_i$), and therefore we can work on the *coverability graph* of the net, instead on its actual marking graph, which is finite [20]. The results follows from our *invariant conservation* theorem (Th. 4.2). □

5.4. Combination of BMC and PDR

In the next section (Sect.6), we report on the results obtained with our implementation of BMC and PDR (with and without reductions), on an independent and comprehensive set of benchmarks.

With PDR, we restrict ourselves to the proof of liveness properties, $\mathrm{EF}\, \phi$ where ϕ is monotonic (or equivalently, invariants $\mathrm{AG}\, \phi$ with ϕ anti-monotonic). In practice, we do not check if ϕ is monotonic using our "semantical" definition. Instead, our implementation uses a syntactical restriction that is a sufficient condition for monotonicity. This is the case, for example, when testing the quasi-liveness of a set of transitions. On the other hand, deadlock is not monotonic. In such cases, we can only rely on the BMC procedure, which may not terminate if the net has no deadlocks. Hence, our best-case scenario is when we check a monotonic property (or if a model for the property exists). In our benchmarks, we find that almost 30% of all the properties are monotonic.

We have plans to improve our PDR procedure to increase the set of properties that can be handled. In particular, we know how to do better when the net is k-bounded (and we know the value of k). We also have several proposals to improve the computation of a good witness, and its MIC, in the general case. We should explore all these ideas in a future work.

6. Experimental results

We have implemented the approach described in Sect. 5 into a new tool, called SMPT (for Satisfiability Modulo P/T Nets). The tool is open-source, under the GPLv3 license, and is freely available on GitHub (`https://github.com/nicolasAmat/SMPT/`). In this section, we report on some experimental results obtained with SMPT (v2) on an extensive benchmark of models and formulas provided by the Model Checking Contest (MCC) [5, 21].

SMPT serves as a front-end to generic SMT solvers, such as z3 [22, 23]. The tool can output sets of constraints using the SMT-LIB format [6] and pipe them to a z3 process through the standard input. We have implemented our tool with the goal to be as interoperable as possible, but we have not conducted experiments with other solvers yet.

SMPT takes as inputs Petri nets defined using the `.net` format of the TINA toolbox and can therefore also accept nets defined using the PNML syntax. For formulas, we accept properties defined with the XML syntax used in the MCC competition. The tool does not compute net reductions directly but relies on the tool REDUCE, that we described at the end of Sect. 3.

The tool REDUCE is distributed with the standard distribution of the TINA toolbox, which includes the most mature versions of our verification tools. We also provide an open source, feature complete version of an equivalent tool, called SHRINK (`https://github.com/Fomys/pnets`), that provide several Rust libraries for manipulating Petri nets and performing structural reductions.

6.1. Benchmarks and distribution of reduction ratios

Our benchmark suite is built from a collection of 102 models used in the MCC competition. Most of the models are parametrized, and therefore there can be several different *instances* for the same model. There are about $1\,000$ different instances of Petri nets whose size vary widely, from 9 to $50\,000$ places, and from 7 to $200\,000$ transitions. Most nets are ordinary (non-zero weights on all arcs are equal to 1), but a significant number of instances (about 150) use weighted arcs. Overall, the collection provides a large number of examples with various structural and behavioral characteristics, covering a large variety of use cases.

Since our approach relies on the use of net reductions, it is natural to wonder if reductions occur in practice. To answer this question, we computed the reduction ratio (r), obtained using REDUCE, as a quotient between the number of places before (p_{init}) and after (p_{red}) reduction: $r = (p_{\text{init}} - p_{\text{red}})/p_{\text{init}}$. We display the results for the whole collection of instances in Fig. 8, sorted in descending order.

A ratio of 100% ($r = 1$) means that the net is *fully reduced*; the resulting net has only one (empty) marking. We see that there is a surprisingly high number of models that are totally reducible with our approach (about 20% of the total number), with approximately half of the instances that can be reduced by a ratio of 30% or more.

For each edition of the MCC, a collection of about 30 random reachability properties are generated for each instance. We evaluated the performance of SMPT using the formulas of the MCC'2020, on a selection of 426 Petri nets taken from instances with a reduction ratio greater than 1%. (To avoid any bias introduced by models with a large number of instances, we selected at most 5 instances with a similar reduction ratio from each model.)

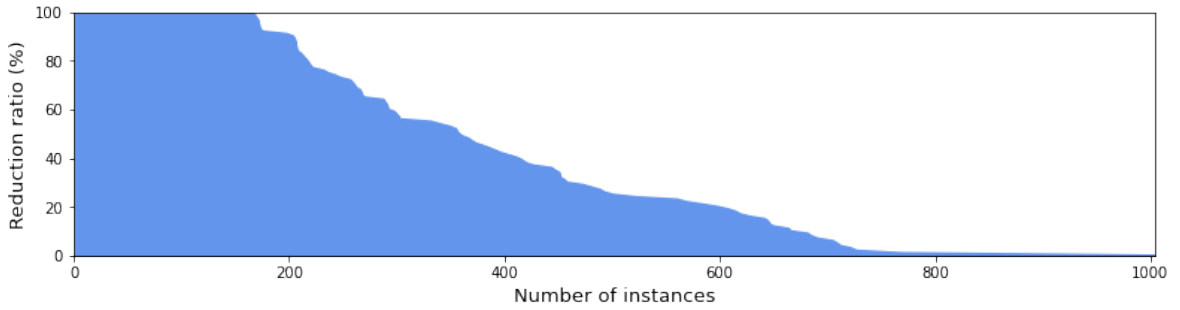


Figure 8: Distribution of reduction ratios over the instances in the MCC

A pair of an instance and a formula is called a *test case*. For each test case, we check the formulas with and without the help of reductions (using both the BMC and PDR methods in parallel) and with a fixed timeout of 120 s. This adds up to a total of 13 265 *test cases* which required the equivalent of 447 hours of CPU time.

6.2. Impact on the number of solvable queries

We report our results in the table of Fig. 9. Out of the almost 13 000 test cases, we were able to compute approximately 7 000 results using reductions and only 3 555 without reductions (so approximately twice more).

We compared our results with the ones provided by an *oracle* [24], which gives the expected answer (as computed by a majority of tools, using different techniques, during the MCC competition). We achieve 100% reliability on the benchmark; meaning we always give the answer predicted by the oracle.

We give the number of computed results for four different categories of test cases: *Full* contains only the fully reducible instances (the best possible case with our approach); while *Low*/*Good*/*High* correspond to instances with a low/moderate/high level of reduction. We chose the limits for these categories in order to obtain samples with comparable sizes. We also have a general category, *All*, for the complete set of benchmarks.

We observe that we are able to compute almost twice as many results when we use reductions than without. This gain is greater on the *High* ($\times 3.1$) than on the *Good* ($\times 1.7$) instances. Nonetheless, the fact that the number of additional queries solved using reductions is still substantial, even for a reduction ratio under 50%, indicates that our approach can benefit from all the reductions we can find in a model (and that our results are not skewed by the large number of fully reducible instances).

In the special case of *fully reducible* nets, checking a query amounts to solving a linear system on the initial marking of the reduced net. There are no iterations. Moreover this is the same system for both the BMC and PDR procedures. For this category, we are able to compute a result for all but one of the queries (that could be computed using a timeout of 180 s). Most of these queries can be solved in less than a few seconds.

When the distinction makes sense, we also report the number of cases solved using BMC/PDR.

Reduction Ratio (r)	# Test Cases	Results (BMC/PDR)			
		With reductions		Without	
All $r \in]0,1]$	13 265	6 986		3 555	(3 261 / 294)
Low $r \in]0,0.25[$	4 586	1 662	(1 532 / 130)	1 350	(1 247 / 103)
Good $r \in [0.25,0.5[$	2 823	1 176	(1 084 / 92)	704	(631 / 73)
High $r \in [0.5,1[$	3 298	1 591	(1 412 / 179)	511	(457 / 54)
Full $r = 1$	2 558	2 557		990	(926 / 64)

Figure 9: Impact of the reduction ratio on the number of solved instances.

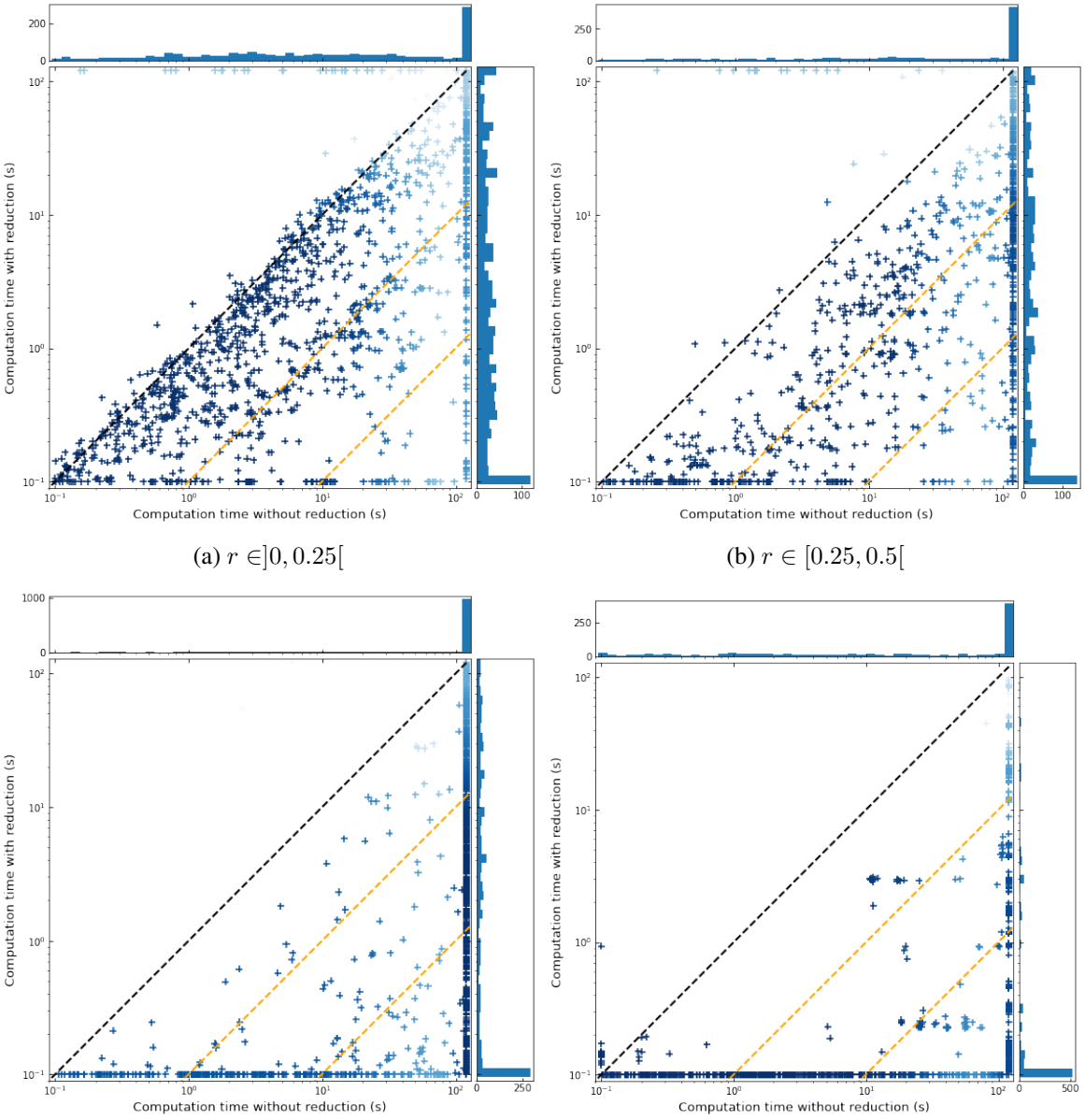
(As said previously, the two procedures coincide in category *Full*, with reductions.) We observe that the contribution of PDR is poor. This can be explained by several factors. First, we restricted our implementation of PDR to monotonic formulas (which represents 30% of all properties). Among these, PDR is useful only when we have an invariant that is true (meaning BMC will certainly not terminate). On the other hand, PDR is able to give answers on the most complex cases. Indeed, it is much more difficult to prove an invariant than to find a counter-example (and we have other means to try and find counter-examples, like simulation for instance). This is why we intend to improve the performances and the "expressiveness" of our PDR implementation. Another factor, already observed in [25], is the existence of a bias in the MCC benchmark: in more than 60% of the cases, the result follows from finding a counter-example (meaning an invariant that is false or a reachability property that is true).

6.3. Impact on computation time

To better understand the impact of reductions on the computation time, we compare the computation time, with or without reductions, for each test case. These results do not take into account the time spent for reducing each instance. This time is negligible when compared to each test, usually in the order of $1\,\mathrm{s}$. Also, we only need to reduce the net once when checking the 30 properties for the same instance.

We display our results in Fig. 10, where we give four scatter plots comparing the computation time "with" (y-axis) and "without" reductions (x-axis), for the *Low*, *Good*, *High* and *Full* categories of instances. Each chart uses a logarithmic scale. We also display a histogram, for each axis on the charts, that gives the density of points for a given duration. To avoid overplotting, we removed all the "trivial" properties (the bottom left part of the chart), that can be computed with and without reduction in less than $10\,\mathrm{ms}$. These "trivial" queries (507 in total) correspond to instances with a small state space or to situations where a counter-example can be found very quickly.

We observe that almost all the data points are below the diagonal, meaning reductions accelerate the computation, with many test cases exhibiting speed-ups larger than $\times 100$. We have added



(a) $r \in]0, 0.25[$

(b) $r \in [0.25, 0.5[$

(c) $r \in [0.5, 1[$

(d) $r = 1$

Figure 10: Comparing computation time, "with" (y-axis) and "without" (x-axis) reductions for categories *Low* (a), *Good* (b), *High* (c) and *Full* (d).

two light-coloured, dashed lines to materialize data points with speed-ups larger than $\times 10$ and $\times 100$ respectively.

On our $13\,265$ test cases, we timeout with reductions but compute a result without on only 51 cases (0.4%). These exceptions can be explained by border cases where the order in which transitions are processed has a sizeable impact.

Another interesting point is the ratio of properties that can be computed only using reductions. This is best viewed when looking at the histogram values. A vast majority of the points in the charts are either on the right border (computation without reductions timeout) or on the x-axis (they can be computed in less than $10\,\mathrm{ms}$ using reductions).

7. Related work and conclusion

We propose a new method to combine structural reductions with SMT solving in order to check invariants on arbitrary Petri nets. While this idea is not original, the framework we developed is new. Our main innovation resides in the use of a principled approach, where we can trace back reachable markings (between an initial net and its residual) by means of a conjunction of linear equalities (the formula $\tilde{E}$). Basically, we show that we can adapt a SMT-based procedure for checking a property on a net (that relies on computing a family of formulas of the form $(\phi_i)_{i\in I}$) into a procedure that relies on a reduced version of the net and formulas of the form $(\phi_i \wedge \tilde{E})_{i\in J}$.

As a proof of concept, we apply our approach to two basic implementations of the BMC and PDR procedures. Our empirical evaluation shows promising results. For example, we observe that we are able to compute twice as many results using reductions than without. We believe that our approach can be adapted to more decision procedures and could easily accommodate various types of optimizations.

7.1. Related work

Our main theoretical results (the conservation theorems of Sect. 4) can be interpreted as examples of *reduction theorems* [26, 27], that allow to deduce properties of an initial model (N) from properties of a simpler, coarser-grained version (N^R). While these works are related, they mainly focus on reductions where one can group a sequence of transitions into a single, atomic action. Hence, in our context, they correspond to a restricted class of reductions, similar to a subset of the agglomeration rules used in [3].

We can also mention approaches where the system is simplified with respect to a given property, for instance by eliminating parts that cannot contribute to its truth value, like with the slicing or *Cone of Influence* abstractions [28] used in some model checkers. Finding such "parts" (places and transitions) in a Petri net is not always easy, especially when the formula involves many places. This is not a problem with our approach, since we can always abstract away a place, as long as its effect is preserved in the E-transform formula.

In [29], the author uses net invariants to compress the representation of markings. This approach is based on the fact that place invariants provide linear constraints between the markings of several places; like in our use of redundancy rules. But the goal is to reduce the memory footprint when computing the explicit state space, while verification is still performed on "uncompressed" markings.

On the contrary, our approach can be used with symbolic methods–working on a reduced version of the net–and can use more general rules. For instance, it cannot benefit from rules that agglomerate places.

In practice, we derive polyhedral abstractions using *structural reductions*, a concept introduced by Berthelot in [1]. In our work, we are interested in reductions that preserve the reachable states. This is in contrast with most works about reductions, where more powerful transformations can be applied when we focus on specific properties, such as the absence of deadlocks. Several tools use reductions for checking reachability properties. TAPAAL [30], for instance, is an explicit-state model checker that combines Partial-Order Reduction techniques and structural reductions and can check properties on Petri nets with weighted arcs and inhibitor arcs.

A more relevant example is ITS Tools [25, 31], which combines several techniques, including structural reductions and the use of SAT and SMT solvers. This tool relies on efficient methods for finding counter-examples—with the goal to invalidate an invariant—based on the collaboration between pseudo-random exploration techniques; hints computed by an SMT engine; and reductions that may simplify atoms in the property or places and transitions in the net. It also describes a semi-decision procedure, based on an over-approximation of the state space, that may detect when an invariant holds (by ruling out infeasible behaviours). This leads to a very efficient tool, able to compute a result for most of the queries in our benchmark, when we solve only 52% of our test cases. Nonetheless, we are able to solve 46 queries with SMPT (with a timeout of $120\,\mathrm{s}$) that are not in the oracle results collected from ITS Tools [24]; which means that no other tool was able to compute a result on these queries.

It has to be kept in mind, though, that our goal is to study the impact of polyhedral abstraction, in isolation from other techniques. However, the methods described in [25] provide many ideas for improving our approach, such as: using linear arithmetic over reals—which is more tractable than integer arithmetic—to over-approximate the state space of a net; adding extra constraints to strengthen invariants (for instance using the state equation or constraints derived from traps); dividing up a formula into smaller sub-parts, and checking them incrementally or separately; ... But the main lesson to be learned is that there is a need for a complete decision procedure devoted to the proof of satisfiable invariants, which further our interest in improving our implementation of PDR.

A byproduct of our work is to provide a partial implementation of PDR that is correct and complete when the property is monotonic (see Sect. 4), even in the case of nets with an infinite state space. Our current solution can be understood as a restriction to the case of "coverability properties", which seems to be the current state-of-the-art with Petri nets; see for example [32] and [33], or the extension of PDR to "well-structured transition systems" [34]. The reachability problem for Petri nets or, equivalently, for Vector Addition Systems with States (VASS) is decidable [35]. Even if this result is based on a constructive proof, and its "construction" streamlined over time [36], 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 [37]. 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 can mention the tool FASTFORWARD [38], that explicitly targets the case of unbounded nets. We can also mention the works on inductive procedures for infinite-state and/or parametrized systems, such as the verification

methods used in Cubicle [39]; see also [18, 40].

7.2. Follow up work

We propose a new method that adapts our approach—initially developed for model checking with decision diagrams [2, 3]—for use with SMT solvers.

We have continued working with our polyhedral abstraction since our initial publication in [12]. In particular, we tried applying our approach to the verification of properties more complex than reachability, like with our recent work on the *concurrent places* problem [41]. The problem, in this case, is to enumerate all pairs of places that can be marked together, for some reachable states. In this work we defined a new data-structure that precisely captures the structure of reduction constraints, what we call the *Token Flow Graph* (TFG), and we used the TFGs to accelerate the computation of the concurrency relation

We also continued improving our adaptation of PDR, which is the most promising part of our work and raises several interesting theoretical problems. In this context, we recently proposed two new adaptions of PDR, to deal with non-monotonic formulas [19]. But there is still a lot of work to be done, like for instance concerning the completeness of our new approach and/or its limits.

7.3. Future work

There is still ample room for improving our tool. We already mentioned some ideas for enhancements that we could borrow from ITS Tools, but we also plan to specialize our verification procedures in some specific cases, for example when we know that a net is 1-safe. A first step should be to compare our performances with other tools in more details. This is what motivates our participation to the next edition of the MCC, with SMPT alone in the reachability examinations, even though it is common knowledge that winning tools need to combine several different techniques.

On a more theoretical side, we also identified a need to develop an automated (or a semi-automatic) method to prove the correctness of new reduction rules.

References

[1] Berthelot G. Transformations and Decompositions of Nets. In: Petri Nets: Central Models and their Properties, volume 254 of *LNCS*. Springer, 1987 doi:10.1007/978-3-540-47919-2_13.

[2] Berthomieu B, Le Botlan D, Dal Zilio S. Petri net reductions for counting markings. In: Model Checking Software (SPIN), volume 10869 of *LNCS*. Springer, 2018 doi:10.1007/978-3-319-94111-0_4.

[3] Berthomieu B, Le Botlan D, Dal Zilio S. Counting Petri net markings from reduction equations. *International Journal on Software Tools for Technology Transfer*, 2019. **22**. doi:10.1007/s10009-019-00519-1.

[4] Thierry-Mieg Y, Poitrenaud D, Hamez A, Kordon F. Hierarchical set decision diagrams and regular models. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 5505 of *LNCS*. Springer, 2009 doi:10.1007/978-3-642-00768-2_1.

[5] Amparore E, Berthomieu B, Ciardo G, Dal Zilio S, Gallà F, Hillah LM, Hulin-Hubard F, Jensen PG, 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), volume 11429 of *LNCS*. Springer, 2019 doi:10.1007/978-3-662-58381-4_9.

[6] Barrett C, Fontaine P, Tinelli C. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `http://www.smt-lib.org/`.

[7] Besson F, Jensen T, Talpin JP. Polyhedral analysis for synchronous languages. In: Static Analysis Symposium (SAS), volume 1694 of *LNCS*. Springer, 1999 doi:10.1007/3-540-48294-6_4.

[8] Feautrier P. Automatic parallelization in the polytope model. In: The Data Parallel Programming Model, volume 1132 of *LNCS*. Springer, 1996. doi:10.1007/3-540-61736-1_44.

[9] 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), volume 1579 of *LNCS*. Springer, 1999 doi:10.1007/3-540-49059-0_14.

[10] Bradley AR. SAT-Based Model Checking without Unrolling. In: Verification, Model Checking, and Abstract Interpretation (VMCAI), volume 6538 of *LNCS*. Springer, 2011. doi:10.1007/978-3-642-18275-4_7.

[11] Bradley AR. Understanding IC3. In: Theory and Applications of Satisfiability Testing (SAT), volume 7317 of *LNCS*. Springer, 2012. doi:10.1007/978-3-642-31612-8_1.

[12] Amat N, Berthomieu B, Dal Zilio S. On the Combination of Polyhedral Abstraction and SMT-Based Model Checking for Petri Nets. In: Application and Theory of Petri Nets and Concurrency (Petri Nets), volume 12734 of *LNCS*. Springer, 2021 doi:10.1007/978-3-030-76983-3_9.

[13] Lloret JC, Azéma P, Vernadat F. Compositional design and verification of communication protocols, using labelled Petri nets. In: Computer-Aided Verification (CAV), volume 531 of *LNCS*. Springer, Berlin, Heidelberg, 1991 doi:10.1007/BFb0023723.

[14] Hujsa T, Berthomieu B, Dal Zilio S, Le Botlan D. Checking marking reachability with the state equation in Petri net subclasses. *arXiv preprint arXiv:2006.05600*, 2020.

[15] Clarke E, Biere A, Raimi R, Zhu Y. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 2001. **19**. doi:10.1023/A:1011276507260.

[16] Heljanko K. Bounded Reachability Checking with Process Semantics. In: International Conference on Concurrency Theory (CONCUR), volume 2154 of *LNCS*. Springer, 2001 doi:10.1007/3-540-44685-0_15.

[17] Armando A, Mantovani J, Platania L. Bounded Model Checking of Software Using SMT Solvers Instead of SAT Solvers. In: Model Checking Software (SPIN), volume 3925 of *LNCS*. Springer, 2006 doi:10.1007/11691617_9.

[18] Cimatti A, Griggio A, Mover S, Tonetta S. Infinite-state invariant checking with IC3 and predicate abstraction. *Formal Methods in System Design*, 2016. **49**. doi:10.1007/s10703-016-0257-4.

[19] Amat N, Dal Zilio S, Hujsa T. Property Directed Reachability for Generalized Petri Nets. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 13243 of *LNCS*. Springer, 2022 doi:10.1007/978-3-030-99524-9_28.

[20] Finkel A. The minimal coverability graph for Petri nets. In: International Conference on Application and Theory of Petri Nets (ICATPN). Springer, 1991 doi:10.1007/3-540-56689-9_45.

[21] Hillah L, Kordon F. Petri Nets Repository: A Tool to Benchmark and Debug Petri Net Tools. In: Application and Theory of Petri Nets and Concurrency (Petri Nets), volume 10258 of *LNCS*. Springer, 2017 doi:10.1007/978-3-319-57861-3_9.

[22] de Moura L, Bjørner N. Z3: An Efficient SMT Solver. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 4963 of *LNCS*. Springer, 2008 doi:10.1007/978-3-540-78800-3_24.

[23] Bjørner N. The Z3 Theorem Prover. `https://github.com/Z3Prover/z3/`, 2020.

[24] Thierry-Mieg Y. Oracle for the MCC 2020 edition. Available at `https://github.com/yanntm/pnmcc-models-2020`, 2020.

[25] Thierry-Mieg Y. Structural Reductions Revisited. In: Application and Theory of Petri Nets and Concurrency (Petri Nets), volume 12152 of *LNCS*. Springer, 2020 doi:10.1007/978-3-030-51831-8_15.

[26] Lipton RJ. Reduction: a method of proving properties of parallel programs. *Communications of the ACM*, 1975. **18**. doi:10.1145/361227.361234.

[27] Cohen E, Lamport L. Reduction in TLA. In: Concurrency Theory (CONCUR), volume 1466 of *LNCS*. Springer, 1998 doi:10.1007/BFb0055631.

[28] Clarke EM, Grumberg O, Peled D. Model Checking. MIT Press, 1999.

[29] Schmidt K. Using Petri Net Invariants in State Space Construction. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 2619 of *LNCS*. Springer, 2003 .

[30] Bønneland FM, Dyhr J, Jensen PG, Johannsen M, Srba J. Stubborn versus structural reductions for Petri nets. *Journal of Logical and Algebraic Methods in Programming*, 2019. **102**. doi:10.1016/j.jlamp.2018.09.002.

[31] Thierry-Mieg Y. Symbolic and Structural Model-Checking. *Fundam. Informaticae*, 2021. **183**(3-4). doi:10.3233/FI-2021-2090.

[32] Esparza J, Ledesma-Garza R, Majumdar R, Meyer P, Niksic F. An SMT-Based Approach to Coverability Analysis. In: Computer Aided Verification (CAV), volume 8559 of *LNCS*. 2014 doi:10.1007/978-3-319-08867-9_40.

[33] Kang J, Bai Y, Jiao L. Abstraction-Based Incremental Inductive Coverability for Petri Nets. In: Applications and Theory of Petri Nets and Concurrency (Petri Nets), volume 12734 of *LNCS*. Springer, 2021 doi:10.1007/978-3-030-76983-3_19.

[34] Kloos J, Majumdar R, Niksic F, Piskac R. Incremental, Inductive Coverability. In: Computer Aided Verification (CAV), volume 8044 of *LNCS*. Springer, 2013 doi:10.1007/978-3-642-39799-8_10.

[35] Kosaraju SR. Decidability of Reachability in Vector Addition Systems (Preliminary Version). In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC. ACM, 1982 doi:10.1145/800070.802201.

[36] Leroux J. The general vector addition system reachability problem by Presburger inductive invariants. In: Logic in Computer Science (LICS). IEEE, 2009 doi:10.1109/LICS.2009.10.

[37] Dixon A, Lazić R. KReach: A Tool for Reachability in Petri Nets. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 12078 of *LNCS*. Springer, 2020 doi:10.1007/978-3-030-45190-5_22.

[38] Blondin M, Haase C, Offtermatt P. Directed Reachability for Infinite-State Systems. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 12652 of *LNCS*. Springer, 2021 doi:10.1007/978-3-030-72013-1_1.

[39] Conchon S, Goel A, Krstic S, Mebsout A, Zaïdi F. Cubicle: A Parallel SMT-Based Model Checker for Parameterized Systems. In: Computer Aided Verification (CAV), volume 7358 of *LNCS*. Springer, 2012 doi:10.1007/978-3-642-31424-7_55.

[40] Gurfinkel A, Shoham S, Meshman Y. SMT-based verification of parameterized systems. In: International Symposium on Foundations of Software Engineering. ACM, 2016 doi:10.1145/2950290.2950330.

[41] Amat N, Dal Zilio S, Le Botlan D. Accelerating the Computation of Dead and Concurrent Places Using Reductions. In: Model Checking Software (SPIN), volume 12864 of *LNCS*. Springer, 2021 doi:10.1007/978-3-030-84629-9_3.