Structural Reductions Revisited

Yann Thierry-Mieg

Structural Reductions Revisited

Yann Thierry-Mieg[1] [0000−0001−7775−1978]

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France first.last@lip6.fr

Abstract. Structural reductions are a powerful class of techniques that reason on a specification with the goal to reduce it before attempting to explore its behaviors. In this paper we present new structural reduction rules for verification of deadlock freedom and safety properties of Petri nets. These new rules are presented together with a large body of rules found in diverse literature. For some rules we leverage an SMT solver to compute if application conditions are met. We use a CEGAR approach based on progressively refining the classical state equation with new constraints, and memory-less exploration to confirm counter-examples. Extensive experimentation demonstrates the usefulness of this structural verification approach.

1 Introduction

Structural reductions can be traced back at least to Lipton's transaction reduction [13] and in the context of Petri nets to Berthelot's seminal paper [1]. A structural reduction rule simplifies the structure of the net under study while preserving properties of interest. Structural reductions are complementary of any other verification or model-checking strategies, since they build a simpler net that can be further analyzed using other methods.

The main idea in reduction rules is either to discard parts of the net or to accelerate over parts of the behaviors by fusing adjacent transitions. Reduction rules exploit the locality property of transitions to define a reduction's effect in a small neighborhood. Most rules can be adapted to support preservation of stutter-invariant temporal logic.

Structural reductions have been widely studied with generalisations that apply to many other models than Petri nets e.g. [12]. The classical reduction rules [1] include pre and post agglomeration, for which [9, 11] give broad general definitions that can be applied also to colored nets. More recently, several competitors in the Model Checking Contest have worked on the subject, [5] defines 8 reduction rules used in the tool Tapaal and [2] defines very general transition-centric reduction rules used in the tool Tina.

In this paper, we develop a new framework that integrates an SMT solver, a memory-less pseudo random walk and structural reduction rules with the goal of jointly reducing a model and a set of properties expressed as invariants. The SMT constraints and the reduction rules we propose include classic ones as well as many contributions original to this paper. These components together form a powerful reduction engine, that can in many cases push reductions to obtain an empty net and only true or false properties.

2 Definitions

Petri net syntax and semantics

Definition 1. *Structure. A Petri net* $N = \langle \mathcal{P}, \mathcal{T}, \mathcal{W}_-, \mathcal{W}_+, m_0 \rangle$ *is a tuple where* $\mathcal{P}$ *is the set of places,* $\mathcal{T}$ *is the set of transitions,* $\mathcal{W}_- : \mathcal{P} \times \mathcal{T} \mapsto \mathbb{N}$ *and* $\mathcal{W}_+ : \mathcal{P} \times \mathcal{T} \mapsto \mathbb{N}$ *represent the pre and post incidence matrices, and* $m_0 : \mathcal{P} \mapsto \mathbb{N}$ *is the initial marking.*

Notations: We use p (resp. t) to designate a place (resp. transition) or its index dependent on the context. We let markings m be manipulated as vectors of natural with $|\mathcal{P}|$ entries. We let $\mathcal{W}_-(t)$ and $\mathcal{W}_+(t)$ for any given transition t represent vectors with $|\mathcal{P}|$ entries. $\mathcal{W}_-^\top, \mathcal{W}_+^\top$ are the transposed flow matrices, where an entry $\mathcal{W}_-^\top(p)$ is a vector of $|\mathcal{T}|$ entries. We note $\mathcal{W}_e = \mathcal{W}_+ - \mathcal{W}_-$ the integer matrix representing transition *effects*.

In vector spaces, we use $v \geq v'$ to denote $\forall i, v(i) \geq v'(i)$, and offer sum $v + v'$ and scalar product $k \cdot v$ for scalar k with usual element-wise definitions.

We note $\bullet n$ (resp. $n\bullet$) the pre set (resp. post set) of a node n (place or transition). E.g. for a transition t its pre set is $\bullet t = \{p \in \mathcal{P} \mid \mathcal{W}_-(p,t) > 0\}$. A marking m is said to *enable* a transition t if and only if $m \geq \mathcal{W}_-(t)$. A transition t is said to *read* from place p if $\mathcal{W}_-(p,t) > 0 \wedge \mathcal{W}_e(p,t) = 0$.

Definition 2. *Semantics. The semantics of a Petri net are given by the firing rule* $\xrightarrow{t}$ *that relates pairs of markings: in any marking* $m \in \mathbb{N}^{|\mathcal{P}|}$*, if* $t \in \mathcal{T}$ *satisfies* $m \geq \mathcal{W}_-(t)$*, then* $m \xrightarrow{t} m'$ *with* $m' = m + \mathcal{W}_+(t) - \mathcal{W}_-(t)$*. The reachable set* $\mathcal{R}$ *is inductively defined as the smallest subset of* $\mathbb{N}^{|\mathcal{P}|}$ *satisfying* $m_0 \in \mathcal{R}$*, and* $\forall t \in \mathcal{T}, \forall m \in \mathcal{R}, m \xrightarrow{t} m' \Rightarrow m' \in \mathcal{R}$.

Properties of Interest We focus on deadlock detection and verification of safety properties. A net contains a deadlock if its reachable set contains a marking m in which no transition is enabled. A safety property asserts an invariant $\mathcal{I}$ that all reachable states must satisfy. The invariant is given as a Boolean combination $(\vee, \wedge, \neg)$ of atomic propositions that can compare $(\bowtie \in \{<, \leq, =, \geq, >\})$ arbitrary weighted sum of place markings to another sum or a constant, e.g. $\sum_{p \in \mathcal{P}} \alpha_p \cdot m(p) \bowtie k$, with $\alpha_p \in \mathbb{Z}$ and $k \in \mathbb{Z}$.

In the case of safety properties, the *support* of a property is the set of places whose marking is truly used in the predicate, i.e. such that at least one atomic proposition has a non zero α_p in a sum. The support $Supp \subseteq \mathcal{P}$ of the property defines the subset $\mathcal{S}_t \subseteq \mathcal{T}$ of *invisible* or *stuttering* transitions t satisfying $\forall p \in Supp, \mathcal{W}_e(p,t) = 0$ [1]. For safety, we are only interested in the projection of reachable markings over the variables in the support, values of places in $\mathcal{P} \setminus Supp$ are not *observable* in markings. A small support means more potential reductions, as rules mostly cannot apply to observed places or their neighborhood.

3 Property Specific Reduction

We are given a Petri net N and either a set of safety invariants or a deadlock detection query. We consider the problem of building a structurally smaller net N' and/or simpler properties such that the resulting properties hold on the smaller net if and only if the original properties hold on the original net. In particular, properties that can be proven or

[1] This sufficient condition for being stuttering might be relaxed by a more refined examination of the property and the effect of transitions on its truth value with respect to its value initially.

disproven reduce to true or false, and when all properties are thus simplified, an empty system N' is enough to answer the problem.

In this paper we introduce a combination of three solution strategies: 1. we try to disprove an invariant by using a memory-less exploration that can randomly or with guidance encounter counter-example states thus *under-approximate* the behavior, 2. we try to prove an invariant holds using a system of SMT constraints to over-approximate reachable states and 3. we use structural reduction rules that preserve the properties of interest through a transformation. These approaches reinforce and complement each other and together provide a structural decision procedure often able to reduce the system to an empty net.

We consider an over-approximation of the state space, symbolically represented as set of constraints over a system of variables encoded in an SMT solver. We use this approximation to detect unfeasible behavior.

The SMT procedure while powerful is only a semi-decision procedure in the sense that UNSAT answers *prove that the invariant* holds ($\neg\mathcal{I}$ is *not* reachable), but SAT answers are not trusted because we work with an over-approximation of the system. The now classic CEGAR scheme [6] proposes an elegant solution to this problem, consisting in replaying the abstract candidate counter-example on the original system, to try to exhibit a concrete counter-example thus proving the invariant *does not hold*. Similarly to [17] our constraint system is able to provide along with SAT answers a Parikh firing count (see Sec. 4) that can guide to a concrete counter-example.

We thus engineered a memory-less and fast transition engine able to explore up to millions of states per second to *under-approximate* behaviors by sampling. This engine can run in pseudo-random exploration mode or can be guided by a Parikh firing count coming from the SMT engine. If it can find a reachable marking that does not satisfy $\mathcal{I}$ the invariant is disproved.

We combine these solutions with a set of structural reduction rules (Sec. 5), that can simplify the net by examining its structure, and provide a smaller net where parts of the behavior are removed . The resulting simplified net and set of properties can then be exported in a format homogeneous to the initial input, typically for processing by a full-blown model-checker. If all properties have been reduced to true or false, the net will be empty.

4 Proving with SMT Constraints

In this section, we define an over-approximation of the state space, symbolically manipulated as a set of constraints over a system of variables encoded in a Satisfiability Modulo Theory (SMT) solver [15]. We use this approximation to detect unfeasible behavior. We present constraints that can be progressively added to a solver to over-approximate the state space with increasing accuracy. Structural reduction rules based on a behavioral characterisation of target conditions are also possible in this context (see Sec .7).

4.1 Approximating with SMT

SMT solvers are a powerful and relatively recent technology that enables flexible modeling of constraint satisfaction problems using first order logic and rich data types and

theories, as well as their combinations. We use both linear arithmetic over reals and integers (LRA and LIA) to approximate the reachable set of states by constraints over variables representing the marking of places.

As first step in all approaches, we define for each place $p \in \mathcal{P}$ a variable m_p that represents its marking. These variables are initially only constrained to be positive: $\forall p \in \mathcal{P}, m_p \geq 0$. If we know that the net is one-safe (all place markings are at most one) e.g. because the net was generated, we add that information: $\forall p \in \mathcal{P}, m_p \leq 1$.

We then suppose that we are trying to find a reachable marking that *invalidates* a given invariant property I over a support. In other words we assert that the m_p variables satisfy $\neg I$. For deadlocks, we consider the invariant I asserting that at least one transition is enabled, expressed in negative form as $\neg I = \forall t \in \mathcal{T}, \exists p \in \bullet t, m_p < \mathcal{W}_-(p,t)$, and thus reduce the Deadlock problem to Safety.

An UNSAT answer is a definitive "NO" that ensures that I is indeed an invariant of the system. A SAT answer provides a candidate marking m_c with an assignment for the m_p variables, but is unreliable since we are considering an over-approximation of the state-space. To define the state equation constraint (see below) we also add a variable n_t for each transition t that counts the number of times it fired to reach the solution state. This Parikh count provides a guide for the random explorer.

Workflow Because we are hoping for a definitive UNSAT answer, and that we have a large set of constraints, they are fed incrementally to the solver, hoping for an early UNSAT result. In practice we check satisfiability after every few assertions, and start with the simpler constraints that do not need additional variables.

Real arithmetic is much easier to solve than integer arithmetic, and reals are an over-approximation of integers since if no solution exists in reals (UNSAT), none exists in integers either. We therefore always first incrementally add constraints using the real domain, then at the end of the procedure, if the result is still SAT, we test if the model computed (values for place markings and Parikh count) are actually integers. If not, we escalate the computation into integer domain, restarting the solver and introducing again from the simplest constraints. At the end of the procedure we either get UNSAT or a "likely" Parikh vector that can be used to cheaply test if the feasibility detected by SAT answer is actually doable.

4.2 Incremental Constraints

We now present the constraints we use to approximate the state space of the net, in order of complexity and induced variables that corresponds to the order they are fed to our solver. We progressively add generalized flow constraints, trap constraints, the state equation, read arc constraints, and finally add new "causality" constraints.

Generalized flows A flow $\mathcal{F}$ is a weighted sum of places $\sum_{p \in \mathcal{P}} \alpha_p \cdot m_p$ that is an invariant of the state space. A semi-flow only has positive integers as $\alpha_i \in \mathbb{N}$ coefficients while generalized flows may have both positive and negative integer coefficients $\alpha_i \in \mathbb{Z}$.

It is possible to compute all generalized flows of a net in polynomial time and space, and the number of flows is itself polynomial in the size of the net. We use for this

purpose a variant of the algorithm described in [7], initially adopted from the code base of [3] and then optimized for our scenario. This provides a polynomial number of simple constraints each only having as variables a subset of places (the α_i are fixed). The constant value of the flow in any marking can be deduced from the initial marking of the net. We do not attempt to compute semi-flows as there can be an exponential number of them, but we still first assert semi-flow constraints (if any were found) before asserting generalized flows as these have far fewer solutions due to markings being positive.

Trap constraints In [8], to reinforce a system of constraints on reachable states, "trap constraints" are proposed. Such a constraint asserts that an initially marked trap must remain marked.

Definition 3. *Trap. A trap S is a subset of places such that any transition consuming from the set must also feed the set. $S \subseteq \mathcal{P}$ is a* trap *iff.*

$$\forall p \in S, \forall t \in p\bullet, \exists p' \in t\bullet \wedge p' \in S.$$

The authors show that traps provide constraints that are a useful complement to state equation based approaches, as they can discard unfeasible behavior that is otherwise compatible with the state equation. The problem is that in general these constraints are worst case exponential in number. Leveraging the incremental nature of SMT solvers, we therefore propose to only introduce "useful" trap constraints, that actually contradict the current candidate solution.

We consider the candidate state m_c produced as SAT answer to previous queries, and try to contradict it using a trap constraint: we look for an initially marked trap that is empty in the solution. The search for such a potential trap can be done using a separate SMT solver instance.

For each place p in the candidate state m_c, we introduce a Boolean variable b_p that will be true when p is in the trap. We then add the trap constraints:

$\exists p \in \mathcal{P}, m_0(p) > 0 \wedge b_p$ trap was initially marked
$\forall p \in \mathcal{P}, m_c(p) > 0 \Rightarrow \neg b_p$ no finally marked places
$b_p \Rightarrow \forall t \in p\bullet, \exists p' \in t\bullet \wedge b_{p'}$ trap definition

If this problem is SAT, we have found a trap S from which we can derive a constraint expressed as $\bigvee_{p \in S} m_p > 0$ that can be added to the main solution procedure. Otherwise, no trap constraint existed that could contradict the given witness state. The procedure is iterated until no more useful trap constraints are found or UNSAT is obtained.

State equation The state equation [16] is one of the best known analytical approximations of the state space of a Petri net.

Definition 4. *State equation. We define for each transition $t \in \mathcal{T}$ a variable $n_t \in \mathbb{N}$. We assert that*

$$\forall p \in P, m_p = m_0(p) + \sum_{t \in p\bullet \cup \bullet p} n_t \cdot \mathcal{W}_e(p,t)$$

The state equation constraint is thus implemented by adding for each transition $t \in \mathcal{T}$ a variable $n_t \geq 0$ and then asserting for each place $p \in \mathcal{P}$ a linear constraint. Instead of

considering all transitions and adding a variable for each of them, we can limit ourselves to one variable per possible transition *effect*, thus having a single variable for transitions t, t' such that $\mathcal{W}_e(t) = \mathcal{W}_e(t')$. Care must be taken when interpreting the resulting Parikh vectors however. In the worst case, the state equation adds $|\mathcal{T}|$ variables and $|\mathcal{P}|$ constraints, which can be expensive for large nets. We always start by introducing the constraints bearing on places in the support.

A side effect of introducing the state equation constraints is that we now have a candidate Parikh firing vector, in the form of the values taken by n_t variables. The variables in this Parikh vector can now be further constrained, as we now show.

Read ⇒ Feed Constraints A known limit of the state equation is the fact that it does not approximate read arc behavior very well, since it only reasons with actual effects $\mathcal{W}_e$ of transitions. However we can further constrain our current solution to the state equation by requiring that for any transition t used in the candidate Parikh vector, that reads from an initially insufficiently marked place p, there must be a transition t' with a positive Parikh count that feeds p.

Definition 5. *Read Arc Constraint. For each transition $t \in \mathcal{T}$, for every initially insufficiently marked place it reads from i.e. $\forall p \in \bullet t$, such that $\mathcal{W}_e(p,t) = 0 \wedge \mathcal{W}_-(p,t) > m_0(p)$, we assert that:*

$$n_t > 0 \Rightarrow \bigvee_{\{t' \in \bullet p \setminus \{t\} \ | \mathcal{W}_e(p,t') > 0\}} n_{t'} > 0$$

These read arc constraints are easy to compute and do not introduce any additional variables so they can usually safely be added after the problem with the state equation returned SAT, thus refining the solution.

While in practice it is rare that these additional constraints allow to conclude UNSAT, they frequently improve the feasibility of the Parikh count solution on nets that feature a lot of read arcs (possibly due to reductions), going from an unfeasible solution to one that is in fact realizable.

Causality Constraints Solutions to the state equation may contain transition cycles, that "borrow" non-existing tokens and then return them. This leads to spurious solutions that are not in fact feasible. However, we can break such cycles of transitions if we consider the partial order that exists over the first occurrence of each transition in a potential concrete trace realizing a Parikh vector.

Indeed, any time a transition t consumes a token in place p, but p is not initially sufficiently marked, it must be the case that there is another transition t' that feeds p, and that t' *precedes* t in the trace.

We thus consider a precedes $\prec \subseteq \mathcal{T} \times \mathcal{T}$ relation between transitions that is : non-reflexive $\forall t \in \mathcal{T}, \neg(t \prec t)$, transitive $\forall t_1, t_2, t_3 \in \mathcal{T}, t_1 \prec t_2 \wedge t_2 \prec t_3 \Rightarrow t_1 \prec t_3$, anti-symmetric $\forall t_1, t_2 \in \mathcal{T}, t_1 \prec t_2 \Rightarrow \neg(t_2 \prec t_1)$. This relation defines a strict partial order over transitions.

Definition 6. *Causality Constraint. We add the definition of the precedes relation to the solver. For each transition $t \in \mathcal{T}$, for each input of its input places that is insufficiently*

marked i.e. $\forall p \in \bullet t, \mathcal{W}_-(p,t) > m_0(p)$*, we assert that*

$$n_t > 0 \Rightarrow \bigvee_{\{t' \in \bullet p \setminus \{t\} \mid \mathcal{W}_e(p,t') > 0\}} (n_{t'} > 0 \wedge t' \prec t)$$

These constraints reflect the fact that insufficiently marked places must be fed *before* a continuation can take place. These constraints offer a good complement to the state equation as they forbid certain Parikh solutions that use a cycle of transitions and "borrow" tokens: such an Ouroboros-like cycle now needs a causal predecessor to be feasible. Our solutions still over-approximate the state-space as we are only reasoning on the first firing of each transition, and we construct conditions for each predecessor place separately, so we cannot guarantee that all input places of a transition have been *simultaneously* marked.

Notes: The addition of causal constraints forming a partial order to refine state equation based reasoning has not been proposed before in the literature to our knowledge.

To encode the *precedes* constraints in an SMT solver the approach we found most effective in practice consists in defining a new integer (or real) variable o_t for each transition t, and use strict inferior $o_{t_1} < o_{t_2}$ to model the precedes relation $t_1 \prec t_2$. This remains a partial order as some o_t variables may take the same value, and avoids introducing any additional theories or quantifiers.

5 Structural Reduction Rules

This section defines a set of structural reduction rules. For each rule, we give a name and identifier; whether it is applicable to deadlock detection, safety or both; an informal description of the rule; a formal definition of the rule; a sketch of correctness where $\Rightarrow$ proves that states observably satisfying the property (or deadlocks) are not removed by the reduction, $\Leftarrow$ proves that new observable states (or deadlocks) are not added.

Deadlock detection can be stated as the invariant "at least one transition is enabled". But this typically implies that all places are in the support, severely limiting rule application. So instead we define deadlock specific reductions that consider that the support is empty, and that are mainly concerned with preserving divergent behavior (loops).

5.1 Elementary Transition rules

Rule 1. Equal transitions modulo k
Applicability: Safety, Deadlock
Description: When two transitions are equal modulo k, the larger one can be discarded.
Definition: If $\exists t, t' \in \mathcal{T}, \exists k \in \mathbb{N}, \mathcal{W}_-(t) = k \cdot \mathcal{W}_-(t') \wedge \mathcal{W}_+(t) = k \cdot \mathcal{W}_+(t')$, discard t.
Correctness: $\Rightarrow$: In any state where t is enabled, t' must be enabled, and firing k times t' leads to the same state as firing t. $\Leftarrow$: Discarding transitions cannot add states. For deadlocks, any state enabling t in the original net still has a successor by t'.

Rule 2. Dominated transition
Applicability: Safety, Deadlock
Description: When a transition t has the same effect as t' but more preconditions, which

can happen due to read arc behavior, t can be discarded.
Definition: If $\exists t, t' \in \mathcal{T}, \mathcal{W}_e(t) = \mathcal{W}_e(t') \wedge \mathcal{W}_-(t) \geq \mathcal{W}_-(t')$, discard t.
Correctness: $\Rightarrow$: any state that enables t also enables t', and the resulting state is the same. $\Leftarrow$: Discarding transitions cannot add states. For deadlocks, any state enabling t in the original net still has successors by t' in the resulting net.

Rule 3. Redundant Composition
Applicability: Safety, Deadlock
Description: When a transition t has the same effect and more input places than a composition $t_1.t_2$, where t_1 enabled implies $t_1.t_2$ is enabled, t can be discarded.
Definition: If $\exists t, t_1, t_2 \in \mathcal{T}, \mathcal{W}_-(t) \geq \mathcal{W}_-(t_1), \mathcal{W}_+(t_1) \geq \mathcal{W}_-(t_2), \mathcal{W}_e(t) = \mathcal{W}_e(t_1) + \mathcal{W}_e(t_2)$, discard t.
Correctness: $\Rightarrow$: any state that enables t also enables t_1, t_2 is necessarily enabled after firing t_1, and the state reached by the sequence $t_1.t_2$ is the same as reached by t. $\Leftarrow$: Discarding transitions cannot add states. For deadlocks, any state enabling t in the original net still has successors by t_1 in the resulting net.
Notes: This pattern could be extended to compositions of more transitions, but there is a risk of explosion as we end up exploring intermediate states of the trace. [2] notably proposes a more general version that subsumes the four first rules presented here but is more costly to evaluate.

Rule 4. Neutral transition
Applicability: Safety
Description: When a transition has no effect it can be discarded.
Definition: If $\exists t \in \mathcal{T}, \mathcal{W}_-(t) = \mathcal{W}_+(t)$, discard t.
Correctness: $\Rightarrow$: any state reachable by a firing sequence using t can also be reached without firing t. $\Leftarrow$: Discarding transitions cannot add states.

Rule 5. Sink Transition
Applicability: Safety
Description: A transition t that has no outputs and is stuttering can be discarded.
Definition: If $\exists t \in \mathcal{S}_t, t\bullet = \emptyset$, discard t.
Correctness: $\Rightarrow$: any state reached by firing t enables less subsequent behaviors since tokens were consumed by t. Any firing sequence of the original net using t is still possible in the new net if occurrences of t are removed, and leads to a state satisfying the same properties because t stutters. $\Leftarrow$: Discarding transitions cannot add states.

Rule 6. Source Transition
Applicability: Deadlock
Description: If a transition has no input places then the net has no reachable deadlocks.
Definition: If $\exists t \in \mathcal{T}, \bullet t = \emptyset$, discard all places and discard all transitions except t.
Correctness: $\Rightarrow$: Since t is fireable in any reachable state, there cannot be reachable deadlock states. $\Leftarrow$: The resulting model has no deadlocks, like the original model.

5.2 Elementary Place rules

Rule 7. Equal places modulo k
Applicability: Deadlock, Safety

Description: When two places are equal modulo k (flow matrices and initial marking), either one can be discarded (we discard the larger one).
Definition: If $\exists p, p' \in \mathcal{P} \setminus Supp, \exists k \in \mathbb{N}, m_0(p) = k \cdot m_0(p'), \mathcal{W}_-^\top(p) = k \cdot \mathcal{W}_-^\top(p') \wedge \mathcal{W}_+^\top(p) = k \cdot \mathcal{W}_+^\top(p')$, discard p.
Correctness: $\Rightarrow$: Removing a place cannot remove any behavior. $\Leftarrow$: Inductively we can show that $m(p) = k \cdot m(p')$ in any reachable marking m. Thus enabling conditions on output transitions t of $p\bullet = p'\bullet$ are always equivalent: either p and p' are both insufficiently marked or both are sufficiently marked to let t fire. Removing one of these two conditions thus does not add any behavior.

Rule 8. Sink Place
Applicability: Deadlock, Safety
Description: When a place p has no outputs, and is not in the support of the property, it can be removed.
Definition: If $\exists p \in \mathcal{P} \setminus Supp, p\bullet = \emptyset$, discard p
Correctness: $\Rightarrow$: Removing a place cannot remove any behavior. $\Leftarrow$: Since the place had no outputs it already could not enable any transition in the original net.

Rule 9. Constant place
Applicability: Deadlock, Safety
Description: When a place's marking is constant (typically because of read arc behavior), the place can be removed and the net can be simplified by "evaluating" conditions on output transitions.
Definition: If $\exists p \in \mathcal{P}, \mathcal{W}_-^\top(p) = \mathcal{W}_+^\top(p)$, discard p and any transition $t \in \mathcal{T}$ such that $\mathcal{W}_-(p,t) > m_0(p)$.
Correctness: $\Rightarrow$: Removing a place cannot remove any behavior. The transitions discarded could not be enabled in any reachable marking so no behavior was lost. $\Leftarrow$: The remaining transitions of $p\bullet$ have one less precondition, but it evaluated to true in all reachable markings, so no behavior was added. Discarding transitions cannot add states.
Notes: This reduction also applies to places in the support, leading to simplification of the related properties.

Rule 10. Maximal Unmarked Siphon
Applicability: Deadlock, Safety
Description: An unmarked *siphon* is a subset of places that are not initially marked and never will be in any reachable state. These places can be removed and adjacent transitions can be simplified away.
Definition: A maximal unmarked siphon $S \subseteq \mathcal{P}$ can be computed by initializing with the set of initially unmarked places $S = \{p \in \mathcal{P} \mid m_0(p) = 0\}$, and $T \subseteq \mathcal{T}$ with the full set $\mathcal{T}$ then iterating:

- Discard from T any transition that has no outputs in S, $t \in T, t\bullet \cap S = \emptyset$,
- Discard from T any transition t that has no inputs in S and discard all of t's output places from S. So $\forall t \in T$, if $\bullet t \cap S = \emptyset$, discard t from T and discard $t\bullet$ from S,
- iterate until a fixed point is reached.

If S is non-empty, discard any transition t such that $\bullet t \cap S \neq \emptyset$ and all places in S.
Correctness: $\Rightarrow$: The discarded transitions were never enabled so no behavior was

lost. Removing places cannot remove behavior. $\Leftarrow$: Removing transitions cannot add behavior. The places removed were always empty so they could not enable any transition.
Notes: Siphons have been heavily studied in the literature [14]. This reduction also applies to places in the support, leading to simplification of the related properties.

Rule 11. Bounded Marking Place
Applicability: Deadlock, Safety
Description: When a place p has no true inputs, i.e. all transitions effects can only reduce the marking of p, $m_0(p)$ is an upper bound on its marking that can be used to reduce adjacent transitions.
Definition: If $\exists p \in \mathcal{P}, \forall t \in \mathcal{T}, \mathcal{W}_e(p,t) \leq 0$, discard any transition $t \in \mathcal{T}$ such that $\mathcal{W}_-(p,t) > m_0(p)$
Correctness: $\Rightarrow$: Since the transitions discarded were never enabled in any reachable marking, removing them cannot lose behaviors. $\Leftarrow$: Discarding transitions cannot add behaviors.

Rule 12. Implicit Fork/Join place
Applicability: Deadlock, Safety
Description: Consider a place p not in the support that only touches two transitions: t_{fork} with two outputs (of which p) and t_{join} with two inputs (of which p). If we can prove that the only tokens that can mark the other input p' of t_{join} must result from firings of t_{fork}, p is implicit and can be discarded.
Definition: If $\exists p \in \mathcal{P} \setminus Supp, \exists t_f, t_j \in \mathcal{T}, {\bullet}p = \{t_f\} \wedge p{\bullet} = \{t_j\}, \mathcal{W}_+(p,t_f) = \mathcal{W}_-(p,t_j) = 1, t_f{\bullet} = \{p, p''\} \wedge \mathcal{W}_+(p'',t_f) = 1, {\bullet}t_j = \{p, p'\} \wedge \mathcal{W}_-(p',t_j) = 1$, then if p' is *induced by* t_f, discard p.
We use a simple recursive version providing sufficient conditions for the test "is p induced by t":

- If p has t as single input and $\mathcal{W}_+(p,t) = 1$, return true.
- If p has a single input t' and $\mathcal{W}_+(p,t') = 1$, if there exists any input p' of t', such that $\mathcal{W}_+(p',t') = 1$, and (recursively) p' is induced by t return true. Else false.

Correctness: $\Rightarrow$: Removing a place cannot remove any behavior. $\Leftarrow$: In any marking that disabled t_j, either both p and p' were unmarked, or only p' was unmarked. Removing the condition on p thus does not add behavior.
Notes: There are many ways we could refine the "induced by" test, and widen the application scope, but the computation should remain fast. We opted here for a reasonable complexity vs. applicability trade-off. The implementation further bounds recursion depth (to 5 in the experiments), and protects against recursion on a place already in the stack. Implicit places are studied in depth in [10], the concept is used again in SMT backed Rule 21.

Rule 13. Future equivalent place
Applicability: Deadlock, Safety
Description: When two places p and p' enable isomorphic behaviors up to permutation of p and p', i.e. any transition consuming from p has an equivalent but that consumes from p', the tokens in p and p' enable the same future behaviors. We can fuse the two

places into p, by redirecting arcs that feed p' to instead feed p.
Definition: We let $v \equiv_{p|p'} v'$ denote equality under permutation of elements at index p and p' of two vectors v and v'.

If $\exists p, p' \in \mathcal{P} \setminus Supp, p\bullet \cap p'\bullet = \emptyset, \forall t \in p\bullet, \mathcal{W}_-(p,t) = 1$
$\wedge \exists t' \in p'\bullet, \mathcal{W}_-(t) \equiv_{p|p'} \mathcal{W}_-(t') \wedge \mathcal{W}_+(t) \equiv_{p|p'} \mathcal{W}_+(t')$,
then $\forall t \in \bullet p'$, set $\mathcal{W}_+(p,t) = \mathcal{W}_+(p,t) + \mathcal{W}_+(p',t)$, update initial marking to $m'_0(p) = m_0(p) + m_0(p')$, and discard p' and transitions in $p'\bullet$.
Correctness: $\Rightarrow$: Any firing sequence of the original net using transitions consuming from p' still have an image using the transitions feeding from p. These two traces are observation equivalent since neither p nor p' are in the support, so no behavior was lost. This transformation does not preserve the bounds on p's marking however. $\Leftarrow$: The constraints on having only arcs with value 1 feeding from p and not having common output transitions feeding from both p and p' ensure there is no confusion problem for the merged tokens in the resulting net; a token in p' of the original net allowed exactly the same future behaviors (up to the image permutation) as any token in p of the resulting net. Merging the tokens into p thus did not add more behaviors. Discarding transitions cannot add states.
Notes: This test can be costly, but sufficient conditions for non-symmetry allow to limit complexity and prune the search space, e.g. we group places by number of output transitions, use a sparse "equality under permutation test"... The effect can be implemented as simply moving the tokens in p' to p and redirecting arcs to p, other rules will then discard the now constant place p' and its outputs.

5.3 Agglomeration rules

Agglomeration in p consists in replacing p and its surrounding transitions (feeders and consumers) to build instead a transition for every element in the Cartesian product $\bullet p \times p\bullet$ that represents the effect of the sequence of firing a transition in $\bullet p$ then immediately a transition in $p\bullet$. This "acceleration" of tokens in p reduces interleaving in the state space, but can preserve properties of interest if p is chosen correctly. This type of reduction has been heavily studied [11, 12] as it forms a common ground between structural reductions, partial order reductions and techniques that stem from transaction reduction.
Definition 7. *Agglomeration of a place $p \in \mathcal{P}$:*
$\forall h \in \bullet p, \forall f \in p\bullet$, *define a new transition t:*

$$\begin{cases} Let\ k = \frac{\mathcal{W}_+(p,h)}{\mathcal{W}_-(p,f)}, k \in \mathbb{N}, k \geq 1, \\ \mathcal{W}_-(t) = \mathcal{W}_-(h) + k \cdot \mathcal{W}_-(f) \wedge \mathcal{W}_+(t) = \mathcal{W}_+(h) + k \cdot \mathcal{W}_+(f) \end{cases}$$

Discard transitions in $\bullet p$ and $p\bullet$. Discard place p.

Note the introduction of the k factor, reflecting how many times f can be fed by one firing of h. This factor should be a natural number for the agglomeration to be well defined. As a post processing, it is recommended to apply identity reduction Rule 1 to the set of newly created transitions, this set is much smaller than the full set of transitions but often contains duplicates.

Rule 14. Pre Agglomeration
Applicability: Deadlock, Safety
Description: Basically, we assert that once an $h \in \bullet p$ transition becomes enabled, it will stay enabled until some tokens move into the place p by actually firing h. Transition h cannot feed any other places, so the only behaviors it enables are continuations f that feed from p. So we can always "delay" the firing of h until it becomes relevant to enable an f transition. We fuse the effect of tokens exiting p using f with its predecessor action h, build a set of $h.f$ agglomerate actions and discard p.
Definition:

$\exists p \in \mathcal{P} \setminus Supp$ p not in support
$m_0(p) = 0$ initially unmarked
$\bullet p \cap p\bullet = \emptyset$ distinct feeders and consumers
$\bullet p \subseteq \mathcal{S}_t$ feeders are stuttering
$\forall h \in \bullet p,$

$$\begin{cases} \mathcal{W}_+(p,h) = 1, & \text{feed arc weights are one} \\ h\bullet = \{p\} & p \text{ is the single output of } h \\ \exists p_1 \in \mathcal{P}, \mathcal{W}_+(p_1,h) < \mathcal{W}_-(p_1,h) & h \text{ is divergent free} \\ \forall p_2 \in \bullet h, p_2\bullet = \{h\} & h \text{ is strongly quasi-persistent} \end{cases}$$

$\forall f \in p\bullet,$

$$\{\mathcal{W}_-(p,f) = 1 \text{ consume arc weights are one}$$

Then perform a pre agglomeration in p.
Correctness: $\Rightarrow$: Any sequence using an h and an f still has an image using the agglomerated transition $h.f$ in the position f was found in the original sequence. Because h is invisible and only feeds p that is itself not in the support, delaying it does not lose any observable behaviors. $\Leftarrow$: Any state that is reachable in the new net by firing an agglomerate transition $h.f$ was already reachable by firing the sequence h then f in the original net, so no behavior is added.
Notes: Pre agglomeration is one of the best known rules, this version is generalized to more than one feeder or consumer and uses terminology taken from [11].

Rule 15. Post Agglomeration
Applicability: Deadlock, Safety
Description: Basically, we assert that once p is marked, it fully controls its outputs, so the tokens arriving in p necessarily have the choice of when and where they wish to go to. Provided p is not in the support and its output transitions are invisible, we can fuse the effects of feeding p with an immediate choice of what happens to those tokens after that. We fuse the effect of tokens entering p using h with a successor action f, build a set of $h.f$ agglomerate actions and discard p.
Definition: If

$\exists p \in \mathcal{P} \setminus Supp$ p not in support
$m_0(p) = 0$ initially unmarked
$\bullet p \cap p\bullet = \emptyset$ distinct feeders and consumers
$p\bullet \subseteq \mathcal{S}_t$ consumers are stuttering
$\forall f \in p\bullet,$

$$\begin{cases} \bullet f = \{p\} & \text{no other inputs to } f \\ \forall h \in \bullet p, \mathcal{W}_-(p,h)/\mathcal{W}_+(p,f) \in \mathbb{N} & \text{natural ratio constraint} \end{cases}$$

Then perform a post agglomeration in p.

Correctness: $\Rightarrow$: Any sequence using an h still has an image using the agglomerated transition $h.f$ in the position h was found in the original sequence, that leads to a state satisfying the same propositions since f transitions stutter. Any sequence using an f transition must also have an h transition preceding it, and the same trace where the f immediately follows the h is feasible in both nets and leads to a state satisfying the same propositions. It is necessary that the f transitions stutter so that moving them in the trace to immediately follow h does not lead to observably different states. $\Leftarrow$: Any state that is reachable in the new net by firing an agglomerate transition $h.f$ was already reachable by firing the sequence h then f in the original net, so no behavior is added.

Notes: Post agglomeration has been studied a lot in the literature e.g. [11], this version is generalized to an arbitrary number of consumers and feeders and a natural ratio constraint on arc weights. This procedure can grow the number of transitions when both $|\bullet p|$ and $|p\bullet|$ are greater than one, which becomes more likely as agglomeration rules are applied, and can lead to an explosion in the number of transitions of the net. In practice we refuse to agglomerate when the Cartesian product size is larger than 32.

Rule 16. Free Agglomeration

Applicability: Safety

Description: Basically, we assert that all transitions h that feed p *only* feed p and are invisible. It is possible that the original net lets h fire but never enables a continuation f, these behaviors are lost since resulting $h.f$ is never enabled, making the rule only valid for safety. In the case of safety, firing h makes the net lose tokens, allowing *less* observable behaviors until a continuation $f \in p\bullet$ is fired, so the lost behavior leading to a dead end was not observable anyway. We agglomerate around p.

Definition:

$\exists p \in \mathcal{P} \setminus Supp$ — p not in support

$m_0(p) = 0$ — initially unmarked

$\bullet p \cap p\bullet = \emptyset$ — distinct feeders and consumers

$\bullet p \subseteq \mathcal{S}_t$ — feeders are stuttering

$\forall h \in \bullet p,$

$$\begin{cases} h\bullet = \{p\} & p \text{ is the single output of } h \\ \mathcal{W}_+(p,h) = 1 & \text{feed arc weights is one} \end{cases}$$

$\forall f \in p\bullet,$

$$\{\mathcal{W}_-(p,f) = 1 \quad \text{consume arc weights is one}$$

Then perform a free agglomeration in p.

Correctness: $\Rightarrow$: If there exists a sequence using one of the f transitions in the original system, it must contain an h that precedes the f. The trace would also be possible if we delay the h to directly precede the f, because h only stores tokens in p, it cannot causally serve to mark any other place than p, and since h transitions are stuttering it leads to the same observable state in the new system. Traces that do not use an f transition are not impacted. Sequences that use an h but not an f are no longer feasible, but because h transitions stutter, the same sequence without the h that is still possible in the new system would lead to the same observable states. So no observable behavior is lost as sequences and behaviors that are lost were not observable. $\Leftarrow$: Any state that is reachable in the new net by firing an agglomerate transition $h.f$ was already reachable

by firing the sequence h then f in the original net, so no behavior is added.
Notes: Free agglomeration is a new rule, original to this paper that can be understood as relaxing conditions on pre-agglomeration in return for less property preservation. It is a reduction that may remove deadlocks, as it is no longer possible to fire h without f, which forbids having tokens in p that could potentially be stuck because no f can fire. After firing h the net is less powerful since we took tokens from it and placed them in p, these situations are no longer reachable.

Rule 17. Controlling Marked Place
Applicability: Deadlock, Safety
Description: A place p that is initially marked and which is the only input of its single stuttering output transition t, can be emptied using t. Since p controls its output, once it is emptied it will be post-agglomerable.
Definition: If $\exists p \in \mathcal{P}, p\bullet = \{t\}, \bullet t = \{p\}, p \notin t\bullet, \exists k \in \mathbb{N}, m_0(p) = k \cdot \mathcal{W}_-(p,t)$, update $m'_0 = m_0 + k \cdot \mathcal{W}_e(t)$.
Correctness: $\Rightarrow$: Since t is stuttering and only consumes from p, firing it at the beginning of any firing sequence will not change the truth value of the property in the reached state. $\Leftarrow$: the new initial state was already reachable in the original model.
Notes: This is the first time to our knowledge that a structural reduction rule involving token movement is proposed. This reduction also may also consume some prefix behavior as long as a single choice is available.

6 Graph-based Reduction Rules

In this section we introduce a set of new rules that reason on a structural over approximation of the net behavior to quickly discard irrelevant behavior. The main idea is to study variants of the token flow graph underlying the net to compute when sufficient conditions for a reduction are met.

In these graphs, we use places as nodes and add edges that partly abstract away the transitions. Different types of graphs considered, all are abstractions of the structure of the net. A graph is a tuple $G = (N, E)$ where nodes N are places $N \subseteq \mathcal{P}$ and edges E in $\mathcal{P} \times \mathcal{P}$ are oriented. We can notice these graphs are small, at most $|\mathcal{P}|$ nodes, so these approaches are structural.

We consider that computing the prefix of a set of nodes, and computing strongly connected components (SCC) of the graph are both solved problems. The prefix of S is the least fixed point of the equation $\forall s \in S, \exists s', (s', s) \in E \Rightarrow s' \in S$. The SCC of a graph form a partition of the nodes, where for any pair of nodes (p, p') in a subset, there exists a path from p to p' and from p' to p. The construction of the prefix is trivial; decomposition into SCC can be computed in linear time with Tarjan's algorithm.

Rule 18. Free SCC
Applicability: Deadlock, Safety
Description: Consider a set of places P not in the support are linked by by elementary transitions (one input, one output). Tokens in any of these places can thus travel freely to any other place in this SCC. We can compute such SCC, and for each one replace all places in the SCC by a single "sum" place that represents it.

Definition: We build a graph that contains a node for every place in $\mathcal{P} \setminus Supp$ and an edge from p to p' iff. $\exists t \in \mathcal{T}, \bullet t = \{p\} \wedge \mathcal{W}_-(p,t) = 1, t\bullet = \{p'\} \wedge \mathcal{W}_+(p',t) = 1$.

For each SCC S of size 2 or more of this graph, we define a new place p such that: $\forall t \in \mathcal{T}, \mathcal{W}_-(p,t) = \sum_{p' \in S} \mathcal{W}_-(p',t) \wedge \mathcal{W}_+(p,t) = \sum_{p' \in S} \mathcal{W}_+(p',t)$, and $m_0(p) = \sum_{p' \in S} m_0(p')$. Then we discard all places in the SCC S.

Correctness: $\Rightarrow$: Any scenario that required to mark one or more places of the SCC is still feasible (more easily) using the "sum" place; no behavior has been removed. $\Leftarrow$: The sum place in fact represents any distribution of the tokens it contains within the places of the SCC in the original net. Because these markings were all reachable from one another in the original net, the use of the abstract "sum" place does not add any behavior.

Notes: This powerful rule is computationally cheap, provides huge reductions, and is not covered by classical pre and post agglomerations. [5] has a similar rule limited to fusing two adjacent places linked by a pair of elementary transitions. This rule (and a generalization of it) is presented using a different formalization in [2].

Rule 19. Prefix of Interest: Deadlock

Applicability: Deadlock

Description: Consider the graph that represents all potential token flows, i.e. it has an edge from every input place of a transition to each of its output places. Only SCC (lakes) in this token flow graph can lead to absence of deadlocks in the system, if the net flows has no SCC (like a river), it must eventually must lose all its tokens and deadlock. Tokens and places that are initially above (feeding streams) or in an SCC are relevant, as well as tokens that can help empty an SCC (they control floodgates). The rest of the net can simply be discarded.

Definition: We build a graph G that contains a node for every place in $\mathcal{P}$ and an edge from p to p' iff. $\exists t \in \mathcal{T}, p \in \bullet t \wedge p' \in t\bullet$

We compute the set of non trivial SCC of this graph G: SCC of size two or more, or a consisting of a single place p but only if it has a true self-loop $\exists t \in \mathcal{T}, \bullet t = t\bullet = \{p\} \wedge \mathcal{W}_-(p,t) = \mathcal{W}_+(p,t)$. We let S contain the union of places in these non trivial SCC. We add predecessors of output transitions of this set to the set, $S \leftarrow S \cup \{\bullet t \mid \exists p \in S, \exists t \in p\bullet\}$. This step not iterated. We then compute in the graph the nodes in the prefix of S add them to this set S.

We finally discard any places that do not belong to Prefix of Interest S, as well as any transition fed by such a place. Discard all places p, $p \notin S$, and transitions in $p\bullet$.

Correctness: $\Rightarrow$: The parts of the net that are removed inevitably led to a deadlock (ending in a place with no successor transitions or being consumed) for the tokens that entered them. These tokens now disappear immediately upon entering the suffix region, correctly capturing the fact this trace would eventually lead to a deadlock in the original net. So no deadlocks have been removed. $\Leftarrow$: Any scenario leading to a deadlock must now either empty the tokens in the SCC or consist in interlocking the tokens in the SCC. Such a scenario using only transitions that were preserved was already feasible in the original net, reaching a state from which a deadlock was inevitable once tokens had sufficiently progressed in the suffix of the net that was discarded.

Notes: This very powerful rule is computationally cheap and provides huge reductions. The closest work we could find in the literature was related to program slicing rather

than Petri nets. The main strength is that we ignore the structure of the discarded parts, letting us discard complex (not otherwise reducible) parts of the net. The case where S is empty because the net contains no SCC is actually relatively common in the MCC and allows to quickly conclude.

Rule 20. Prefix of Interest: Safety
Applicability: Safety
Description: Consider the graph that represents all actual token flows, i.e. it has an edge from every input place p of a transition t to each of its output places p' distinct from p, but only if t is not just reading from p'. This graph represents actual token movements and takes into account read arcs with an asymmetry. A transition consuming from p_1 to feed p_2 under the control of reading from p_3 would induce an edge from p_1 to p_2 and from p_3 to p_2, but not from p_1 to p_3. Indeed p_1 is not causally responsible for the marking in p_3 so it should not be in its prefix.

We start from places in the support of the property, which are interesting, as well as all predecessors of transitions consuming from them (these transitions are visible by definition). These places and their prefix in the graph are interesting, the rest of the net can simply be discarded.
Definition: We build a graph G that contains a node for every place in $\mathcal{P}$ and an edge from p to p' iff.

$$\exists t \in \mathcal{T}, p \in {}^\bullet t \wedge p' \in t^\bullet \wedge p \neq p' \wedge \mathcal{W}_-(p',t) \neq \mathcal{W}_+(p',t)$$

We let S contain the support of the property $S = Supp$.

We add predecessors of output transitions of this set to the set,
$S \leftarrow S \cup \{{}^\bullet t \mid \exists p \in S, \exists t \in p^\bullet\}$. This step not iterated.

We then add any place in the prefix of S to the interesting places S. We finally discard all places $p \in \mathcal{P} \setminus S$, and for each of them the transitions in $p^\bullet$.
Correctness: $\Rightarrow$: The parts of the net that are removed are necessarily stuttering effects, leading to more stuttering effects. The behavior that is discarded cannot causally influence whether a given marking of the original net projected over the support is reachable or not. Any trace of the original system projected over the transitions that remain in new net is still feasible and leads to a state having the same properties as the original net. So no observable behavior has been removed. $\Leftarrow$: Any trace of the new system is also feasible in the original net, and leads to a state satisfying the same properties as in the original net. So no behavior has been added.
Notes: This very powerful rule is computationally cheap, provides huge reductions, and is not otherwise covered in the literature. Similarly to the rule for Deadlock, it can discard complex (not otherwise reducible) parts of the net. The refinement in the graph for read arcs allows to reduce parts of the net (including SCC) that are *controlled by* the places of interest, but do not themselves actually feed or consume tokens from them.

7 SMT-backed Behavioral Reduction Rules

Leveraging the over-approximation of the state space defined in Section 4, we now define reduction rules that test behavioral application conditions using this approximation.

Rule 21. Implicit place
Applicability: Deadlock, Safety
Description: An implicit place p never restricts any transition t in the net from firing: if t is disabled it is because some other place is insufficiently marked, never because of p. Such a place is therefore not useful, and can be discarded from the net.
Definition: Implicit place: a place p is *implicit* iff. for any transition t that consumes from p, if t is otherwise enabled, then p is sufficiently marked to let t fire. Formally,

$$\forall m \in \mathcal{R}, \forall t \in p\bullet, (\forall p' \in \bullet t \setminus \{p\}, m(p') \geq \mathcal{W}_-(p',t)) \Rightarrow m(p) \geq \mathcal{W}_-(p,t)$$

To use our SMT engine to determine if a place $p \in \mathcal{P} \setminus Supp$ is assuredly implicit, we assert:

$$\exists t \in p\bullet, m_p < \mathcal{W}_-(p,t) \wedge \forall p' \in \bullet t \setminus \{p\}, m_{p'} \geq \mathcal{W}_-(p',t)$$

If the result in UNSAT, we have successfully proved p is implicit and can discard it.

Correctness: $\Rightarrow$: Removing a place cannot remove any behavior. $\Leftarrow$: Removing the place p does not add behavior since it could not actually disable any transition.
Notes: The notion of implicit place and how to structurally or behaviorally characterise them is discussed at length in [10], but appears already in [1].

We recommend to heuristically start by testing the places that have the most output transitions, as removing them has a larger impact on the net. The order is important when two (or more) places are mutually implicit, so that each of them satisfies the criterion, but they share an output transition t that only consumes from them (so both cannot be discarded).

Rule 22. Structurally Dead Transition
Applicability: Deadlock, Safety
Description: If in any reachable marking t is disabled, it can never fire and we can discard t.
Definition: For each transition $t \in \mathcal{T}$, we use our Safety procedure to try to prove the invariant "t is disabled" negatively expressed by asserting:

$$\bigwedge_{p \in \bullet t} m_p \geq \mathcal{W}_-(p,t)$$

If the result is UNSAT, we have successfully proved t is never enabled and can discard it.
Correctness: $\Rightarrow$: t was never enabled even in the over-approximation we consider, therefore discarding it does not remove any behavior. $\Leftarrow$: Removing a transition cannot add any behavior.
Notes: Because of the refined approximation of the state space we have, this test is quite strong in practice at removing otherwise reduction resistant parts of the net.

8 Evaluation

8.1 Implementation

The implementation of the algorithms described in this paper was done in Java and relies on Z3 [15] as SMT solver. The code is freely available under the terms of

Gnu GPL, and distributed from `http://ddd.lip6.fr` as part of the ITS-tools. Using sparse representations everywhere is critical; we work with transition based column sparse matrix (so preset and postset are sparse), and transpose them when working with places. The notations we used when defining the rules in this paper deliberately present immediate parallels with an efficient sparse implementation of markings and flow matrices. For instance, because we assume that $\forall p \in \bullet t$ is a sparse iteration we always prefer it to $\forall p \in \mathcal{P}$ in rule definitions.

Our random explorer is also sparse, can restart (in particular if it reaches a deadlock, but not only), is more likely to fire a transition again if it is still enabled after one firing (encouraging to fully empty places), can be configured to prefer newly enabled transitions (pseudo DFS) or a contrario transitions that have been enabled a long time (pseudo BFS). It can be guided by a Parikh firing count vector, where only transitions with positive count in the vector are (pseudo randomly) chosen and counts decremented after each firing. For deadlock detection, it can also be configured to prefer successor states that have the least enabled events. These various heuristics are necessary as some states are exponentially unlikely to be reached by a pure random memory-less explorer. Variety in these heuristics where each one has a strong bias in one direction is thus desirable. After each restart we switch heuristic for the next run.

The setting of Section 2 is rich enough to capture the problems given in the Model Checking Contest in the *Deadlock*, *ReachabilityFireability* and *ReachabilityCardinality* examinations. We translate fireability of a transition t to the state based predicate $m \geq \mathcal{W}_-(t)$. We also negate reachability properties where appropriate so that all properties are positive invariants that must hold on all states. To perform a reduction of a net and a set of safety properties, we iterate the following steps, simplifying properties and net as we progress:

1. We perform a random run to see if we can visit any counter-example markings within a time bound.
2. We perform structural reductions preserving the union of the support of remaining properties.
3. We try to prove that the remaining properties hold using the SMT based procedure.
4. If properties remain, we now have a candidate Parikh vector for each of them that we try to pseudo-randomly replay to contradict the properties that remain.
5. If the computation has not progressed yet in this iteration, we apply the more costly SMT based structural reduction rules (see Sec. 7)
6. If the problem has still not progressed in this iteration, we do a refined analysis to simplify atoms of the properties, reducing their support
7. As long as at least one step in this process has made progress, and there remain properties to be checked, we iterate the procedure.

Step 6 is trying to prove for every atomic proposition in every remaining property that the atom is invariant: its value in all states is the same as in the initial state. Any atom thus proved to be constant can then be replaced by its value in the initial state to simplify the properties and their support. This procedure is also applicable to arbitrary temporal logic formulas, as shown in [4]. We can thus in some cases even solve CTL and LTL logic formulas by reducing some of their atomic propositions to true or false.

8.2 Experimental Validation

We used the MCC2019 models and formulas, limiting ourselves to examinations where all formulas were solved in 2019. All the formulas solved by our tool agree with the control values from the contest [2]. An examination consists in a model instance and either 16 safety predicates (cardinality or fireability) or a single deadlock detection task. Model instances come from 90 distinct families of Petri nets, some of which features colors.

For deadlock detection, the approach was able to fully solve 902 (536 true, 366 false) out of 932 deadlock problem instances (96.8%), where true means a deadlock was found. For safety properties, we fully solved 1634 out 1748 examinations (93.5%), and in total reduced 27594 out of 27968 formulas to true or false (98.6%).

We limited our experiments to 12 minutes of runtime and $8GB$ of RAM. We feel this is a reasonable timeout for a filter in front of an exhaustive model-checker since the contest gives 1 hour per examination. 21 of the 2680 examinations timed out; the total runtime was $82k$ seconds thus averaging at 31 seconds per examination overall.

Of the 28496 formulas solved, the solutions were due to pseudo-random exploration or Parikh guided exploration in 17829(62%) of formulas, to structural reductions and immediate simplification for 4720(17%) formulas and the SMT procedure proved 5947(21%) formulas in total. These statistics reveal a bias in the benchmark in favor of invariants that can be disproved by a counter-example.

The reduction rules presented in this paper are all relevant on this benchmark for more than one model, and combine on top of each other to achieve superior reductions.

The problems that are not fully solved often have small state spaces on which invariants do hold, but which the SMT constraints fail to prove and our memory-less explorer cannot prove. The SMT solver is particularly useful; besides proving properties to be true, in many cases the reduction becomes stuck until SMT can prove some places to be implicit, which starts another round of reductions. The random memory-less walker also benefits hugely from reductions since they make it increasingly likely that we can observe the target situation within a reasonable number of steps.

9 Conclusion

The approach presented in this paper combines over-approximation using an SMT solver, under-approximation by sampling with a random walk, and works with a system that is progressively simplified by property preserving structural reduction rules.

Structural approaches are a strong class of techniques to analyse Petri nets, that bypass state space explosion in many cases. Structural reductions are particularly appealing because any gain in structural complexity usually implies an exponential state space reduction. The approach presented in this paper is complementary of other verification strategies as the behavior for the given properties is preserved by the transformations: it can act as an elaborate filter in front of any verification tool.

[2] The raw logs and procedure to reproduce the experiment are available on `https://lip6.github.io/ITSTools-web/structural.html` as well as graphically rendered examples.

The choice of using an SMT based solver rather than a more classical ILP engine gives us flexibility and versatility, so that extending the refinement with new constraints is relatively easy. Our current directions include some new partial agglomeration rules where only some transitions stutter, investigating other more complete ways of replaying a Parikh candidate such as the approach proposed in [17], and extending our set of reduction rules to cover more fully the new advanced rules presented in [2].

References

1. Berthelot, G.: Checking properties of nets using transformation. In: Applications and Theory in Petri Nets. Lecture Notes in Computer Science, vol. 222, pp. 19–40. Springer (1985)
2. Berthomieu, B., Le Botlan, D., Dal Zilio, S.: Counting Petri net markings from reduction equations. International Journal on Software Tools for Technology Transfer (Apr 2019)
3. Best, E., Schlachter, U.: Analysis of Petri nets and transition systems. In: ICE. EPTCS, vol. 189, pp. 53–67 (2015)
4. Bønneland, F., Dyhr, J., Jensen, P.G., Johannsen, M., Srba, J.: Simplification of CTL formulae for efficient model checking of Petri nets. In: Petri Nets. Lecture Notes in Computer Science, vol. 10877, pp. 143–163. Springer (2018)
5. Bønneland, F.M., Dyhr, J., Jensen, P.G., Johannsen, M., Srba, J.: Stubborn versus structural reductions for Petri nets. J. Log. Algebr. Meth. Program. **102**, 46–63 (2019)
6. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: CAV. LNCS, vol. 1855, pp. 154–169. Springer (2000)
7. D'Anna, M., Trigila, S.: Concurrent system analysis using Petri nets: an optimized algorithm for finding net invariants. Computer Communications **11**(4), 215–220 (1988)
8. Esparza, J., Melzer, S.: Verification of safety properties using integer programming: Beyond the state equation. Formal Methods in System Design **16**(2), 159–189 (2000)
9. Evangelista, S., Haddad, S., Pradat-Peyre, J.: Syntactical colored Petri nets reductions. In: ATVA. Lecture Notes in Computer Science, vol. 3707, pp. 202–216. Springer (2005)
10. García-Vallés, F., Colom, J.M.: Implicit places in net systems. In: PNPM. pp. 104–113. IEEE Computer Society (1999)
11. Haddad, S., Pradat-Peyre, J.: New efficient Petri nets reductions for parallel programs verification. Parallel Processing Letters **16**(1), 101–116 (2006)
12. Laarman, A.: Stubborn transaction reduction. In: NFM. Lecture Notes in Computer Science, vol. 10811, pp. 280–298. Springer (2018)
13. Lipton, R.J.: Reduction: A method of proving properties of parallel programs. Commun. ACM **18**(12), 717–721 (1975)
14. Liu, G., Barkaoui, K.: A survey of siphons in Petri nets. Inf. Sci. **363**, 198–220 (2016)
15. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: TACAS. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008)
16. Murata, T.: State equation, controllability, and maximal matchings of Petri nets. IEEE Transactions on Automatic Control **22**, 412–416 (1977)
17. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. In: TACAS. Lecture Notes in Computer Science, vol. 6605, pp. 224–238. Springer (2011)