New Search Strategies for the Petri Net CEGAR Approach

Ákos Hajdu[1], András Vörös[1], and Tamás Bartha[2]

[1] Department of Measurement and Information Systems
Budapest University of Technology and Economics, Budapest, Hungary
`vori@mit.bme.hu`
[2] Institute for Computer Science and Control
MTA SZTAKI, Budapest, Hungary

Abstract. Petri nets are a successful formal method for the modeling and verification of asynchronous, concurrent and distributed systems. Reachability analysis can provide important information about the behavior of the model. However, reachability analysis is a computationally hard problem, especially when the state space is infinite. Abstraction-based techniques are often applied to overcome complexity. In this paper we analyze an algorithm, which uses counterexample guided abstraction refinement. This algorithm proved its efficiency on the model checking contest. We examine the algorithm from a theoretical and practical point of view. On the theoretical side, we show that the algorithm cannot decide reachability for relatively simple instances. We propose a new iteration strategy to explore the invariant space, which extends the set of decidable problems. We also give proofs on the theoretical limits of our approach. On the practical side, we examine different search strategies and we present our new, complex strategy with superior performance compared to traditional strategies. Measurements show that our new contributions perform well for traditional benchmark models as well.

Keywords: Petri nets, reachability analysis, abstraction, CEGAR, ILP

1 Introduction

The development of complex, distributed and safety-critical systems requires mathematically precise proofs in order to ensure the suitability and correctness of the design. Formal modeling and verification methods provide such tools. However, a major drawback of using formal techniques is their computation and memory-intensive nature. Even for relatively small asynchronous and concurrent models, the state space and the set of possible behaviors can be unmanageably large, or even infinite. This is usually referred to as the "state space explosion" problem in the literature.

This problem also holds for one of the most popular modeling formalisms, Petri nets. The behavior of a Petri net model is determined by the set of reachable states and fireable transitions. Therefore, reachability analysis is an important formal verification technique for Petri nets. The reachability problem

Authors' manuscript. Published in R. Devillers, A. Valmari (eds.): *Application and Theory of Petri Nets and Concurrency*, LNCS 9115, 2015. The final publication is available at Springer via `http://dx.doi.org/10.1007/978-3-319-19488-2_16`.

answers the question whether a given state is reachable from the initial state of the modeled system. However, solving reachability is a computationally hard problem. Therefore, abstraction-based techniques are often involved to overcome complexity.

Wimmel and Wolf published an algorithm [18], which applies counterexample guided abstraction refinement to the reachability problem of Petri nets. Their algorithm proved its efficiency at the model checking contest in 2013 [10]. After its publication, we analyzed the algorithm regarding correctness and completeness, and published our results in [8]. Although the algorithm can solve many problems efficiently, we proved that it fails to decide reachability for relatively simple instances. In worse cases it may even give a wrong answer. We suggested improvements and we also extended the algorithm to be able to handle inhibitor arcs and submarking coverability problems. Furthermore, we proved that even the improved algorithm is incomplete due to its iteration strategy.

In this paper we continue our work with further theoretical and practical investigations. In Section 2 we introduce the theoretical background of our work. We present the algorithm of Wimmel and Wolf [18] and a brief overview of our previous findings [8] in Section 3. Then, we introduce our current results. On the theoretical side, we propose a new iteration strategy to be used during the phase that explores the invariant space (Section 4). We show that our new approach extends the set of decidable problems and we also give theoretical results on its limits. On the practical side, we examine the behavior of well-known search strategies (depth- and breadth-first search) for the solution space traversal and we also present our new, complex strategy combining the advantages of BFS and DFS (Section 5). We prove the efficiency of our new approaches with measurements on traditional benchmark models and on our special nets as well (Section 6). Finally, we conclude our work in Section 7.

2 Background

In this section we introduce the theoretical background of our work. First, we present Petri nets (Section 2.1), then we introduce reachability analysis (Section 2.2).

2.1 Petri Nets

Petri nets [13] are graphical models for concurrent and asynchronous systems, providing both structural and dynamical analysis. A discrete Petri net is a tuple $PN = (P, T, E, W)$, where P is the set of *places*, T is the set of *transitions*, with $P \neq \emptyset \neq T$ and $P \cap T = \emptyset$, $E \subseteq (P \times T) \cup (T \times P)$ is the set of *arcs* and $W\colon E \mapsto \mathbb{Z}^+$ is the weight function assigning weights $w^-(p_j, t_i)$ to the edge $(p_j, t_i) \in E$ and $w^+(p_j, t_i)$ to the edge $(t_i, p_j) \in E$. Places and transitions are numbered from zero in our work.

A *marking* of a Petri net is a mapping $m\colon P \mapsto \mathbb{N}$. If a place p contains k *tokens* in a marking m then $m(p) = k$. The initial marking is denoted by m_0.

Dynamic Behavior. A transition $t \in T$ is *enabled* in a marking m, if $m(p_j) \geq w^-(p_j, t)$ holds for each $p_j \in P$ with $(p_j, t) \in E$. An enabled transition t can *fire*, consuming $w^-(p_j, t)$ tokens from places $p_j \in P$ with $(p_j, t) \in E$ and producing $w^+(p_j, t)$ tokens on places $p_j \in P$ with $(t, p_j) \in E$. The firing of a transition t in a marking m is denoted by $m[t\rangle m'$ where m' is the marking after firing t.

A word $\sigma = t_1 t_2 \dots t_n \in T^*$ is a *firing sequence*. A firing sequence is *realizable* in a marking m and leads to m' (denoted by $m[\sigma\rangle m'$), if $m[t_1\rangle \dots [t_n\rangle m'$. The *Parikh image* of a firing sequence σ is a vector $\wp(\sigma)\colon T \mapsto \mathbb{N}$, where $\wp(\sigma)(t_i)$ is the number of the occurrences of t_i in σ. The empty firing sequence is denoted by ε.

2.2 Reachability Problem

A marking m' is *reachable* from m if a realizable firing sequence $\sigma \in T^*$ exists for which $m[\sigma\rangle m'$ holds. The set of all reachable markings from the initial marking m_0 of a Petri net PN is denoted by $R(PN, m_0)$. The reachability problem is to decide if $m' \in R(PN, m_0)$ holds for a given marking m'. The aim of reachability analysis is to solve the reachability problem by finding a realizable firing sequence $m_0[\sigma\rangle m'$. The reachability problem is decidable [12], but it is at least EXPSPACE-hard [11] and no upper bound is known yet.

State Equation. The *incidence matrix* of a Petri net is a matrix $C_{|P|\times|T|}$, where $C(i, j) = w^+(p_i, t_j) - w^-(p_i, t_j)$. The element $C(i, j)$ represents the change in the number of tokens in p_i after firing t_j. Let m and m' be markings of the Petri net, then the *state equation* takes the form $m + Cx = m'$. Any vector $x \in \mathbb{N}^{|T|}$ fulfilling the state equation is called a *solution*. Note, that for any realizable firing sequence σ leading from m to m', the Parikh image of the firing sequence fulfills the equation $m + C\wp(\sigma) = m'$. On the other hand, not all solutions of the state equation are Parikh images of a realizable firing sequence. Therefore, the existence of a solution for the state equation is a necessary but not sufficient criterion for reachability. A solution x is called *realizable* if a realizable firing sequence σ exists with $\wp(\sigma) = x$.

T-invariants. A vector $y \in \mathbb{N}^{|T|}$ is called a *T-invariant* if $Cy = 0$ holds. A realizable T-invariant represents the possibility of a cyclic behavior in the modeled system, since its complete occurrence does not change the marking. However, during firing the transitions of the T-invariant, some intermediate markings can be of interest. If each component of the T-invariant y is either zero or one we also denote y by enumerating the components with value one, e.g., $y = (1, 0, 1, 0)$ can be denoted by $y = \{t_0, t_2\}$.

Solution Space. The solution space of the state equation $m + Cx = m'$ is semi-linear. Each solution x can be written as the sum of a *base solution* and the linear combination of T-invariants [18], which can formally be written as

$x = b + \sum_i n_i y_i$, where $b \in \mathbb{N}^{|T|}$ is the base solution and $n_i \in \mathbb{N}$ is the coefficient of the T-invariant $y_i \in \mathbb{N}^{|T|}$.

3 CEGAR Approach on Petri Nets

In this section we introduce the CEGAR approach generally (Section 3.1) and we present an algorithm published by Wimmel and Wolf [18], which applies the CEGAR approach to the reachability problem of Petri nets (Section 3.2). After its publication, we examined the correctness and completeness of their algorithm [8]. These findings form a basis for our current work, so we introduce them briefly in Section 3.3.

3.1 CEGAR Approach

Abstraction is a general mathematical approach for solving hard problems. It hides the irrelevant details, so the abstract model can be handled easier. One such technique is existential abstraction [5], which means that the abstract model over-approximates the original one. Therefore, if an invariant holds in the abstract model, it also holds in the original model. However, if there is a counterexample for which the invariant does not hold, it might be caused by the over-approximation. Thus, every counterexample must be examined whether it has a corresponding concrete counterexample in the original model. If a concrete counterexample exists, the invariant does not hold in the original model. Otherwise, the abstract counterexample is spurious and the abstraction has to be refined using the information from the examination. This technique is called the “counterexample guided abstraction refinement” (CEGAR) and it is widely used in model checking [1], [4], [9].

3.2 Reachability Analysis of Petri Nets Using CEGAR

Wimmel and Wolf published an algorithm [18], which applies the CEGAR approach to the reachability analysis of Petri nets, using the state equation. Figure 1 shows an overview of their algorithm, while each step is detailed in this section.

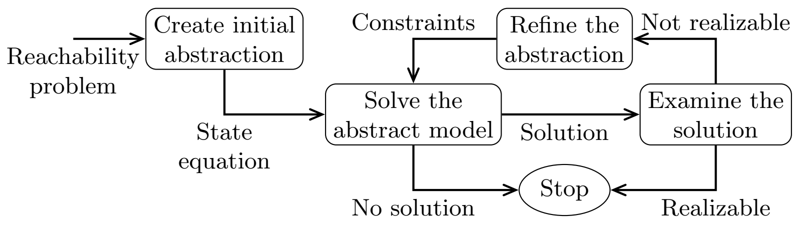


Fig. 1. Petri net CEGAR algorithm flowchart

Initial Abstraction. The input of the algorithm is a reachability problem $m' \in R(PN, m_0)$, which is transformed into the initial abstraction, namely the state equation of the form $m_0 + Cx = m'$.

Solving the Abstract Model. Solving the abstract model (i.e., the state equation) is an integer linear programming problem [6]. The ILP solver yields a minimal solution with respect to the cost function. In the algorithm of Wimmel and Wolf [18], the sum of the firing count of transitions is minimized in order to obtain trajectories with the shortest length.

The state equation is an over-approximation of the set of reachable markings, since its feasibility is a necessary, but not sufficient condition for reachability. Therefore, if no abstract solution exists, the target marking cannot be reached in the Petri net either. However, a solution of the abstract model may or may not be realizable by a firing sequence. Thus, further examinations are needed.

Examining the Solution. The solution of the state equation is a vector $x \in \mathbb{N}^{|T|}$, where $x(t)$ denotes the number of times a transition $t \in T$ has to fire in order to reach m' from m_0. However, x does not include any information about the order of the transition firings and whether they are enabled. Thus, the algorithm has to explore the state space of the Petri net with the limitation that each transition t can fire at most $x(t)$ times. If the target marking m' can be reached with this limit (i.e., x is realizable), it is a sufficient proof for reachability. Otherwise, x is a counterexample and the abstraction has to be refined.

Abstraction Refinement. If a solution x is not realizable, the ILP solver has to be forced to generate a different solution. This can be done by adding additional *constraints* (i.e., linear inequalities over transitions) to the state equation. The following two types of constraints were defined by Wimmel and Wolf [18].

– *Jump constraints* have the form $|t_i| < n$, where $n \in \mathbb{N}$, $t_i \in T$ and $|t_i|$ represents the firing count of the transition t_i. Jump constraints can be used to obtain different base solutions, exploiting their pairwise incomparability.
– *Increment constraints* have the form $\sum_{i=1}^{k} n_i |t_i| \geq n$, where $n_i \in \mathbb{Z}$, $n \in \mathbb{N}$, and $t_i \in T$. Increment constraints can be used to reach non-base solutions, i.e., T-invariants are added in some linear combination.

After adding the new constraint, the state equation may become infeasible, or a new solution is obtained. Figure 2 presents the solution space. The bottom dots represent base solutions, while the cones represent the linear space formed by the T-invariants. The upper dots correspond to non-base solutions. Jumps are denoted by dashed arrows and increments by continuous arrows. The precise method for generating constraints and traversing the solution space is presented later in this section, but first, *partial solutions* are introduced.

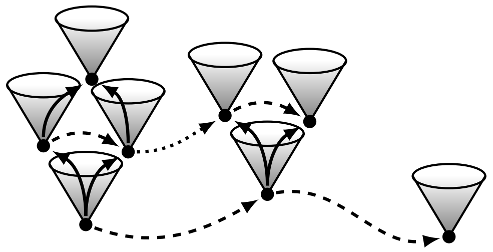


Fig. 2. Solution space of the state equation [18]

Partial Solutions. Given a Petri net $PN = (P, T, E, W)$ and a reachability problem $m' \in R(PN, m_0)$, a *partial solution* is a tuple $ps = (\mathcal{C}, x, \sigma, r)$, where:

- $\mathcal{C}$ is the set of (jump and increment) constraints, together with the state equation they define the ILP problem,
- x is the minimal solution satisfying the state equation and the constraints belonging to the set $\mathcal{C}$,
- $\sigma \in T^*$ is a maximal realizable firing sequence, with $\wp(\sigma) \leq x$, i.e., each transition $t \in T$ can fire at most $x(t)$ times and enabled transitions must fire in some order,
- $r = x - \wp(\sigma)$ is the remainder vector.

Partial solutions are generated during the examination of the solution x by exploring the state space of the Petri net. For this purpose, Wimmel and Wolf use a "brute force" method with some optimization. The algorithm builds a tree with markings as nodes and occurrences of transitions as edges. The root of the tree is the initial marking m_0, and there is an edge labeled by t between nodes m_1 and m_2 if $m_1[t\rangle m_2$ holds. On each path leading from the root of the tree to a leaf, each transition t_i can occur at most $x(t_i)$ times. Each path to a leaf represents a maximal firing sequence, thus a new partial solution. The marking reached is referred to as the *final marking* of the partial solution.

A partial solution is called a *full solution* if $r = 0$ holds, thus $\wp(\sigma) = x$, which means that σ realizes the solution vector x. Wimmel and Wolf proved that for each realizable solution of the state equation a full solution exists. This full solution can be reached by continuously expanding the minimal solution of the state equation with constraints [18].

Consider now a partial solution $ps = (\mathcal{C}, x, \sigma, r)$, which is not a full solution, i.e., $r \neq 0$. This means that some transitions could not fire enough times. There are three possible situations in this case:

1. x may be realizable by another firing sequence σ', thus a full solution $ps' = (\mathcal{C}, x, \sigma', 0)$ can be found in the tree.
2. By adding jump constraints, greater, but pairwise incomparable solutions can be obtained.

3. For transitions $t \in T$ with $r(t) > 0$ increment constraints can be added to increase the token count in the input places of t, while the final marking m' must be unchanged. This can be achieved by adding new T-invariants to the solution. These T-invariants can "borrow" tokens for transitions in the remainder vector.

Generating Constraints. When a partial solution is not a full solution, both jump and increment constraints can be added, but they are applied on a different level:

– Jump constraints are generated from solution vectors of the state equation.
– Increment constraints are generated from partial solutions (which are obtained from solution vectors).

Jump Constraints. Given a solution vector x, for each transition $t_i \in T$ with $x(t_i) > 0$ a jump constraint c_i of the form $|t_i| < x(t_i)$ can be added to the state equation. If a new solution vector y_i is obtained after adding one of the constraints c_i, this process can be recursively repeated for y_i. Wimmel and Wolf proved that every base solution can be obtained using jump constraints [18].

Increment Constraints. Let $ps = (\mathcal{C}, x, \sigma, r)$ be a partial solution with $r > 0$. This means that some transitions could not fire enough times. Wimmel and Wolf use a heuristic to find the places and number of tokens needed to enable these transitions. If a set of places actually needs n ($n > 0$) tokens, the heuristic estimates a number from 1 to n. If the estimate is too low, this method can be applied again, converging to the actual number of required tokens. The heuristic consists of the following three steps:

1. First, it builds a dependency graph to collect the transitions and places that are of interest. These are transitions that could not fire, and places that disable these transitions under the final marking of ps. An edge from a place p to a transition t means that p disables t, while an edge in the opposite direction means that firing t would increase the token count in p. Each source SCC[3] of the dependency graph has to be investigated, because it cannot get tokens from other components. Therefore, an increment constraint is needed.
2. The second step is to calculate the minimal number of missing tokens for each source SCC. There are two sets of transitions, $T_i \subseteq T$ and $X_i \subseteq T$. If one transition in T_i becomes fireable, it may enable all the other transitions of the SCC, while transitions in X_i cannot activate each other, therefore their token shortage must be fulfilled at once.
3. The third step is to construct an increment constraint c for each source SCC from the information about the places and their token requirements. These constraints will force transitions (with $r(t) = 0$) to produce tokens in the given places. Since the final marking is left unchanged, a T-invariant is added to the solution vector.

[3] Source strongly connected component, i.e., one without incoming edges from other components.

When applying the new constraint c, three situations are possible depending on the T-invariants in the Petri net:

– If the state equation and the set of constraints become infeasible, this partial solution cannot be extended to a full solution, therefore it is no longer of interest.
– If the ILP solver can produce a solution $x + y$ (with y being a T-invariant), new partial solutions can be found for y. If none of them helps getting closer to a full solution, the algorithm can get into an infinite loop, but no full solution is lost. A method to avoid this non-termination phenomenon will be discussed later in this section.
– If there is a new partial solution ps' where some transitions in the remainder vector could fire, this method can be repeated.

The following theorem of Wimmel and Wolf [18] states that if the reachability problem has a solution, it can be reached by the CEGAR approach:

Theorem 1. *If the reachability problem has a solution, a realizable solution of the state equation can be reached by continuously expanding the minimal solution with jump and increment constraints.*

Optimizations. Wimmel and Wolf also presented some methods for optimization [18]. In our current work, only the following T-invariant filtering optimization is important. After adding a T-invariant y to the partial solution $ps = (\mathcal{C}, x, \sigma, r)$, all the transitions of y may fire without enabling any transition in r, yielding a partial solution $ps' = (\mathcal{C}', x + y, \sigma', r)$ with $\wp(\sigma') = \wp(\sigma) + y$. The final marking and remainder vector of ps' is the same as in ps, therefore the same T-invariant y is added to the solution by the heuristic again, which can prevent termination. Thus, the algorithm cuts the search space at ps'. However, during firing the transitions of y, the algorithm could get closer to enabling a transition in r (without reaching the limit where it becomes enabled). These "better" intermediate markings should be detected, and be used as new partial solutions. Wimmel and Wolf gave a definition for better intermediate markings, which we generalized it in our former work [8]. Our definition is as follows.

Definition 1 (Better intermediate marking). *An intermediate marking m_i is considered better than the final marking m' of the firing sequence σ if there exists a transition t with $r(t) > 0$ and a place p with $(p, t) \in E$ for which $m'(p) < w^-(p, t) \wedge m_i(p) > m'(p)$ holds.*

This means that t is disabled by p and p had more tokens in the intermediate marking m_i than in the final marking m'.

3.3 Correctness and Completeness of the Algorithm

After Wimmel and Wolf published their algorithm, we examined the correctness and completeness properties and we published our findings in [8]. This section summarizes these results.

Correctness. We proved by a counterexample that the algorithm is incorrect due to an over-estimation in the increment constraint generating heuristic. In this case, incorrectness resulted in an answer "not reachable" for a reachable marking. We suggested a method to detect such situations giving the answer "not decidable". We also presented a new algorithm that tries to find the solution in such cases.

Completeness. We presented several subclasses of Petri nets for which the algorithm could not decide reachability and we suggested solutions to most of them. However, we proved that the improved algorithm is still incomplete due to its iteration strategy. In our current work we present a similar, but simpler proof (Section 4.1) and we propose a new iteration strategy to extend the set of decidable problems (Section 4.2).

4 New Iteration Strategy to Explore the Invariant Space

In this section we show that the algorithm of Wimmel and Wolf cannot decide reachability for relatively simple examples, because not every necessary invariant is explored (Section 4.1). We propose a new iteration strategy to traverse the invariant space by involving so-called "distant" invariants (Section 4.2). We show that this new approach extends the set of decidable problems and we also give theoretical results on its limitations. We also present a new filtering criterion (Section 4.3), which can avoid non-termination of the algorithm.

4.1 Proof of the Incompleteness

We prove the incompleteness of the algorithm published by Wimmel and Wolf [18] with the following example. Consider the Petri net PN in Figure 3 with the reachability problem $(1,1,0) \in R(PN,(0,1,0))$, i.e., producing a token in p_0. The vector $x_s = (1,1,1,1,1)$ is a solution, realized by the firing sequence $\sigma_s = t_3 t_1 t_0 t_2 t_4$.

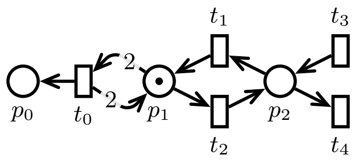


Fig. 3. A counterexample of completeness

The algorithm does the following steps. The minimal solution vector is $x_0 = (1,0,0,0,0)$, i.e., firing t_0. Since t_0 is not enabled, the only partial solution is $ps_0 = (\emptyset, x_0, \sigma_0 = \varepsilon, r_0 = (1,0,0,0,0))$. The algorithm finds that an additional token is required in p_1 and only t_1 can satisfy this need. With an increment constraint $c_1\colon |t_1| \geq 1$, the T-invariant $\{t_1, t_2\}$ is added to the new solution vector $x_1 = (1, 1, 1, 0, 0)$. Only t_2 and t_1 can fire (in this order), thus the only partial solution for x_1 is $ps_1 = (\{c_1\}, x_1, \sigma_1 = t_2 t_1, r_1 = r_0)$. This partial solution is skipped by the T-invariant filtering optimization, since the only difference from ps_0 is that all transitions of a T-invariant were fired. Furthermore, there are no better intermediate markings, since no additional token was "borrowed" from the T-invariant $\{t_1, t_2\}$. The algorithm terminates at this point, leaving the problem undecided. Without the filtering optimization, the algorithm would add the T-invariant $\{t_1, t_2\}$ again and again, preventing termination.

The problem is that the original algorithm does not recognize that although $\{t_1, t_2\}$ can fire, it only circulates the same token, instead of "lending" a new one. An extra token could be produced in p_2 (and then moved in p_1) using the T-invariant $\{t_3, t_4\}$. However, $\{t_3, t_4\}$ is not connected directly to p_1 (where the tokens are missing), so the iteration strategy of the algorithm does not try to involve it. We propose an extension to the iteration strategy in Section 4.2 in order to involve such "distant" invariants into the solution vector.

4.2 Involving Distant Invariants

Let y and z be T-invariants. We say that z is a distant invariant for y if z can produce tokens in a place connected to y. This can be written formally as follows.

Definition 2 (Distant invariant). *The T-invariant z is a distant invariant for the T-invariant y if a place p and transitions t_1, t_2 exist with $y(t_1) > 0$, $z(t_2) > 0$, $((t_1, p) \in E \vee (p, t_1) \in E)$, $w^+(p, t_2) - w^-(p, t_2) > 0$ and $y(t_2) = 0$.*

The definition states that y includes t_1, z includes t_2 and t_1 is connected to p, where the firing of t_2 increases the number of tokens. This way z can "borrow" tokens for y. The extra criterion $y(t_2) = 0$ is needed to ensure that we do not produce tokens for y by itself. In the example in Figure 3, $\{t_3, t_4\}$ is a distant invariant for $\{t_1, t_2\}$ because t_3 can produce tokens in p_2, which is connected to t_1 (and t_2).

When a transition in the remainder could not fire, the original algorithm tried to increase the token count on its input places. Our definition of distant invariants generalizes this concept the following way. When a partial solution is skipped by the T-invariant filtering optimization, it means that a T-invariant was fired, but could not "lend" enough tokens to enable a transition in the remainder. The basic idea of involving distant invariants is to try to increase the token count in any place connected to the filtered T-invariant. If some tokens can be produced, the filtered invariant will then be able to transfer them indirectly to the place that lacks tokens. There are two problems to be solved:

- How many tokens should be produced for the invariant that caused filtering?
- Termination criterion: if the distant invariant cannot help, adding it again can lead to non-termination.

Number of Tokens Produced in the Invariant. Estimating the required number of tokens is a hard problem, since the sum of the tokens in the places of a T-invariant may change during firing. Over-estimation can also be a problem: the final marking of the invariant may not be the “best” state regarding the number of tokens. Therefore, we produce only one token at a time and repeat this process if it was not enough.

Termination Criterion. When a distant invariant does not help, there are two possible cases. The distant invariant z could either not lend any tokens to the filtered invariant y or it could lend some, but not enough to enable a transition in the remainder.

The first case means that not only y lacks tokens, but z as well. Thus, we can now apply our strategy again, i.e., involving a distant invariant for $y + z$. This way we form a “chain” of distant invariants, which is defined formally as follows.

Definition 3 (Chain of distant invariants). *Let $y_1, y_2, \ldots, y_n$ ($n \in \mathbb{N}$) be T-invariants. We say that $y_1 + y_2 + \ldots + y_n$, $n \in \mathbb{N}$ is a chain of distant invariants if y_{i+1} is a distant invariant for y_i (for $1 \leq i < n$). A subchain of a chain $y_1 + y_2 + \ldots + y_n$ is a chain $y_1 + y_2 + \ldots + y_k$, with $k \leq n$.*

The definition of distant invariants ensures termination for such chains, since the newly involved distant invariant must have at least one transition that is not included in the previous ones and the number of transitions in a Petri net is finite.

The second case indicates that z could lend some tokens, but not enough. Therefore, we can involve distant invariants again for y. If z is the only distant invariant for y, this simply results in adding z again, but in general any distant invariant can be involved. However, if $y = y_1 + y_2 + \ldots + y_n$ is a chain, this would only produce tokens in places connected to y_n. Thus, we have to involve a distant invariant for every subchain in order to transfer the tokens to the originally filtered invariant (y_1).

Our new ideas above are formulated in Algorithm 1. The input of the algorithm is a partial solution ps' that was skipped due to ps and the number of better intermediate markings during the firing sequence of ps'. Partial solutions are extended to store a chain of distant invariants, which is initially 0.

At first we calculate the difference between the solution vectors of ps and ps' and we initialize the list of constraints with the constraints of ps'. The following two cases are possible.

– If the chain of $ps \neq 0$, some distant invariants were already involved. If there are better intermediate markings ($n_b > 0$), then these invariants helped (but not enough) to enable a transition in the remainder. In this case we can involve them again, so the chain of ps' is the same as in ps and we involve a distant invariant for every subchain.

Algorithm 1: Distant invariant algorithm

```
   Input  : $ps'$: Partial solution skipped
            $ps$: Partial solution that caused skipping $ps'$
            $n_b$: Number of better intermediate markings for $ps'$
   Output : $x$: New solution vector found by involving distant invariants
 1 $z \leftarrow$ difference invariant between $ps$ and $ps'$ ;
 2 $\mathcal{C}^* \leftarrow$ constraints of $ps'$;
 3 if the chain of $ps \neq 0 \ \wedge n_b > 0$ then
 4     Chain of $ps' \leftarrow$ Chain of $ps$;
 5     for each subchain of $ps'$ do
 6         $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup$ {constraint to involve a distant invariant for the subchain};
 7     end
 8 end
 9 else if $z$ is a distant invariant for the chain of $ps$ then
10     Chain of $ps' \leftarrow$ Chain of $ps + z$;
11     $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup$ {constraint to involve a distant invariant for the chain of $ps'$};
12 end
13 $x \leftarrow$ solve the state equation with $\mathcal{C}^*$;
14 return $x$;
```

– Otherwise we extend the chain of ps with z and involve distant invariants only for the whole chain. However, we have to first check if z is really an extension to the chain of ps, since ps' can be a solution obtained by the original increment constraints.

Finding a constraint to involve a distant invariant for a chain (or subchain) y is quite straightforward. We get the places connected to the transitions of y and we create a constraint using the third step of the increment constraint generating heuristic to produce a token in these places. If no constraint can be found, the algorithm returns no new solution. If there are multiple distant invariants for y, all of them can be found using jump constraints from the original algorithm. Finally, we solve the state equation extended with $\mathcal{C}^*$ and return the solution (if found).

This new strategy can solve the example in Figure 3 trivially. As a complex example, consider the Petri net PN in Figure 4 with the reachability problem $(1, 1, 0, 0, 2) \in R(PN, (0, 1, 0, 0, 2))$, i.e., producing a token in p_0.

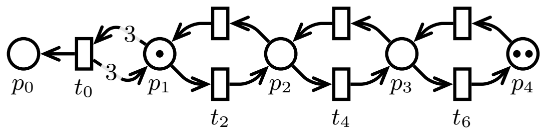
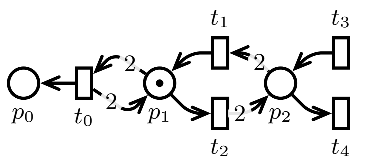


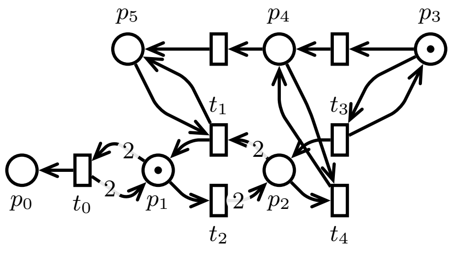
Fig. 4. Distant invariant example

The minimal solution of the abstract model is firing t_0, which is not enabled. Thus, the T-invariant $\{t_1, t_2\}$ is added twice in order to get two additional tokens in p_1. This invariant can fire but it does not help getting closer to enabling t_0 so the partial solution is skipped. At this point, our new algorithm tries to produce a token in any of the places connected to $\{t_1, t_2\}$, i.e., p_1 and p_2 by distant invariants. Therefore, the T-invariant $\{t_3, t_4\}$ is added once to the new solution. This invariant can also fire but does not help enabling t_0. The partial solution is skipped, and since $\{t_3, t_4\}$ is a distant invariant for $\{t_1, t_2\}$, the algorithm now tries to produce a token in places connected to the chain $\{t_1, t_2\} \cup \{t_3, t_4\}$, i.e., in p_1, p_2, and p_3. This implies that the invariant $\{t_5, t_6\}$ is added once. Firing this invariant does not enable t_0, but yields an extra token in p_1, which is a better intermediate marking. Thus, the partial solution is skipped but the algorithm now tries to involve distant invariants for every subchain, namely for $\{t_1, t_2\}$ and $\{t_1, t_2, t_3, t_4\}$, resulting in the addition of $\{t_3, t_4\}$ and $\{t_5, t_6\}$. The solution vector is now $(1, 2, 2, 2, 2, 2, 2)$, which can be realized by the firing sequence $t_5 t_5 t_3 t_3 t_1 t_1 t_0 t_2 t_2 t_4 t_4 t_6 t_6$.

Limitations. Although our new approach can solve a new range of problems, it also has some limitations. As an example consider the Petri net PN in Figure 5(a) with the reachability problem $(1, 1, 0) \in R(PN, (0, 1, 0))$, i.e., producing a token in p_0.



(a) Not decidable example



(b) Example on non-termination

Fig. 5. Example nets for the limitation of distant invariants

The minimal solution is firing t_0, which is not enabled. Thus, the T-invariant $\{t_1, t_2\}$ is added once in order to get an additional token in p_1. This invariant can fire, but it does not help getting closer to enabling t_0 so the partial solution is filtered. At this point the algorithm tries to produce tokens for $\{t_1, t_2\}$ using distant invariants, which implies adding $\{t_3, t_4\}$ once. This invariant can fire, lending a token in p_2. However, t_1 requires two tokens to fire and produce one in p_1. This partial solution is also filtered and there are no better intermediate markings, since we only count the tokens in places connected to the disabled transition t_0, which is p_1. The algorithm terminates at this point leaving the problem undecided.

A trivial idea for this example would be to extend the definition of better intermediate markings (Definition 1) to count tokens not only in places connected to the transition that cannot fire, but in places connected to the filtered T-invariant as well. This can be formalized as follows. Let $ps = (\mathcal{C}, x + y, \sigma, r)$ be a partial solution that was skipped due to the invariant y. Suppose that we obtained $ps' = (\mathcal{C}', x + y + z, \sigma', r)$ by involving the distant invariant z for y, which could not enable any transition in the remainder, thus ps' is skipped as well. Furthermore, suppose that no better intermediate marking was found using Definition 1 (as in the example in Figure 5(a)). Given a partial solution ps and a place p let $\max(ps, p)$ be $\max(m(p))$ during firing σ of ps from the initial marking m_0. Then the definition of better intermediate markings can be generalized in the following way.

Definition 4. *Given the partial solutions ps and ps' as described above, an intermediate marking m_i of σ' is considered better than the final marking m' if Definition 1 holds or a transition t with $y(t) > 0$ and a place p with $(p, t) \in E \vee (t, p) \in E$ exists for which $m_i(p) > \max(ps, p)$ holds.*

The generalized definition states that the intermediate marking is also considered better if there is a place connected to the filtered T-invariant, which contains more tokens than in any marking in the firing sequence of the previous partial solution. If a better intermediate marking exists for ps' using this definition, then we can involve z again. However, this definition would often lead to non-termination since the filtered T-invariant (y) is already enabled (otherwise it would not have been filtered). Thus, we cannot give an upper bound on the number of tokens in p, as opposed to our original definition, where we produce tokens in p until the transition that is disabled by p gets enabled.

As an example consider the Petri net PN in Figure 5(b) with the reachability problem $(1, 1, 0, 0, 0, 1) \in R(PN, (0, 1, 0, 1, 0, 0))$, i.e., producing a token in p_0 and moving the token from p_3 to p_5. This net works similarly to the net in Figure 5(a), but occurrences of the transitions t_3, t_4, and t_1 can only appear in this order, due to the upper part (places p_3, p_4, p_5) of the net. As in the previous example, $\{t_1, t_2\}$ is added first, then $\{t_3, t_4\}$. Suppose now, that we consider it a better intermediate marking when t_3 produced a token in p_2. This implies that $\{t_3, t_4\}$ is added again. Now t_4 can fire two times, producing two tokens in p_2. There are two possible sequels. If t_1 fires, it produces an extra token in p_1 and enables t_0. However, the extra tokens must be consumed in order to reach the final marking, but t_4 cannot fire after t_1. The search terminates on this path, since no more solutions can be found. The second case is that t_4 fires, which consumes the tokens from p_2 so t_1 cannot transfer them to p_1. Thus, t_0 is still not enabled, but we had a better intermediate state, since we had two tokens in p_2. Therefore, $\{t_3, t_4\}$ is added again and this process repeats avoiding termination.

The examples in Figure 5 show that the generalized definition (Definition 4) may help to decide reachability for some instances, but it may also yield non-termination.

4.3 New Filtering Criterion

Although a partial solution is skipped using the T-invariant filtering optimization, we may obtain new solutions from it through intermediate markings or distant invariants. This yields a new branch in the search space, which can also lead to non-termination.

There are special cases where T-invariants can either fire or not, both being a maximal firing sequence. As an example, consider the Petri net in Figure 4 and suppose that t_1, t_2, t_3, and t_4 each has to fire once. A possible maximal firing sequence is $t_2 t_4 t_3 t_1$, but $t_2 t_1$ is also maximal, since neither t_4 nor t_3 is enabled afterwards. When such invariants exist, it is possible that the following two partial solutions are obtained from $ps = (\mathcal{C}, x, \sigma, r)$ after adding the invariant y:

- $ps' = (\mathcal{C}', x + y, \sigma', r)$, with $\wp(\sigma') = \wp(\sigma) + y$, and
- $ps'' = (\mathcal{C}', x + y, \sigma, r + y)$.

In the first case, the invariant was fired (i.e., added to the firing sequence), while in the second case it was not fired (i.e., added to the remainder). The first case can be detected by the T-invariant filtering optimization. However, we found that the second case can also lead to non-termination if there are at least two T-invariants with this property.

To overcome this problem, we detect when a T-invariant is added to the remainder, i.e., we get $ps'' = (\mathcal{C}', x + y, \sigma, r + y)$ from $ps = (\mathcal{C}, x, \sigma, r)$. However, ps'' cannot be filtered immediately because the remainder is different so the abstraction refinement may add new invariants that can help. We only skip ps'' if ps was skipped by the original T-invariant filtering optimization, which also means that ps'' was obtained through intermediate markings or distant invariants.

5 Search Strategies

As already mentioned in Section 3.2, the algorithm of Wimmel and Wolf traverses the semi-linear solution space of the state equation. At each non-realizable solution, multiple (jump and/or increment) constraints can be applied, each yielding a new path in the solution space. However, the authors did not publish the strategy for the solution space traversal in [18]. An overview pseudo-code was published later in [19]. In this section we present three different search strategies: depth-first search (Section 5.1), breadth-first search (Section 5.2) and our new approach, a complex strategy (Section 5.3), which combines the advantages of DFS and BFS. Measurement results supporting our statements in this section can be found in Section 6.2.

5.1 Depth-First Search

Depth-first search (DFS) can be very effective regarding memory usage and computation time as well. It only stores one path of the solution space in memory at a time for backtracking purposes and it has a fast convergence if several invariants have to be added to reach a realizable solution. However, DFS has some disadvantages as well:

– It may not find the minimal solution by choosing a path, which contains a solution but not the minimal one.
– It may fail to terminate in an infinite solution space by choosing a path, where T-invariants can be added infinitely many times without finding a realizable solution.

The T-invariant filtering optimization (Section 3.2) and our new filtering criterion (Section 4.3) cuts the search space, but does not always detect infinite loops. We tried to give stronger criteria for cutting, but then realizable solutions were lost, reducing the set of decidable problems.

5.2 Breadth-First Search

Due to the problems of DFS, we implemented a breadth-first search (BFS) version of the algorithm as well. The number of base solutions can grow exponentially, but it is always finite so we still use DFS between the base solutions and only use BFS in the linear space of invariants. As opposed to DFS, it is less efficient, but always finds the minimal solution if the target marking is reachable. When the target marking is not reachable, BFS may fail to terminate in an infinite solution space. The T-invariant filtering optimization can prevent this in some cases and can also make the computational time shorter.

5.3 Complex Search

We also developed a new, complex search strategy, which combines the advantages of DFS and BFS. We traverse the base solutions using DFS as previously. When exploring the invariant space over a base solution our main strategy is DFS, but with a little BFS extension: at each solution x, we generate all partial solutions belonging to x, instead of continuing the search with the first one and filter them based on a partial order.

Ordering of Partial Solutions. We define an ordering over vectors and partial solutions as follows.

Definition 5 (Ordering of vectors). *A vector x is less than a vector y (denoted by $x < y$), if and only if $x(i) \leq y(i)$ for each index i and $x \neq y$.*

Definition 6 (Ordering of partial solutions). *A partial solution $ps_1 = (\mathcal{C}, x, \sigma_1, r_1)$ is less than a partial solution $ps_2 = (\mathcal{C}, x, \sigma_2, r_2)$ (denoted by $ps_1 < ps_2$), if and only if $r_2 < r_1$.*

A partial solution ps_1 is less than a partial solution ps_2 if the remainder r_2 is less than r_1. This means that ps_2 is closer to realization, since every transition fired in the sequence of ps_1 was also fired in ps_2, but ps_2 may have more fired transitions. Note that this is a partial order, since partial solutions ps_1, ps_2 may exist with $ps_1 \not< ps_2$ and $ps_2 \not< ps_1$, e.g., if $r_1 = (1, 0)$ and $r_2 = (0, 1)$.

Filtering Partial Solutions. For our filtering criterion we define maximal and minimal partial solutions.

Definition 7 (Maximal partial solution). *A partial solution ps of a solution x is maximal, if and only if no other partial solution ps' exists for x with $ps < ps'$.*

Definition 8 (Minimal partial solution). *A partial solution ps of a solution x is minimal, if and only if no other partial solution ps' exists for x with $ps' < ps$.*

The filtering criterion is quite simple, we only keep minimal and maximal partial solutions. Since the ordering is partial, there can be more than one minimal and maximal partial solutions.

We keep the maximal partial solution because it has a minimal remainder, i.e., it is the closest to realizing the solution vector. Also, the T-invariant filtering optimization works well for maximal partial solutions, since every T-invariant that can fire, must also fire (i.e., it is added to the firing sequence). A minimal partial solution has maximal remainder, i.e., not every enabled T-invariant was fired. This yields a slower convergence to a realizable solution. However, since the remainder is different from the remainder of the maximal partial solution, the abstraction refinement may involve different invariants.

6 Evaluation

We implemented our algorithm as a plug-in for the *PetriDotNet* [15] framework to evaluate its performance. We compared our approach to other tools and algorithms (Section 6.1) and we also measured the performance of the different search strategies (Section 6.2).

6.1 Comparison to Other Tools and Algorithms

We compared our algorithm to the implementation of Wimmel and Wolf, which is called the *SARA* tool [17]. We also compared our approach to the well-known saturation-based model checking algorithm [2], [14]. The results can be seen in Table 1, where *TO* refers to an unacceptable run-time (> 600 seconds), *ERR* means a run-time exception and *NS* implies that the algorithm terminated, but could not solve the problem.

The FMS model [3] represents a flexible manufacturing system. The parameter of the model determines the size of the state space, while the structure of the net is fixed. The results show that our algorithm outperforms both saturation and the SARA tool. The Kanban model [3] illustrates a production scheduling method. The parameter determines the size of the state space. We experienced that our algorithm can find a realizable solution quickly, but it examines many partial solutions before finding the full solution. The Dining philosophers model [7] is often used to show the problems of parallel programming and mutual exclusion. As the parameter grows, both the structure of the net and the state space becomes larger. Saturation and SARA performs better for these models.

Table 1. Comparison of our algorithm to SARA and saturation

Model	**Our algorithm**	**SARA**	**Saturation**
FMS-10	0,041 s	0,001 s	0,06 s
FMS-50	0,048 s	0,018 s	1,09 s
FMS-100	0,056 s	0,059 s	8,03 s
FMS-200	0,071 s	0,278 s	69,7 s
FMS-400	0,105 s	0,868 s	TO
FMS-800	0,226 s	3,537 s	TO
FMS-1600	0,317 s	ERR	TO
FMS-3200	0,65 s	ERR	TO
FMS-6400	1,274 s	ERR	TO
FMS-12800	2,54 s	ERR	TO
Kanban-10	0,032 s	0,03 s	0,002 s
Kanban-13	1,074 s	0,05 s	0,003 s
Kanban-16	3,055 s	0,09 s	0,01 s
Kanban-19	7,128 s	0,134 s	0,03 s
Kanban-22	16,039 s	0,2 s	0,03 s
Kanban-25	31,181 s	0,268 s	0,05 s
Dphil-10	0,078 s	0,005 s	0,01 s
Dphil-20	0,204 s	0,012 s	0,02 s
Dphil-30	0,399 s	0,021 s	0,03 s
Dphil-50	1,156 s	0,037 s	0,03 s
Dphil-100	6,989 s	0,094 s	0,04 s
Dphil-200	67,603 s	0,33 s	0,05 s
Distant1	0,027 s	0,001 s	-
Distant2	0,068 s	NS	-
Distant3	0,083 s	NS	-
Distant4	0,116 s	NS	-
Distant5	0,078 s	NS	-
Distant6	0,063 s	NS	-
Distant7	0,137 s	NS	-

The Distant models are built by us [16] to test our new iteration strategy, which involves distant invariants. The Distant1 and Distant3 models can also be seen in Fig. 3 and Fig. 4. After publishing our former proof of incompleteness [8], we contacted Wimmel and Wolf and they extended their implementation to be able to solve Distant1. However, the original algorithm cannot solve complex examples on distant invariants. As the state space of these models are infinite, saturation cannot handle these problems.

Due to the complexity of the models, further examination is required to determine how the structure and behavior of the models affect the performance of the algorithms and which algorithm is the most effective for a given type of models. This is an interesting future research direction.

6.2 Comparison of Search Strategies

The solution space (i.e., the abstract model) is usually small for the examples presented in Table 1, so every search strategy has a similar performance. We created models with many T-invariants (i.e., a large solution space) to evaluate the different search strategies. The results can be seen in Table 2, where the cost corresponds to the size of the solution, i.e., $\sum_{t \in T} x(t)$. The two parameters in the model name determine the number of invariants. The asterisk indicates a different ordering of places and transitions.

Table 2. Measurement results for different search strategies

	DFS		**BFS**		**Complex**	
Model	**Time**	**Cost**	**Time**	**Cost**	**Time**	**Cost**
Chain 1+2	0,04 s	7	0,055 s	7	0,039 s	7
Chain 1+3	0,095 s	13	0,828 s	13	0,1 s	13
Chain 1+4	0,291 s	21	85,24 s	21	0,288 s	21
Chain 1+4*	24,2 s	35	55,28 s	21	1,498 s	29
Chain 1+5	54,59 s	39	TO	31	56,36 s	39
Chain 2+2	0,076 s	11	0,277 s	11	0,074 s	11
Chain 2+3	0,197 s	19	12,768 s	19	0,288 s	23
Chain 2+3*	2,28 s	29	5,288 s	19	1,387 s	23

It is clear that DFS is more efficient than BFS regarding computational time. However, it often fails to find the minimal solution. Our combined strategy often outperforms DFS, while also being closer to the minimal solution.

7 Conclusions

In our paper we examined an abstraction-based algorithm for the reachability problem of Petri nets. From the theoretical point of view, we showed that the original algorithm cannot decide reachability for relatively simple nets. We presented a new iteration strategy based on distant invariants in order to overcome this deficiency. We also gave theoretical results on the limits of our new approach. From the practical point of view, we examined the behavior of the solution space traversal with DFS and BFS strategies and we also proposed a new, complex strategy based on a partial order between solutions. We demonstrated the efficiency of our new approaches with measurements.

Acknowledgement. This work was partially supported by the ARTEMIS JU and the Hungarian National Research, Development and Innovation Fund in the frame of the R5-COP (Reconfigurable ROS-based Resilient Reasoning Robotic Cooperating Systems) project.

References

1. Beyer, D., Henzinger, T., Jhala, R., Majumdar, R.: The software model checker Blast. International Journal on Software Tools for Technology Transfer 9(5-6), 505–525 (2007)
2. Ciardo, G., Lüttgen, G., Siminiceanu, R.: Saturation: An efficient iteration strategy for symbolic state-space generation. In: Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 328–342. Springer-Verlag (2001)
3. Ciardo, G., Zhao, Y., Jin, X.: Ten years of saturation: A Petri net perspective. In: Jensen, K., Donatelli, S., Kleijn, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency V, Lecture Notes in Computer Science, vol. 6900, pp. 51–95. Springer Berlin Heidelberg (2012)
4. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
5. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. 16(5), 1512–1542 (1994)
6. Dantzig, G.B., Thapa, M.N.: Linear programming 1: introduction. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1997)
7. Dijkstra, E.: Hierarchical ordering of sequential processes. Acta Informatica 1(2), 115–138 (1971)
8. Hajdu, Á., Vörös, A., Tamás, B., Mártonka, Z.: Extensions to the CEGAR approach on Petri nets. Acta Cybernetica 21(3), 401–417 (2014)
9. John, A., Konnov, I., Schmid, U., Veith, H., Widder, J.: Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In: Formal Methods in Computer-Aided Design (FMCAD), 2013. pp. 201–209 (Oct 2013)
10. Kordon, F., Linard, A., Becutti, M., Buchs, D., Fronc, L., Hulin-Hubard, F., Legond-Aubry, F., Lohmann, N., Marechal, A., Paviot-Adet, E., Pommereau, F., Rodrígues, C., Rohr, C., Thierry-Mieg, Y., Wimmel, H., Wolf, K.: Web report on the model checking contest @ Petri net 2013, available at http://mcc.lip6.fr (June 2013)
11. Lipton, R.: The Reachability Problem Requires Exponential Space. Research report, Yale University, Dept. of Computer Science (1976)
12. Mayr, E.W.: An algorithm for the general Petri net reachability problem. In: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing. pp. 238–246. STOC '81, ACM, New York, NY, USA (1981)
13. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (April 1989)
14. Vörös, A., Darvas, D., Bartha, T.: Bounded Saturation Based CTL Model Checking. In: Proceedings of the 12th Symposium on Programming Languages and Software Tools, SPLST'11 (2011)
15. Website of PetriDotNet, `http://inf.mit.bme.hu/en/research/tools/petridotnet` [Online, accessed: 22/03/2015]
16. Website of the models used in the measurements, `http://inf.mit.bme.hu/en/pn2015` [Online, accessed: 22/03/2015]
17. Website of the SARA tool, `http://www.service-technology.org/sara/index.html` [Online, accessed: 22/03/2015]
18. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. In: Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 224–238. Springer (2011)
19. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. Logical Methods in Computer Science 8(3) (2012)