Extensions to the CEGAR Approach on Petri Nets[⋆]

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

[1] Dept. of Measurement and Information Systems
Budapest University of Technology and Economics, Budapest, Hungary
`vori@mit.bme.hu`
[2] Computer and Automation Research Institute
MTA SZTAKI,
Budapest, Hungary

Abstract. Formal verification is becoming more prevalent and often compulsory in the safety-critical system and software development processes. Reachability analysis can provide information about safety and invariant properties of the developed system. However, checking the reachability is a computationally hard problem, especially in the case of asynchronous or infinite state systems. Petri nets are widely used for the modeling and verification of such systems. In this paper we examine a recently published approach for the reachability checking of Petri net markings. We give proofs concerning the completeness and the correctness properties of the algorithm, and we introduce algorithmic improvements. We also extend the algorithm to handle new classes of problems: submarking coverability and reachability of Petri nets with inhibitor arcs.

1 Introduction

The development of complex, distributed systems, and safety-critical systems in particular, require mathematically precise verification techniques in order to prove the suitability and faultlessness of the design. Formal modeling and analysis methods provide such tools. However, one of the major drawbacks of formal methods is their computation and memory-intensive nature: even for relatively simple distributed, asynchronous systems the state space and the set of possible behaviors can become unmanageably large and complex, or even infinite.

This problem also appears in one of the most popular modeling formalisms, Petri nets. Petri nets have a simple structure, which makes it possible to use strong structural analysis techniques based on the so-called *state equation.* As structural analysis is independent of the initial state, it can handle even infinite state problems. Unfortunately, its pertinence to practical problems, such

[⋆] This work was partially supported by the European Union and the European Social Fund through the project FuturICT.hu (grant no. TAMOP-4.2.2.C-11/1/KONV-2012-0013) of VIKING Zrt Balatonfured.

as reachability analysis, has been limited. Recently, a new algorithm [12] using Counter-Example Guided Abstraction Refinement (CEGAR) extended the applicability of state equation based reachability analysis.

Our paper improves this new algorithm in several important ways. The authors of the original CEGAR algorithm have not published proofs for the completeness of their algorithm and the correctness of a heuristic used in the algorithm. In this paper we analyze the correctness and completeness of their work as well as our extensions. We prove the lack of correctness in certain situations by a counterexample, and provide corrections to overcome this problem. We also prove that the algorithm is incomplete, due to its iteration strategy. We describe algorithmic improvements that extend the set of decidable problems, and that effectively reduce the search space. We extend the applicability of the approach even further: we provide solutions to handle Petri nets with inhibitor arcs, and the so-called *submarking coverability problem.* At the end of our paper we demonstrate the efficiency of our improvements by measurements.

2 Background

In this section we introduce the background of our work. First, we present Petri nets (Section 2.1) as the modeling formalism used in our work. Section 2.2 introduces the counterexample guided abstraction refinement method and its application for the Petri net reachability problem.

2.1 Petri nets

Petri nets are graphical models for concurrent and asynchronous systems, providing both structural and dynamical analysis. A discrete ordinary Petri net is a tuple $PN = (P, T, E, W)$, where P is the set of *places*, T is the set of *transitions*, with $P \neq T \neq \emptyset$ and $P \cap T = \emptyset$, $E \subseteq (P \times T) \cup (T \times P)$ is the set of *arcs* and $W : E \to \mathbb{Z}^+$ 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 $(t_i, p_j) \in E$ [9].

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

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

A word $\sigma \in T^*$ is a *firing sequence*. A firing sequence is *realizable* in a marking m and leads to m', $m[\sigma\rangle m'$, if either $m = m'$ and σ is an empty word, or there exists a $w \in T^*$ realizable firing sequence, a $t_i \in T$, and an m'' such that $m[w\rangle m''[t_i\rangle m'$. The *Parikh image* of a firing sequence σ is a vector $\wp(\sigma) : T \to \mathbb{N}$, where $\wp(\sigma)(t_i)$ is the number of the occurrences of t_i in σ.

Petri nets can be extended with *inhibitor arcs* to become a tuple $PN_I = (PN, I)$, where $I \subseteq (P \times T)$ is the set of inhibitor arcs. There is an extra condition for a transition $t_i \in T$ with inhibitor arcs to be enabled: for each $p_j \in P$, if $(p_j, t_i) \in I$, then $m(p_j) = 0$ must hold. A Petri net extended with inhibitor arcs is *Turing complete.*

Reachability problem. A marking m' is *reachable* from m if there exists a realizable firing sequence $\sigma \in T^*$, 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 aim of the *reachability problem* is to check if $m' \in R(PN, m_0)$ holds for a given marking m'.

We define a *predicate* as a linear inequality on markings of the form $Am \geq b$, where A is a matrix and b is a vector of coefficients [6]. The aim of the *submarking coverability problem* is to find a reachable marking $m' \in R(PN, m_0)$ for which a given predicate $Am' \geq b$ holds.

The reachability problem is decidable [8], but it is at least EXPSPACE-hard [7]. Using inhibitor arcs, the reachability problem in general is undecidable [3].

State equation. The *incidence matrix* of a Petri net is a matrix $C_{|P|\times|T|}$, where $C(i,j) = w^+(p_i, t_j) - w^-(p_i, t_j)$. 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 the reachability. A solution x is called *realizable* if there exists a realizable firing sequence σ, with $\wp(\sigma) = x$.

T-invariants. A vector $x \in \mathbb{N}^{|T|}$ is called a *T-invariant* if $Cx = 0$ 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 interesting for us later.

Solution space. Each solution x of the state equation $m + Cx = m'$, can be written as the sum of a *base vector* and the linear combination of T-invariants [12], which can formally be written as $x = b + \sum_i n_i y_i$, where b is the base vector and n_i is the coefficient of the T-invariant y_i.

2.2 The CEGAR approach

The counterexample guided abstraction refinement (CEGAR) is a general approach for analyzing systems with large or infinite state space. The CEGAR method works on an abstraction of the original model, which has fewer restrictions. During the iteration steps, the CEGAR method refines the abstraction using the information from the explored part of the state space. When applying CEGAR on the Petri net reachability problem [12], the initial abstraction is the state equation. Solving the state equation is an integer linear programming problem [5], for which the ILP solver tool can yield one solution, minimizing a target function of the variables. Since the algorithm seeks the shortest firing sequences leading to the target marking, it minimizes the function $f(x) = \sum_{t \in T} x(t)$. When solving the ILP problem, the following situations are possible:

– If the state equation is infeasible, the necessary criterion does not hold, thus the target marking is not reachable.
– If the state equation has a realizable solution, the target marking is reachable.
– If the state equation has an unrealizable solution, it is a counterexample and the abstraction has to be refined.

The purpose of the abstraction refinement is to exclude counterexamples from the solution space, without losing any realizable solution. For this purpose, the CEGAR approach uses linear inequalities over transitions, called *constraints*.

Constraints. Two types of constraints were defined by Wimmel and Wolf [12]:

– *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 switch between base vectors, 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.

Partial solutions. For a given Petri net $PN = (P, T, E, W)$ and a reachability problem $m' \in R(PN, m_0)$, a *partial solution* is a tuple $(\mathcal{C}, x, \sigma, r)$, 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 in $\mathcal{C}$,
– $\sigma \in T^*$ is a maximal realizable firing sequence, with $\wp(\sigma) \leq x$, i.e., each transition can fire as many times as it is included in the solution vector x,
– $r = x - \wp(\sigma)$ is the remainder vector.

Generating partial solutions. Partial solutions can be produced from a solution vector x (and a constraint set $\mathcal{C}$) by firing as many transitions as possible. For this purpose, the algorithm uses a “brute force” method. 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. Even though the tree can be traversed only storing one path in the memory at a time using depth first search, the size of the tree can grow exponentially. Some optimizations are presented later in this section to reduce the size of the tree.

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. For each realizable solution x of the solution space there exists a full solution [12]. This full solution can be reached by continuously expanding the minimal solution of the state equation with constraints.

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 $ps' = (\mathcal{C}, x, \sigma', r)$ exists.
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 on 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 jump constraints. Each base vector of the solution space can be reached by continuously adding jump constraints to the minimal solution [12]. In order to reach non-base solutions, increment constraints are needed, but they might conflict with previous jump constraints. Jump constraints are only needed to obtain a different base solution vector. However, after the computation of the base solution, jump constraints can be transformed into equivalent increment constraints ([12]).

Generating increment constraints. Let $ps = (\mathcal{C}, x, \sigma, r)$ be a partial solution with $r > 0$. This means that some transitions (in r) could not fire enough times. The algorithm uses 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, the algorithm builds a dependency graph [10] to get the transitions and places that are of interest. These are transitions that could not fire, and places which disable these transitions. Each source SCC[3] of the dependency graph has to be investigated, because it cannot get tokens from another components. Therefore, an increment constraint is needed.

[3] Strongly connected component

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.

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 can be skipped.
- If the ILP solver can produce a solution $x + y$ (with y being a T-invariant), new partial solutions can be found. If none of them help getting closer to the 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 below.
- If there is a new partial solution ps' where some transitions in the remainder vector could fire, this method can be continued.

Theorem 1. *(Reachability of solutions) [12] If the reachability problem has a solution, a realizable solution of the state equation can be reached by continuously adding constraints, transforming jumps before increments.*

Optimizations. Wimmel and Wolf [12] presented also some methods for optimization. The following are important for our work:

- **Stubborn set** The stubborn set method [10] investigates conflicts, concurrency and dependencies between transitions, and reduces the search space by filtering the transitions: stubborn set method usually leads to a search tree with lower degree.
- **Subtree omission** When a transition has to fire more than once ($x(t) > 1$), the stubborn set method does not provide efficient reduction. The same marking is often reached by firing sequences which only differ in the order of transitions. During the abstraction refinement, only the final marking of the firing sequence is important. If a marking m' is reached by firing the same transitions as in a previous path, but in a different order, the subtree after m' was already processed. Therefore, it is no longer of interest.
- **Filtering T-invariants** 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 $ps' = (\mathcal{C}', x + y, \sigma', r)$. 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 vector again, which can prevent the

algorithm from terminating. However, during firing the transitions of y, the algorithm could get closer to enabling a transition in r. These intermediate markings should be detected, and be used as new partial solutions.

3 Theoretical results

In this section we present our theoretical results with regard to the correctness and completeness of the original algorithm.

3.1 Correctness

Although Theorem 1 states that a realizable solution can be reached using constraints, we found out that in some special cases the heuristic used for generating increment constraints can overestimate the required number of tokens for proving reachability. We prove the incorrectness by a counterexample, for which the original algorithm [12] gives an incorrect answer.

Consider the Petri net in Figure 1 with the reachability problem $(0,1,0,0,1,0,0,2) \in R(PN,(1,0,0,0,0,0,0,2))$, i.e., we want to move the token from p_0 to p_1 and p_4. The example was constructed so that the target marking is reachable by the firing sequence $\sigma_m = (t_1,t_2,t_0,t_5,t_6,t_3,t_7,t_4)$, realizing the solution vector $x_m = (1,1,1,1,1,1,1,1)$.

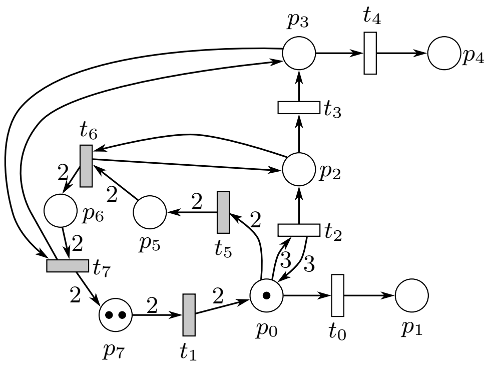

Fig. 1. Counterexample for correctness.

The CEGAR algorithm does the following steps. First, it finds the minimal solution vector $x = (1,0,1,1,1,0,0,0)$, i.e., it tries to fire the transitions t_0,t_2,t_3,t_4. From these transitions only t_0 is enabled, therefore the only partial solution is $ps = (\emptyset, x, \sigma = (t_0), r = (0,0,1,1,1,0,0,0))$. At this point the algorithm looks for an increment constraint. The dependency graph contains transitions t_2,t_3,t_4 (since they could not fire) and places p_0,p_2,p_3 (because they disable the previous transitions). The only source SCC is the set containing one place p_0 with zero tokens (because t_0 has consumed one token from there). The algorithm estimates that three tokens are needed in place p_0, where only transition t_1 can produce tokens. Therefore, the T-invariant t_1, t_5, t_6, t_7 is added twice to the solution vector. This invariant is constructed so that for each of its firing, a token has to be produced in places p_2, p_3, p_4, which token can no longer be removed. In the target marking only one token can be present on these places, therefore the algorithm cannot find the solution for the reachability problem.

Notice that the problem is the over-estimation of tokens required at p_0. Without forcing t_0 to fire, the algorithm could get a better estimation. This would imply that the invariant t_1, t_5, t_6, t_7 is added only once to the solution vector, producing the realizable solution x_m. The problem is that the algorithm always tries to find maximal firing sequences, though some transitions would not be practical to fire (t_0 in the example above). Due to this, the estimated number of tokens needed in the final marking of the firing sequence may not be correct.

Solution. Our improved algorithm counts the maximal number of tokens in each place during the firing sequence of the partial solutions into a vector m_{max}. If the final marking is not the maximal regarding a SCC, the algorithm might have over-estimated the required number of tokens. This can be detected by ordering the intermediate markings. Formally: an over-estimation can occur if a place p exists in a SCC, for which $m_{max}(p) > m'(p)$ holds, where m' is the final marking of the firing sequence.

3.2 Completeness

To our best knowledge, the completeness of the algorithm has neither been proved nor disproved yet. When we examined the iteration strategy of the abstraction loop, we found a whole subclass of nets, which cannot be solved with this strategy. As an example, consider the Petri net in Figure 2 with the reachability problem $(1, 1, 0, 0) \in R(PN, (0, 1, 0, 0))$, i.e., we want to produce a token in p_0. We constructed the net so that the firing sequence $\sigma = (t_1, t_4, t_2, t_3, t_3, t_0, t_1, t_2, t_5)$ solves the problem. The main concept of this example is that we lend an extra token on p_1 indirectly using the T-invariant t_4, t_5.

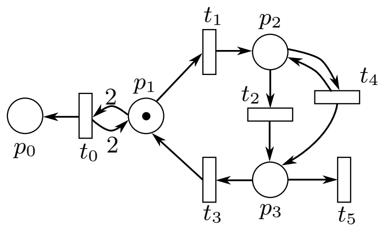

Fig. 2. Counterexample of completeness.

When applying the algorithm on this problem, the minimal solution vector is $x_0 = (1,0,0,0,0,0)$, i.e., firing t_0. Since t_0 is not enabled, the only partial solution is $ps_0 = (\emptyset, x_0, \sigma_0 = (), r_0 = (1,0,0,0,0,0))$. The algorithm finds that an additional token is required in p_1, and only t_3 can satisfy this need. With an increment constraint $c_1 : |t_3| \geq 1$, the T-invariant t_1, t_2, t_3 is added to the new solution vector $x_1 = (1,1,1,1,0,0)$, giving us one partial solution $ps_1 = (c_1, x_1, \sigma_1 = (t_1, t_2, t_3), r_1 = r_0)$. Firing the T-invariant t_1, t_2, t_3 does not help getting closer to enabling t_0, since no extra token can be "borrowed" from the previous T-invariant. The iteration strategy of the original algorithm does not recognize the fact that an extra token could be produced in p_3 (using t_4) and then moved in p_1, therefore it can not decide reachability.

4 Algorithmic contributions

In this section we present our algorithmic contributions. In Section 4.1 we show some classes of problems for which the original algorithm cannot decide reachability, and our improved algorithm solves these problems. In Section 4.2 we present two extensions of the algorithm, solving submarking coverability problems and handling Petri nets with inhibitor arcs.

4.1 Improvements

In the previous section we proved that the algorithm is not complete, but during our work we found some opportunities to extend the set of decidable problems. Moreover, we developed a new termination criterion which we prove to be correct, i.e., no realizable solution is lost using this criterion.

Total ordering of intermediate markings. When a partial solution $ps = (\mathcal{C}, x, \sigma, r)$ is skipped using the T-invariant filtering optimization, the original algorithm checks if it was closer to firing a transition t in the remainder during the firing sequence σ. This is done by "counting the minimal number of missing tokens for firing t in the intermediate markings occurring"[12]. We found out that this criterion is not general enough: in some cases the total number of missing tokens may not be less, but they are missing from different places, where additional tokens can be produced. In our new approach, we use the following definition:

Definition 1. *An intermediate marking m_i is considered better than the final marking m', if there is a transition $t \in T, r(t) > 0$ and place p with $(p,t) \in E$ for which the following criterion holds:*

$$m'(p) < w^-(p,t) \quad \wedge \quad m_i(p) > m'(p). \tag{1}$$

The left inequality in the expression means that in the final marking t is disabled by the insufficient amount of tokens in p. This condition is important, because we do not want to have more tokens on places, that already have enough to enable t. The right inequality means that p has more tokens in the intermediate marking m_i compared to the final marking m'.

Theorem 2. *Definition 1 is a total ordering of the intermediate markings occurring in the firing sequence of a partial solution.*

Proof. We first show that Definition 1 includes the original ordering of the intermediate markings. When the original criterion holds, the total number of missing tokens for enabling t at the marking m_i is less than at m'. This means that at least one place p must exist, which disables t, but $m_i(p) > m'(p)$, therefore (1) must hold. Furthermore, Definition 1 also recognizes markings which are pairwise incomparable, because if there is at least one place p with lesser tokens missing, (1) holds.

Corollary 1. *The total ordering of intermediate markings extends the set of decidable problems.*

Definition 1 is more general than the original criterion, hence it does not reduce the set of decidable problems. On the other hand, we give an example when the original criteria prevents the algorithm from finding the solution. Consider the Petri net in Figure 3 with the reachability problem $(1, 0, 0, 1) \in R(PN, (0, 1, 0, 1))$, i.e., moving one token from p_1 to p_0. The minimal solution vector is $x_0 = (1, 0, 0, 0, 0)$, i.e., firing t_0, which is disabled by p_2, therefore the only partial solution is $ps_0 = (\emptyset, x, \sigma_0 = (), r_0 = (1, 0, 0, 0, 0))$. The algorithm looks for increment constraints and finds that only t_1 can produce tokens on p_2. Consequently, the T-invariant t_1, t_2 is added to the solution vector $x_1 = (1, 1, 1, 0, 0)$. There is one partial solution $ps_1 = (\{|t_1| \geq 1\}, x_1, \sigma_1 = (t_1, t_2), r_1 = (1, 0, 0, 0, 0))$ for x_1, where the T-invariant is fired, but t_0 still could not fire. This partial solution is skipped by the T-invariant filtering optimization, and in all of the intermediate markings of σ_1, totally one token is missing from the input places of t_0. By using the original criterion, the algorithm terminates, leaving the problem as undecided. By using Definition 1 after firing t_1, less tokens are missing from p_2 than in the final marking. Continuing from here, t_0 is disabled by p_1, where t_3 can produce tokens, therefore the T-invariant t_3, t_4 is added to the new solution vector $x_2 = (1, 1, 1, 1, 1)$. A full solution is found for x_2 by the realizable firing sequence $\sigma_2 = (t_1, t_3, t_0, t_2, t_4)$.

T-invariant filtering and subtree omission. Using T-invariant filtering and subtree omission optimizations together can prevent the algorithm from finding full solutions. The order of transitions in the firing sequence of a partial solution does not matter, except in one case. When a partial solution is skipped, the algorithm checks for intermediate markings where it was closer to firing a transition in the remainder vector. By using subtree omission, intermediate markings can get lost.

As an example consider the Petri net in Figure 4 with the reachability problem $(1, 0, 0, 0, 3) \in R(PN, (0, 0, 0, 0, 3))$, i.e., we want to produce a token on p_0.

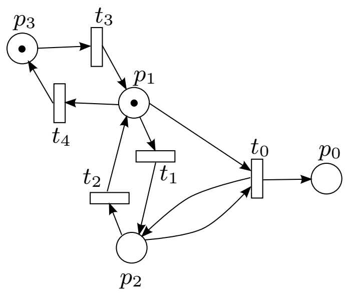

Fig. 3. Example net depicting the usefulness of the total ordering

A possible solution is the vector $x_m = (1, 1, 1, 2, 2, 3, 3)$ realized by the firing sequence $\sigma_m = (t_6, t_6, t_6, t_4, t_4, t_2, t_0, t_1, t_3, t_3, t_5, t_5, t_5)$.

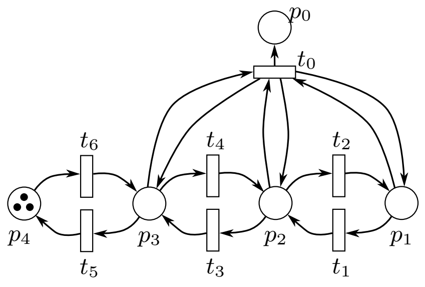

Fig. 4. An example where the order of transitions matter.

Here we present only the interesting points during the execution of the algorithm. As a minimal solution, the algorithm tries to fire t_0, but it is disabled by the places p_1, p_2, p_3. The algorithm searches for increment constraints. All the three places are in different SCCs, so the algorithm first tries to enable t_0 by borrowing one token for all three places. By the T-invariant $t_1, t_2, \ldots, t_6$ a token is carried through places p_1, p_2, p_3, which does not enable t_0, but there are intermediate markings where the enabling of t_0 is closer. Continuing from any of these intermediate markings, another token is borrowed on the places p_1, p_2, p_3, but t_0 is not yet enabled. Here comes the different order of transitions into view:

– If the two tokens are carried through places p_1, p_2, p_3 together, there are intermediate markings that are closer to firing t_0, because previously two tokens were missing, now only one. Continuing from these markings a third token is borrowed on places p_1, p_2, p_3, enabling t_0 and yielding a full solution.
– If the two tokens are carried through places p_1, p_2, p_3 separately (i.e., a token is carried through the places, while the other is left in p_4, and this procedure is repeated), there are no intermediate markings of interest, because two

tokens are still missing to enable t_0. In this case the algorithm will not find the full solution.

The order of transitions is non-deterministic, thus it is unknown which order will be omitted. Therefore, in our approach we reproduce all the possible firing sequences without subtree omission when a partial solution is skipped, and check for intermediate markings in the full tree. Although this may yield a computational overhead in some cases, we might lose full solutions otherwise.

New termination criterion. We have developed a new termination criterion, which can efficiently cut the search space without losing any full solutions. When generating increment constraints for a partial solution ps, as a first step the algorithm finds the set of places $P' \subseteq P$ where tokens are needed. Then it estimates the number of tokens required (n). At this point, our new criterion checks if there exists a marking m' for which the following inequalities hold:

$$\begin{gathered} \sum_{p_i \in P'} m'(p_i) \geq n \\ \forall p_j \in P : \, m'(p_j) \geq 0. \end{gathered} \tag{2}$$

The first inequality ensures that at least n tokens are present on the places of P' while the others guarantee that the number of tokens on each place is non-negative. These inequalities define a submarking coverability problem. Using the ILP solver, we can check if the modified form of the state equation (which we discuss in Section 4.2) holds for this problem. If the state equation does not hold, it is a proof that no such marking exists where we have the required number of tokens on the places of P'. Thus, ps can be omitted without losing full solutions.

This approach can also extend the set of decidable problems compared to the former approach. Consider the Petri net on Figure 5 with the reachability problem $(1, 1, 0) \in R(PN, (1, 0, 0))$, i.e., firing t_0 to produce a token on p_1. The algorithm would add the T-invariant t_1, t_2 again and again to enable t_0. Using T-invariant filtering we cannot decide whether there is no full solution or we lost it. Using our new approach we can prove that no marking exist where two tokens are present on p_0, therefore no full solution exists.

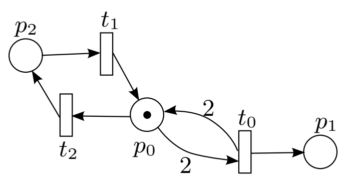

Fig. 5. Example net for the new filtering criterion

4.2 Extensions

We extended the algorithm to handle new types of problems. In this section we present two further extensions: the CEGAR algorithm for solving submarking coverability problems and checking reachability in Petri nets with inhibitor arcs.

Submarking coverability problem. In Section 2 we introduced predicates in the form $Am' \geq b$, where A is a matrix and b is a vector of coefficients. In order to use the state equation, this condition on places must be transformed to a condition on transitions.

At first we substitute m' in the predicate $Am' \geq b$ with the state equation $m_0 + Cx = m'$, which results inequalities of the form $(AC)x \geq b - Am_0$. This set of inequalities can be solved as an ILP problem for transitions. The extended algorithm uses this modified form of the state equation, and expands it with additional (jump or increment) constraints.

Petri nets with inhibitor arcs. The main problem with inhibitor arcs is that they do not appear in any form in the state equation which is used as an abstraction. Therefore, a solution vector produced by the ILP solver may not be realizable because inhibitor arcs disable some transitions. In this case tokens must be removed from some places. Our strategy is to add transitions to the solution vector, that consume tokens from the places connected by inhibitor arcs. Increment constraints are suitable for this purpose, but they have to be generated in a different way:

1. The first step is to construct a dependency graph similar to the original one. The graph consists of transitions that could not fire due to inhibitor arcs and places disabling these transitions. The arcs of the graph have an opposite meaning: an arc from a place to a transition means that the place disables the transition, while the other direction means that firing the transition would decrease the number of tokens on the place. Each source SCC of the graph is interesting, because tokens cannot be consumed from them by another SCC.
2. The second step is to estimate the minimal tokens to be removed from each source SCC. There are two sets of transitions as well, $T_i \subseteq T$ and $X_i \subseteq T$. If one transition in T_i becomes fireable, it may enable all the others in the SCC, while the needs of transitions in X_i must be fulfilled at once.
3. The third step is to construct an increment constraint c for each source SCC from the information of the set of places and the number of tokens to be removed. This yields firing additional transitions (with $r(t) = 0$) to consume tokens from these places.

When a partial solution is not a full solution, and there are transitions disabled by inhibitor arcs, the previous algorithm is used to generate increment constraint. If there are transitions disabled by normal arcs as well, both the original algorithm and the modified version must be used, taking the union of the generated constraints.

Inhibitor arcs also affect some of the optimization methods:

– Stubborn sets currently do not support inhibitor arcs.
– Using T-invariant filtering, an intermediate marking is now of interest when it has less tokens on a place which is connected by inhibitor arc to a transition that cannot fire.
– Our new termination criterion is extended to check whether a reachable marking exists where the required number of tokens are removed.

5 Evaluation

We have implemented our algorithm in the *PetriDotNet* [1] framework. Table 1 contains run-time results, where TO refers to an unacceptable run-time ($>$ 600 seconds). The measured models are published in [4], [11], [12]. In Table 1(a) we have compared our solution to the original algorithm, which is implemented in the *SARA tool* [2] (the numbers in the model names represent the parameters). We have also measured a highly asynchronous consumer-producer model (CP_NR in the table).

Table 1. Measurement results for well-known benchmark problems

(a) Comparison to the original

Model	**SARA**	**Our algorithm**
CP_NR 10	0,2 s	0,5 s
CP_NR 25	111 s	2 s
CP_NR 50	TO	16s
Kanban 1000	0,2 s	1 s
FMS 1500	0,5 s	5 s
MAPK	0,2 s	1 s

(b) Comparison to saturation

Model	**Saturation**	**Our algorithm**
Kanban 1000	TO	1 s
SlottedRing 50	4 s	433 s
DPhil 50	0,5 s	45 s
FMS 1500	TO	5 s

Our implementation is developed in the C# programming language, while the original is in C. This causes a constant speed penalty for our algorithm. Moreover, our algorithm examines more partial solutions, which also yields computational overhead. However, the algorithmic improvements we introduced in this paper significantly reduce the computational effort for certain models (see the consumer-producer model). In addition, our algorithm can in many cases decide a problem that the original one cannot.

We have also compared our algorithm to the well-known saturation-based model checking algorithm [4], implemented in our framework [11]. See the results in Table 1(b). The lesson learned is that if the ILP solver can produce results efficiently (Kanban and FMS models), the CEGAR solution is faster by an order of magnitude than the saturation algorithm. When the size of the model makes the linear programming task difficult, it dominates the run-time, and saturation wins the comparison.

6 Conclusions

The theoretical results presented in this paper are twofold. On one hand, we proved the incompleteness of the iteration strategy of the original CEGAR approach by constructing a counterexample. We also constructed a counterexample that proved the incorrectness of a heuristic used in the original algorithm. We corrected this deficiency by improving the algorithm to detect such situations. On the other hand, our algorithmic improvements reduce the search space, and enable the algorithm to solve the reachability problem for certain, previously unsupported classes of Petri nets. In addition, we extended the algorithm to solve two new classes of problems, namely submarking coverability and handling Petri nets with inhibitor arcs. We demonstrated the efficiency of our improvements with measurements.

References

1. Homepage of the *PetriDotNet* framework., `http://petridotnet.inf.mit.bme.hu/`, [Online; accessed 10-May-2013]
2. Homepage of the *Sara* model checker., `http://service-technology.org/tools/index.html`, [Online; accessed 06-Apr-2013]
3. Chrzastowski-Wachtel, P.: Testing undecidability of the reachability in Petri nets with the help of 10th hilbert problem. In: Donatelli, S., Kleijn, J. (eds.) Application and Theory of Petri Nets 1999, Lecture Notes in Computer Science, vol. 1639, pp. 690–690. Springer (1999)
4. Ciardo, G., Marmorstein, R., Siminiceanu, R.: Saturation unbound. In: Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 379–393. Springer (2003)
5. Dantzig, G.B., Thapa, M.N.: Linear programming 1: introduction. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1997)
6. Esparza, J., Melzer, S., Sifakis, J.: Verification of safety properties using integer programming: Beyond the state equation (1997)
7. Lipton, R.: The Reachability Problem Requires Exponential Space. Research report, Yale University, Dept. of Computer Science (1976)
8. 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)
9. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (April 1989)
10. Valmari, A., Hansen, H.: Can stubborn sets be optimal? In: Lilius, J., Penczek, W. (eds.) Applications and Theory of Petri Nets, Lecture Notes in Computer Science, vol. 6128, pp. 43–62. Springer (2010)
11. Vörös, A., Bartha, T., Darvas, D., Szabó, T., Jámbor, A., Horváth, Á.: Parallel saturation based model checking. In: ISPDC. IEEE Computer Society, Cluj Napoca (2011)
12. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. In: Abdulla, P.A., Leino, K.R.M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 17th International Conference, TACAS 2010 Proceedings. Lecture Notes in Computer Science, vol. 6605, pp. 224–238. Springer (2011)