On the Combination of Polyhedral Abstraction and SMT-based Model Checking for Petri nets

Nicolas Amat, Bernard Berthomieu, Silvano Dal Zilio

▶ **To cite this version:**

Nicolas Amat, Bernard Berthomieu, Silvano Dal Zilio. On the Combination of Polyhedral Abstraction and SMT-based Model Checking for Petri nets. 42rd International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2021), Jun 2021, Paris (virtual), France. 10.1007/978-3-030-76983-3_9 . hal-03202111

HAL Id: hal-03202111
https://laas.hal.science/hal-03202111

Submitted on 19 Apr 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On the Combination of Polyhedral Abstraction and SMT-based Model Checking for Petri nets

Nicolas Amat[1], Bernard Berthomieu[1], and Silvano Dal Zilio[1]

[1]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Abstract

We define a method for taking advantage of net reductions in combination with a SMT-based model checker. We prove the correctness of this method using a new notion of equivalence between nets that we call polyhedral abstraction. 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 during the 2020 edition of the Model Checking Contest. Our experimental results show that our approach works well, even when we only have a moderate amount of reductions.

Keywords— Model Checking; Reachability problems; SMT solving; Abstraction techniques.

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 from current 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 [5] and known as *structural reduction*.

We recently proposed a new abstraction technique based on reductions [6, 7]. 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 simpler than N); and E is a system of linear equations. The idea is to preserve enough information in E so that we can rebuild the reachable markings of N knowing only the one 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 (Sect. 3) and show how to use it for checking safety and invariant properties on nets (Sect. 4).

We have previously applied this technique in a symbolic model checker, called Tedd, that uses Set Decision Diagrams [33] 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) [2], where Tedd won the competition for 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 to use solvers for the theory of Quantifier-Free formulas on Linear Integer Arithmetic, what is known as QF-LIA in SMT-LIB [4].

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 [8, 21]. (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 linear equations.) 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) [13, 14]. 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. Our experimental results show that our approach works well, even when we only have a moderate amount of reductions. For instance, we observe that we are able to compute twice as many results using reductions than without, with a reliability of 100%.

1.1 Outline and Contributions

This paper summarises the key ideas and results of [1], to which we refer the reader for full details. The paper is organized as follows. In Sect. 3, we define our notion of polyhedral abstraction and prove several of its properties. This definition relies on a presentation of Petri net semantics that emphasizes the

relationship with the QF-LIA theory (Section 2). We use these results in Sect. 4 to describe our adaptation of general SMT-based algorithms with reductions and prove their correctness. In Sect. 5 we describe our adaptation of BMC and PDR with reductions. Before concluding, we report on experimental results on an extensive collection of nets and queries.

2 Petri Nets and Linear Arithmetic Constraints

A *Petri net* N is a tuple $(P, T, \mathbf{pre}, \mathbf{post})$ where $P = \{p_1, \ldots, p_n\}$ is a finite set of places, $T = \{t_1, \ldots, t_k\}$ is a finite set of transitions (disjoint from P), and $\mathbf{pre} : T \to (P \to \mathbb{N})$ and $\mathbf{post} : T \to (P \to \mathbb{N})$ are the pre- and post-condition functions (also called the flow functions of N). A state m of a net, also called a *marking*, is a mapping $m : P \to \mathbb{N}$ 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. 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 $t \in T$.

A transition $t \in T$ is *enabled* at marking $m \in \mathbb{N}^P$ when $m(p) \geq \mathbf{pre}(t, p)$ for all places p in P. (We can also simply write $m \geq \mathbf{pre}(t)$, where $\geq$ 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) $m' = m - \mathbf{pre}(t) + \mathbf{post}(t)$. By extension, we say that a *firing sequence* $\sigma = t_1 \ldots t_n \in T^*$ can be fired from m, denoted $m \overset{\sigma}{\Rightarrow} m'$, 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 $i < n$.

We denote $R(N, m)$ the set of markings reachable from m in N. 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.

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 $T^\star \to \Sigma^\star$ such that $l(\epsilon) = \epsilon$, $l(\tau) = \epsilon$ and $l(\sigma\, t) = l(\sigma)\, l(t)$. Given a sequence of labels σ in $\Sigma^\star$, we write $(N, m) \overset{\sigma}{\Rightarrow} (N, m')$ when there is a firing sequence ϱ in $T^\star$ such that $(N, m) \overset{\varrho}{\Rightarrow} (N, m')$ and $\sigma = l(\varrho)$. We say in this case that σ is an *observable sequence* of the marked net (N, m).

We use the standard graphical notation for nets, where places are depicted as circles and transitions as squares. With the net displayed in Fig. 1 (left), the initial marking is $m_1 \triangleq p_0{*}5\ p_6{*}4$ (only 5 and 4 tokens in places p_0 and p_6). We have $m_1 \overset{\sigma}{\Rightarrow} m_1'$ with $\sigma \triangleq t_0\, t_0\, t_1\, t_1\, t_2\, t_3\, t_4$ and $m_1' \triangleq p_0{*}3\ p_2{*}1\ p_3{*}1\ p_6{*}3$; and therefore $m_1 \overset{\mathbf{a\,a\,b\,c}}{\Longrightarrow} m_1'$.

We can define many properties on the markings of a net N using Boolean combinations of linear constraints with integer variables (what is called the QF-LIA theory in SMT-LIB). 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

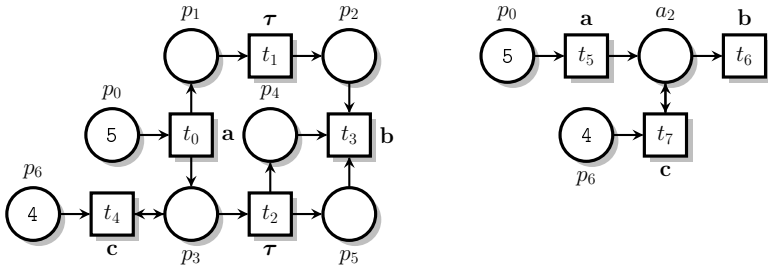


Figure 1: An example of Petri net, M_1 (left), and one of its polyhedral abstraction, M_2 (right), with $E_M \triangleq (p_5 = p_4) \wedge (a_1 = p_1 + p_2) \wedge (a_2 = p_3 + p_4) \wedge (a_1 = a_2)$.

there must be k tokens in place p_i. Formula $\underline{m}$ is obviously a conjunction of literals, what is called a *cube* in [13].

$$\underline{m}(x_1, \ldots, x_n) \triangleq (x_1 = m(p_1)) \wedge \cdots \wedge (x_k = m(p_k)) \tag{1}$$

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 should 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 2.1 (Models 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 $m \models \mathrm{SAFE}_1(\vec{x})$, where $\mathrm{SAFE}_k(\vec{x}) \triangleq \bigwedge_{i \in 1..n}(x_i \leq k)$.

Likewise, the property that transition t is enabled corresponds to formula $\mathrm{ENBL}_t(\vec{x}) \triangleq \bigwedge_{i \in 1..n}(x_i \geq \mathbf{pre}(t, p_i))$, in the sense that t is enabled at m when $m \models \mathrm{ENBL}_t(\vec{x})$. 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. 4, 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 property that we want to check include: checking if some transition t is enabled (commonly known as *quasi-liveness*); checking if there is a deadlock; checking whether some invariant between place markings is true; ...

Definition 2.2 (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: AG ϕ (true only when ϕ is an invariant), and EF ϕ (true when ϕ is reachable), where ϕ is a Boolean combination of atomic properties (it has no modalities).

3 Polyhedral Abstraction and E-Equivalence

We define a new notion, called *E-abstraction equivalence*, that is used to state a correspondence between the set of reachable markings of two Petri nets "modulo" some system of linear equations, E. Basically, we have that (N_1, m_1) is E-equivalent to (N_2, m_2) when, for every sequence $m_2 \stackrel{\sigma_2}{\Longrightarrow} m_2'$ in N_2, there must exist a sequence $m_1 \stackrel{\sigma_1}{\Longrightarrow} m_1'$ in N_1 such that $E \wedge \underline{m_1'} \wedge \underline{m_2'}$ is satisfiable (and reciprocally). Therefore, knowing E, we can compute the reachable markings of N_1 from those of N_2, and vice versa. We also ask for the observable sequences, σ_1 and σ_2 in this case, to be equal. As a result, we will prove that our equivalence is also a congruence.

We can illustrate these notions using the two nets M_1, M_2 in Fig. 1, we have that $m_1' \triangleq p_0{*}3\ p_2{*}1\ p_3{*}1\ p_6{*}3$ is reachable in M_1 and $E_M \wedge \underline{m_1'}$ entails $\underline{m_1'} \wedge (a_1 = 1) \wedge (a_2 = 1)$, which means that 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 corresponds 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 observable sequence $\mathbf{a}\,\mathbf{a}\,\mathbf{b}\,\mathbf{c}$. 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 constraints between linear combinations of integer variables (the marking of places). This is why we should often use the term *reduction equations* 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 equal marking 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 $(P_1 \cup P_2)$ such that $(m_1 \uplus m_2)(p) = m_1(p)$ if $p \in P_1$ and $(m_1 \uplus m_2)(p) = m_2(p)$ otherwise. Hence, with our conventions, $\underline{m_1 \uplus m_2} \Leftrightarrow \underline{m_1} \wedge \underline{m_2}$.

In the following we ask that condition E be *solvable for* N_1, N_2, meaning that for all reachable marking m_1 in N_1 there must exist at least one marking m_2 of N_2, compatible with m_1, such that $m_1 \uplus m_2 \models E$ (by 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 equations 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.

Definition 3.1 (E-abstraction equivalence)**.** *Assume N_1, N_2 are two Petri nets with respective sets of places P_1, P_2 and labeling functions l_1, l_2, over the same alphabet Σ. 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 observation sequences $\sigma \in \Sigma^\star$ such that $(N_1, m_1) \stackrel{\sigma}{\Rightarrow} (N_1, m_1')$ then there is at least one marking $m_2' \in R(N_2, m_2)$ such that $m_1' \uplus m_2' \models E$, and for all markings m_2' over P_2, we have that $m_1' \uplus m_2' \models E$ implies $(N_2, m_2) \stackrel{\sigma}{\Rightarrow} (N_2, m_2')$.*

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 sequences starting from the initial marking of N_1. Hence the relation is usually not true on every pair of matching markings; it is not a bisimulation.

By definition, relation $\rhd_E$ is symmetric. We deliberately use a "comparison symbol" for our equivalence, $\rhd$, in order to stress the fact that N_2 should be 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 3.1 (Bounded Model 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 σ_1 in N_1 such that $(N_1, m_1) \stackrel{\sigma_1}{\Rightarrow} (N_1, m_1')$. 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 $(N_2, m_2) \stackrel{\sigma_2}{\Rightarrow} (N_2, m_2')$ (for some firing sequence σ_2). Therefore we have m_2' reachable in N_2 such that $m_1' \uplus m_2' \models E$. □

Lemma 3.1 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 equations. (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 3.2 (Invariance Checking)**.** *Assume $(N_1, m_1) \rhd_E (N_2, m_2)$. Then for all pairs of markings m_1', m_2' over 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 σ_2 such that $(N_2, m_2) \stackrel{\sigma_2}{\Longrightarrow} (N_2, m'_2)$. By condition (A2), since $m'_1 \uplus m'_2 \models E$, there must be a firing sequence in N_1, say σ_1, such that $(N_1, m_1) \stackrel{\sigma_1}{\Longrightarrow} (N_1, m'_1)$. Hence $m'_1 \in R(N_1, m_1)$. □

Using Lemma 3.2, 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.

Next we prove that polyhedral abstractions are closed by synchronous composition, relabeling, and chaining. Before defining these operations, 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 equations.

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.

In this section we rely on the classical synchronous product operation between labeled Petri nets [29]. Let N_1 and N_2 be two labeled Petri nets with respective sets of places P_1, P_2 and with 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 denote $N_1 \| N_2$ the *synchronous product* between N_1 and N_2. 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 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 $(N_1 \| N_2, m_1 \| m_2)$.

Another standard operation on labeled Petri net 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.

Theorem 3.3 (E-equivalence is a congruence)**.** *Assume we have two compatible equivalence statements* $(N_1, m_1) \rhd_E (N_2, m_2)$ *and* $(N_2, m_2) \rhd_{E'} (N_3, m_3)$*, and that* M *is compatible with respect to these equivalences, then:*

- $(N_1, m_1) \| (M, m) \rhd_E (N_2, m_2) \| (M, m)$.
- $(N_1, m_1) \rhd_{E,E'} (N_3, m_3)$.
- $(N_1[a/b], m_1) \rhd_E (N_2[a/b], m_2)$ *and* $(N_1[a/\tau], m_1) \rhd_E (N_2[a/\tau], m_2)$.

Proof (sketch). The result for the first composition law derives from the fact that we can always compute a unique pair of firing sequences for N_1, M from a firing sequence of $N_1 \| M$. The proof for the other laws are similar, see [1]. □

The composition laws stated in Th. 3.3 are useful to build larger equivalences from simpler axioms (reductions rules). We show some examples of reductions in the next paragraph and how they occur in the example of Fig. 1.

3.3 Deriving E-Equivalences using Reductions

We can compute net reductions by reusing a tool, called Reduce, that was developed in our previous work [7]. The tool takes a marked Petri net as input and returns a reduced net and a sequence of linear equations. 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 example of such rule in Fig. 2 (above), which states that we can fuse places inside a "deterministic sequence" of transitions. This is one of the many *agglomeration* rules defined in [7] and also one of the original rules found in [5].

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 equations E_i, is such that $(M_i, m_i) \rhd_{E_i} (M_{i+1}, m_{i+1})$. Therefore, by Th. 3.3, the results computed by Reduce always translate into valid polyhedral abstractions.

We can look at our running example to explain the inner working of Reduce. It is always safe to remove a *redundant place*, e.g. a place with the same **pre** and **post** relations than another one. This is the case with places p_4, p_5 (see Fig. 2). Redundant places can sometimes be found by looking at the structure of the net, but our tool can also find more elaborate occurrences of redundant places by solving an integer linear programming problem [30].

After the removal of p_5, we are left with a residual net similar to the one in the second equivalence of Fig. 2. In this case, we can use our agglomeration rules to simplify places p_1 and p_3. Similar situations, where we can aggregate several places together, can be found by searching patterns in the net. After this step (introducing two new places a_1 and a_2), we find a new opportunity to reduce a pair of redundant places, (a_1, a_2). Besides these two main kinds of reduction rules (redundancy and agglomeration), the Reduce tool can also identify other opportunities for reductions. 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 [25].

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%.

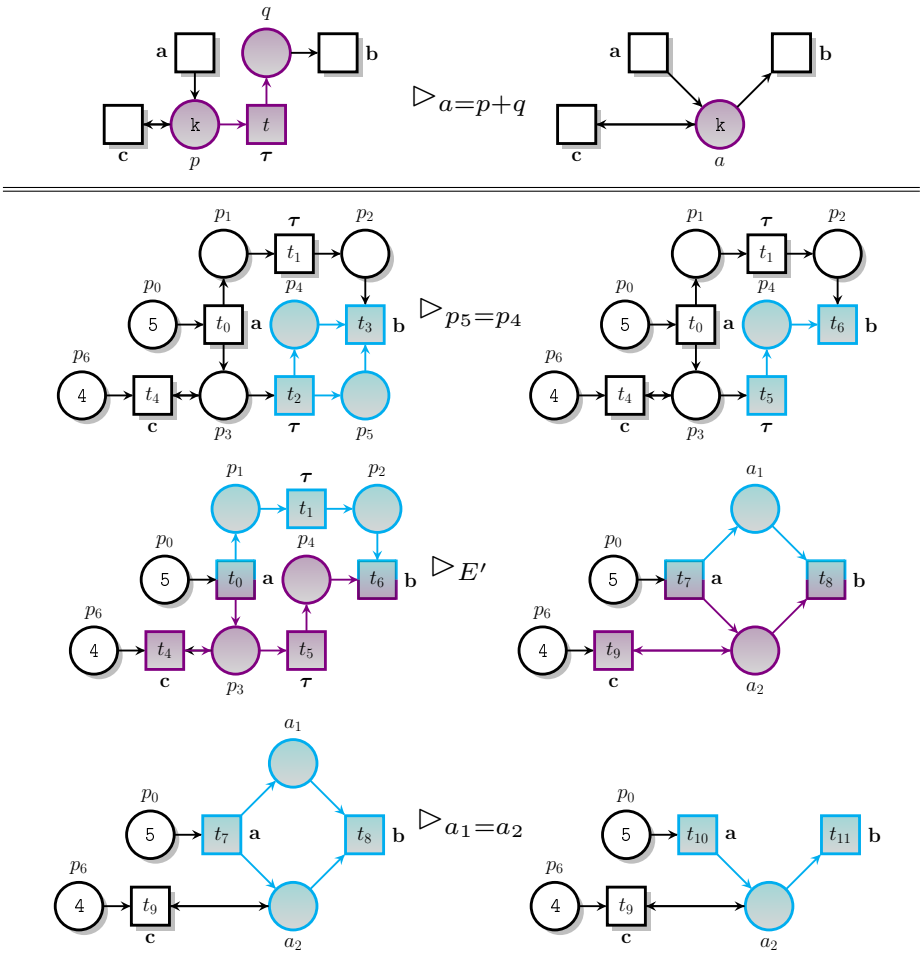


Figure 2: Example of basic reduction rule for agglomerating places (above), and sequence of three reductions (below) leading from the net M_1 to M_2 from Fig. 1, with $E' \triangleq (a_1 = p_1 + p_2) \wedge (a_2 = p_3 + p_4)$.

4 SMT-based Model Checking Using Abstractions

We introduce a general method for combining polyhedral abstraction 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.)

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

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 studied property on the initial net does not contain any new variable introduced during the reduction.

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

Theorem 4.1 (Invariant Conservation)**.** *Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$ and that $F_2(\vec{y})$ is the E-transform of formula F_1 on N_1. Then F_1 is an invariant on N_1 if and only if $F_2(\vec{p_2})$ 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 $\underline{m_1'} \wedge \underline{m_2'} \wedge \tilde{E}(\vec{p_1},\vec{p_2}) \wedge F_1(\vec{p_1})$ 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 4.2 (Reachability Conservation)**.** *Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$ and that $F_2(\vec{y})$ is the E-transform of formula F_1 on N_1. Then formula F_1 is reachable in N_1 if and only if $F_2(\vec{p_2})$ is reachable in N_2.*

5 BMC and PDR Implementation

We developed a prototype model checker that takes advantage of net reductions. The tool offers two main analysis methods that have been developed for generalized Petri nets. (No specific optimizations are applied when we know the net is safe.) These options correspond to the implementation of the BMC and PDR methods, that we sketch below.

5.1 Bounded Model Checking (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 [16]. More recently, this approach was extended to more expressive models, and richer theories, using SMT solvers [3].

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 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 *UNSAT*, 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 *SAT* (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 $\phi_0 \subsetneq \phi_1 \subsetneq \dots$ 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 $P = \{p_1, \dots, p_n\}$ and transitions $T = \{t_1, \dots, t_k\}$. 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 $\psi(x_1, \dots, x_n, x'_1, \dots, x'_n)$.

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, an operator Δ_t that describes the evolution of a marking after transition t fires, see (3) below. It can be used to define another helper formula, $t(\vec{x}, \vec{x}')$, such that $(\underline{m}(\vec{x}) \wedge t(\vec{x}, \vec{x}') \wedge \underline{m'}(\vec{x}'))$ entails that $m \xrightarrow{t} m'$, when t is enabled at m, or $m = m'$ otherwise. With all these notations, we can define $\mathrm{T}(\vec{x}, \vec{x}')$ as the disjunction of all the transition formulas $t(\vec{x}, \vec{x}')$. By construction, formula $\mathrm{T}(m, m') \triangleq \underline{m}(\vec{x}) \wedge \mathrm{T}(\vec{x}, \vec{x}') \wedge \underline{m'}(\vec{x}')$ is true when $m \to m'$, or when $m = m'$.

$$\Delta_t(\vec{x}, \vec{x}') \triangleq \textstyle\bigwedge_{i\in 1..n}(x'_i = x_i + \mathbf{post}(t, p_i) - \mathbf{pre}(t, p_i)) \tag{3}$$

$$\mathrm{EQ}(\vec{x}, \vec{x}') \triangleq \textstyle\bigwedge_{i\in 1..n} x_i = x'_i \tag{4}$$

$$t(\vec{x}, \vec{x}') \triangleq (\mathrm{ENBL}_t(\vec{x}) \Rightarrow \Delta_t(\vec{x}, \vec{x}')) \wedge (\neg\mathrm{ENBL}_t(\vec{x}) \Rightarrow \mathrm{EQ}(\vec{x}, \vec{x}')) \tag{5}$$

$$\mathrm{T}(\vec{x}, \vec{x}') \triangleq \mathrm{EQ}(\vec{x}, \vec{x}') \vee \textstyle\bigvee_{t\in T} (\mathrm{ENBL}_t(\vec{x}) \wedge \Delta_t(\vec{x}, \vec{x}')) \tag{6}$$

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, $(\vec{x}_i)_{i\in\mathbb{N}}$.

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

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 $i \geq 0$ 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 optimizations 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.

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 $\vec{y}$. The following property states that, to find a model of F in the reachable markings of N_1, it is enough to find a model for its E-transform in N_2.

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

Proof (sketch). We start by proving that F_1 reachable in N_1 is equivalent to $F_1 \wedge \phi_i(N_1, m_1)$ satisfiable for some $i \geq 0$. The proof is by induction on the value of i and use the fact that $\mathrm{T}_1(m, m')$ entails $m \Rightarrow m'$ in N_1. As a result, we can prove the existence of a firing sequence $m_1 \stackrel{\sigma}{\Rightarrow} m'_1$, of length at most i, such that $m'_1 \models F_1$. The result follows by our *conservation of reachability* property (Th. 4.2), F_1 reachable in N_1 means F_2 reachable in N_2. Therefore F_1 is reachable iff there is $j \geq 0$ 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 $j \leq i$ 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 agglomeration rule in Fig. 2. 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.

5.2 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 [13, 14] (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.

We use the same notations as with BMC. The PDR method 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 $I \triangleq \underline{m_0}(\vec{x})$. 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 covers m (that is when $m' \geq m$, 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. [22]).

A formula F is *inductive* [14] when $I \Rightarrow F$ and $F(\vec{x}) \wedge \mathrm{T}(\vec{x}, \vec{x'}) \Rightarrow F(\vec{x'})$ hold. It is *inductive relative* to formula G if both $I \Rightarrow F$ and $G(\vec{x}) \wedge F(\vec{x}) \wedge \mathrm{T}(\vec{x}, \vec{x'}) \Rightarrow F(\vec{x'})$ hold. 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: $F_0 = I$, $F_i \Rightarrow F_{i+1}$ 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 $i \leq k+1$. 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.

Our implementation follows closely the algorithm for IC3 described in [14]. We only give a brief sketch of the OARS construction. Each of the F_i is computed as a formula in CNF (the conjunction of a set of clauses $CL(F_i)$) such that $CL(F_{i+1}) \subseteq CL(F_i)$. Intuitively, each clause is built from a *witness*, a marking such that $F_i(\vec{x}) \wedge \mathrm{T}(\vec{x}, \vec{x'}) \wedge (\neg F)(\vec{x'})$ is satisfiable.The procedure iterates through possible witnesses, say m, and pushes the clause $\neg\underline{m}(\vec{x})$ to the formulas F_k with $k < i$. Actually, we push a *minimal inductive cube* (MIC), c, such that $c \Rightarrow \neg\underline{m}$ and c is inductive relative to F_k. To overcome the problem with a potential infinite number of witnesses, 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 $m' \geq m$. 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 minimal inductive clauses from $\neg\underline{\hat{m}}(\vec{x})$ instead of $\neg\underline{m}(\vec{x})$. Another benefit of this choice is that $\underline{\hat{m}}$ is a conjunction of inequalities of the form $(x_j \geq k_i)$, 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).

Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$ and that $G_2(\vec{y})$ is the E-transform of formula $G_1(\vec{x})$ on N_1. We also assume that G_1 and G_2 are 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 Reduced tool when the initial net does not use inhibitor arcs.) To check that formula G_1 is an invariant on N_1, it is enough [13] 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. 4.1) 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 we have* $(N_1, m_1) \rhd_E (N_2, m_2)$ *and that* $G_2(\vec{y})$ *is the* E*-transform of* $G_1(\vec{x})$*, both monotonic formulas. Formula* G_1 *is an invariant on* N_1 *if and only if there exists* $i \geq 0$ *such that* $F'_i = F'_{i+1}$ *in the OARS built from net* N_2 *and formula* G_2*.*

5.3 Combination of BMC and PDR

In the next section, 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, EF ϕ where ϕ is monotonic (or equivalently, invariants AG ϕ 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 on an extensive benchmark of models and formulas provided by the Model Checking Contest (MCC) [2, 24].

SMPT serves as a front-end to generic SMT solvers, such as z3 [19, 10]. The tool can output sets of constraints using the SMT-LIB format [4] 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. For formulas, we accept properties

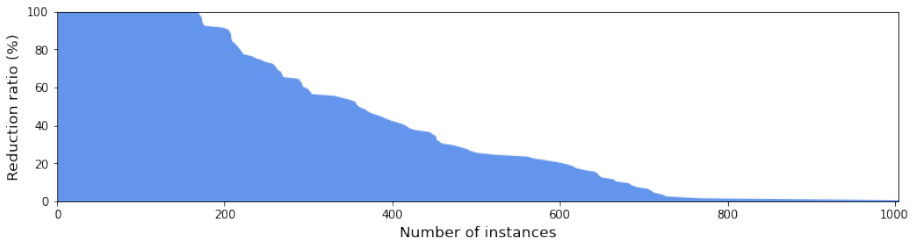


Figure 3: Distribution of reduction ratios over the instances in the MCC

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.

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, but a significant number of them 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. 3, 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 MCC2020, 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.)

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 below. We compared our results with the ones provided by an *oracle* [31], 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.

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)

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. (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 [32], 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).

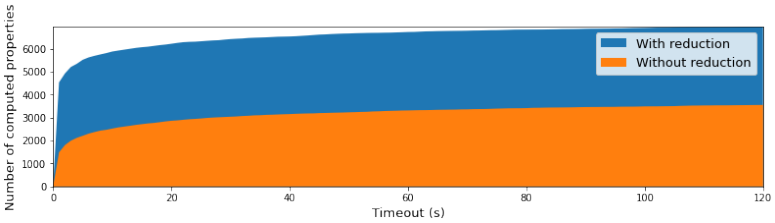


Figure 4: Number of computed properties in a limited time "with" (blue) and "without"(orange) reduction

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 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. 5, 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 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.

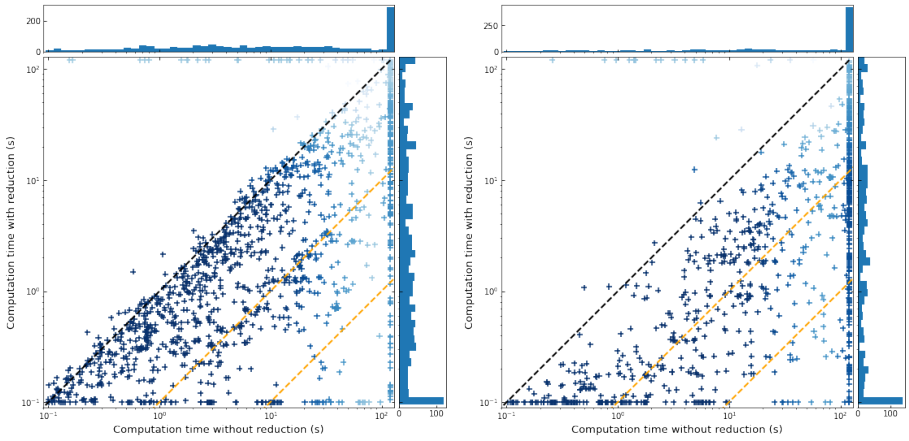
We observe that almost all the data points are below the diagonal, meaning reductions accelerate the computation, with many test cases exhibiting speed-ups larger than $\times 100$. We have added two light-coloured, dashed lines to materialize data points with speed-ups larger than $\times 10$ and $\times 100$ 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 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



(a) $r \in]0, 0.25[$ (b) $r \in [0.25, 0.5[$

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

Figure 5: Comparing computation time, "with" (y-axis) and "without" (x-axis) reductions for categories *Low* (a), *Good* (b), *High* (c) and *Full* (d).

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* [28, 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 [7].

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 [17] 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 practice, we derive polyhedral abstractions using *structural reductions*, a concept introduced by Berthelot in [5]. In our work, we are interested in reductions that preserves 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 [11], for instance, is an explicit-state model checker that combines Partial-Order Reduction techniques and structural reductions and can check property on Petri nets with weighted arcs and inhibitor arcs.

A more relevant example is ITS Tools [32], 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 s) that are not in the oracle results collected from ITS Tools [31].

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 [32] 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.

Indeed, 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 that are not bounded. 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 [20] or the extension of PDR to "well-structured transition systems" [26]. We can also mention the works on inductive procedures for infinite-state and/or parametrized systems, such as the verification methods used in Cubicle [18], or in [15, 23].

7.2 Future Work

We propose a new method that adapts our approach—initially developed for model checking with decision diagrams [6, 7]—for use with SMT solvers. We plan to continue in this direction, trying new verification methods and tackling properties more complex than reachability. For example, we already have plans [1] to apply our notion of polyhedral abstraction to the concurrent places problem [12].

There is also 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 motivate 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.

Finally, the most promising part of our work is to improve our adaptation of PDR, which raises several interesting problems. We have several ideas on how to improve our adaptation of PDR, and the computation of the Minimal Inductive Cube (MIC), while retaining completeness only in the case of bounded nets. This will be the subject of a future work.

References

[1] N. Amat. A new approach for the symbolic model checking of Petri nets. Master's thesis, University of Grenoble, 2020. Available at https://homepages.laas.fr/namat/Amat_master_thesis.pdf.

[2] E. Amparore, B. Berthomieu, G. Ciardo, S. Dal Zilio, F. Gallà, L. M. Hillah, F. Hulin-Hubard, P. G. Jensen, L. Jezequel, F. Kordon, D. Le Botlan, T. Liebke, J. Meijer, A. Miner, E. Paviot-Adet, J. Srba, Y. Thierry-Mieg, T. van Dijk, and K. Wolf. Presentation of the 9th edition of the model checking contest. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2019.

[3] A. Armando, J. Mantovani, and L. Platania. Bounded Model Checking of Software Using SMT Solvers Instead of SAT Solvers. In *Model Checking Software*, LNCS, pages 146–162. Springer, 2006.

[4] C. Barrett, P. Fontaine, and C. Tinelli. 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/.

[5] G. Berthelot. Transformations and Decompositions of Nets. In *Petri Nets: Central Models and their Properties*, LNCS, pages 359–376. Springer, 1987.

[6] B. Berthomieu, D. Le Botlan, and S. Dal Zilio. Petri net reductions for counting markings. In *International Symposium on Model Checking Software (SPIN)*, volume 10869 of *LNCS*, pages 65–84. Springer, 2018.

[7] B. Berthomieu, D. Le Botlan, and S. Dal Zilio. Counting Petri net markings from reduction equations. *International Journal on Software Tools for Technology Transfer*, 2019.

[8] F. Besson, T. Jensen, and J.-P. Talpin. Polyhedral analysis for synchronous languages. In *Static Analysis Symposium (SAS)*, volume 1694 of *LNCS*, pages 51–68. Springer, 1999.

[9] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS, pages 193–207. Springer, 1999.

[10] N. Bjørner. The z3 theorem prover. https://github.com/Z3Prover/z3/, 2020.

[11] F. M. Bønneland, J. Dyhr, P. G. Jensen, M. Johannsen, and J. Srba. Stubborn versus structural reductions for petri nets. *Journal of Logical and Algebraic Methods in Programming*, 102:46–63, 2019.

[12] P. Bouvier, H. Garavel, and H. Ponce-de León. Automatic decomposition of Petri nets into automata networks–a synthetic account. In *International Conference on Applications and Theory of Petri Nets and Concurrency*, volume 12152 of *LNCS*, pages 3–23. Springer, 2020.

[13] A. R. Bradley. SAT-Based Model Checking without Unrolling. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 6538 of *LNCS*, pages 70–87. Springer, 2011.

[14] A. R. Bradley. Understanding IC3. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 7317 of *LNCS*, pages 1–14. Springer, 2012.

[15] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. Infinite-state invariant checking with IC3 and predicate abstraction. *Formal Methods in System Design*, 49(3):190–218, 2016.

[16] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34, July 2001.

[17] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[18] S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaïdi. Cubicle: A Parallel SMT-Based Model Checker for Parameterized Systems. In *Computer Aided Verification (CAV)*, LNCS, pages 718–724. Springer, 2012.

[19] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS, pages 337–340. Springer, 2008.

[20] J. Esparza, R. Ledesma-Garza, R. Majumdar, P. Meyer, and F. Niksic. An SMT-Based Approach to Coverability Analysis. In *Computer Aided Verification (CAV)*, LNCS, pages 603–619, 2014.

[21] P. Feautrier. Automatic parallelization in the polytope model. In *The Data Parallel Programming Model*, volume 1132 of *LNCS*, pages 79–103. Springer, 1996.

[22] A. Finkel. The minimal coverability graph for petri nets. In *International Conference on Application and Theory of Petri Nets*, pages 210–243. Springer, 1991.

[23] A. Gurfinkel, S. Shoham, and Y. Meshman. SMT-based verification of parameterized systems. In *International Symposium on Foundations of Software Engineering*, pages 338–348. ACM, 2016.

[24] L. Hillah and F. Kordon. Petri Nets Repository: A tool to benchmark and debug Petri net tools. In *Application and Theory of Petri Nets and Concurrency*, volume 10258 of *LNCS*. Springer, 2017.

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

[26] J. Kloos, R. Majumdar, F. Niksic, and R. Piskac. Incremental, inductive coverability. In *Computer Aided Verification (CAV)*, pages 158–173. Springer, 2013.

[27] L. Lamport and E. Cohen. Reduction in TLA. 1466, Dec. 2016.

[28] R. J. Lipton. Reduction: a method of proving properties of parallel programs. *Communications of the ACM*, 18(12):717–721, Dec. 1975.

[29] J. C. Lloret, P. Azéma, and F. Vernadat. Compositional design and verification of communication protocols, using labelled petri nets. In E. M. Clarke and R. P. Kurshan, editors, *Computer-Aided Verification*, Lecture Notes in Computer Science, pages 96–105, Berlin, Heidelberg, 1991. Springer.

[30] M. Silva, E. Terue, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In *Advanced Course on Petri Nets*, pages 309–373. Springer, 1998.

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

[32] Y. Thierry-Mieg. Structural reductions revisited. In *Application and Theory of Petri Nets and Concurrency*, volume 12152 of *LNCS*, pages 303–323. Springer, 2020.

[33] Y. Thierry-Mieg, D. Poitrenaud, A. Hamez, and F. Kordon. Hierarchical set decision diagrams and regular models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS, pages 1–15. Springer, 2009.