On the computation of the minimal coverability set of Petri nets

Pierre-Alain Reynier[1] and Frédéric Servais[2]

[1] Aix-Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
pierre-alain.reynier@univ-amu.fr
https://pageperso.lis-lab.fr/~pierre-alain.reynier/
[2] École Supérieure d'Informatique de Bruxelles, Bruxelles, Belgium
frederic.servais@gmail.com

Abstract. The verification of infinite-state systems is a challenging task. A prominent instance is reachability analysis of Petri nets, for which no efficient algorithm is known. The *minimal coverability set* of a Petri net can be understood as an approximation of its reachability set described by means of ω-markings (*i.e.* markings in which some entries may be set to infinity). It allows to solve numerous decision problems on Petri nets, such as any coverability problem. In this paper, we study the computation of the minimal coverability set.
This set can be computed using the Karp and Miller trees, which perform accelerations of cycles along branches [10]. The resulting algorithm may however perform redundant computations. In a previous work [17], we proposed an improved algorithm allowing pruning between branches of the Karp and Miller tree, and proved its correctness. The proof of its correctness was complicated, as the introduction of pruning between branches may yield to incompleteness issues [5, 9].
In this paper, we propose a new proof of the correctness of our algorithm. This new proof relies on an original invariant of the algorithm, leading to the following assets:

1. it is considerably shorter and simpler,
2. it allows to prove the correctness of a more generic algorithm, as the acceleration used is let as a parameter. Indeed, we identify the property that the acceleration should satisfy to ensure completeness.
3. it opens the way to a generalization of our algorithm to extensions of Petri nets.

Keywords: Petri nets · Coverability · Acceleration.

1 Introduction

Verification of infinite-state systems Petri nets [14] constitute one of the most popular formalism for the description and analysis of concurrent systems. While their state space may be infinite, many verification problems are decidable. Dealing with infinite-state systems is useful in numerous situations, such as considering an unbounded number of agents or modelling ressources.

When considering the verification of safety properties for Petri nets, an important problem is the *coverability problem*, which can be understood as a weakening of the reachability problem. It asks whether it is possible to reach a marking larger than or equal to a given target marking, and thus exactly corresponds to fireability of a transition. This problem is ExpSpace-complete [10, 3, 16] and has attracted a lot of interest (see for instance [8, 11, 2]).

The minimal coverability set In this work, we are interested in a related problem, which consists in computing the so-called *minimal coverability set* of a Petri net (MCS for short) [5]. This set can be understood as an approximation of its reachability set described by means of ω-markings (*i.e.* markings in which some entries may be set to infinity). Once it is computed, this set allows to solve any coverability problem, and several other problems such as the *(place) boundedness* and *regularity* problems (see [18]).

The MCS can be derived from the classical Karp and Miller algorithm [10]. This algorithm builds a finite tree representation of the (potentially infinite) unfolding of the reachability graph of the given Petri net. It uses acceleration techniques to collapse branches of the tree and ensure termination. By taking advantage of the fact that Petri nets are strictly monotonic transition systems, the acceleration essentially computes the limit of repeatedly firing a sequence of transitions. However, this algorithm is not efficient as several branches may perform similar computations. This observation led to the Minimal Coverability Tree (MCT) algorithm [5], which introduces comparisons (and pruning) between branches of the tree. However, it was shown that the MCT algorithm is incomplete [13, 9]. The flaw is intricate and, according to [9], difficult to patch, with wrong previous attempts [13].

The Monotone-Pruning algorithm As a solution to this problem, we introduced in [17] the Monotone-Pruning algorithm (MP), an improved Karp and Miller algorithm with pruning. This algorithm can be viewed as the MCT Algorithm with a slightly more aggressive pruning strategy which ensures completeness. The MP algorithm constitutes a simple modification of the Karp and Miller algorithm and thus enjoys the following assets: it is easily amenable to implementation, any strategy of exploration of the Petri net is correct: depth first, breadth first, random ..., and experimental results based on a prototype implementation in Python show promising results [17]. Recently, the MP algorithm has been used successfully in the context of the verification of data-driven workflows [12].

While MP algorithm is simple and includes the elegant ideas of the original MCT Algorithm, the proof of its correctness presented in [17] is long and technical. The main difficulty is to prove the completeness of the algorithm, i.e. to show that the set returned by the algorithm covers every reachable marking (recall that the flaw of MCT algorithm identified in [9] is precisely its incompleteness). In [17], to overcome this difficulty, we reduce the problem to the completeness of the algorithm for a particular class of finite state systems, which we call widened Petri nets (WPN). Yet, the proof of the completeness of MP algorithm

for WPN provided in [17] is approximately ten pages long, and goes through several technical lemmas, making it hard to understand and to generalise.

Contributions of the paper In this paper, we present a new proof of the completeness of MP Algorithm for WPN. More precisely, we consider a more general version of MP Algorithm, in which the acceleration used is considered as a parameter. In the context of WPN, a concretisation function can be associated with an acceleration: it gives a concrete sequence of transitions allowing to reach the ω-marking resulting from the acceleration. We identify a property of the acceleration by means of its concretisation function, which we call *coherence* and prove the completeness of MP Algorithm for WPN provided the acceleration used is coherent. This new proof relies on a simple invariant of the property, whose proof is less than two pages long.

We argue that this new proof has the following assets:

1. it is much more readable, increasing its confidence,
2. it is more general, as the acceleration is now a parameter of the algorithm,
3. it opens the way to a generalisation of MP algorithm to other classes of well-structured transition systems [4, 6, 7, 1].

Related work Other algorithms have been proposed to compute the MCS. First, the CoverProc algorithm has been introduced in [9]. This algorithm follows a different approach and is not based on the Karp and Miller Algorithm. Instead, it relies on pairs of markings, yielding an important overhead in terms of complexity. Another algorithm has been proposed in [15]. This algorithm is however very tailored to Petri nets and relies on ad-hoc tricks to improve its efficiency. In addition, it does not offer the possibility to modify the exploration strategy: it should be depth-first search.

Organisation of the paper Definitions of Petri nets are given in Section 2, together with the notion of minimal coverability set. The Monotone-Pruning algorithm is presented in Section 3, and the overall proof structure of its correctness is given in Section 4. In Section 5, we present our new arguments to prove its completeness. In Section 6, we show that a simple acceleration function satisfies the expected property to ensure completeness of MP Algorithm.

2 Preliminaries

$\mathbb{N}$ denotes the set of natural numbers. A quasi order $\leq$ on a set S is a reflexive and transitive relation on S. Given a quasi order $\leq$ on S, a state $s \in S$ and a subset X of S, we write $s \leq X$ iff there exists an element $s' \in X$ s.t. $s \leq s'$.

Given a finite alphabet Σ, we denote by Σ^* the set of words on Σ, and by ε the empty word. We denote by $\prec$ the (strict) prefix relation on Σ^*: given $u, v \in \Sigma^*$ we have $u \prec v$ iff there exists $w \in \Sigma^*$ such that $uw = v$ and $w \neq \varepsilon$. We denote by $\preceq$ the relation obtained as $\prec \cup =$.

2.1 Markings, ω-markings and labelled trees

Given a finite set P, a *marking on* P is an element of the set $\mathsf{Mark}(P) = \mathbb{N}^P$. The set $\mathsf{Mark}(P)$ is naturally equipped with a partial order denoted $\leq$.

Given a marking $m \in \mathsf{Mark}(P)$, we represent it by giving only the positive components. For instance, $(1, 0, 0, 2)$ on $P = (p_1, p_2, p_3, p_4)$ is represented by the multiset $\{p_1, 2p_4\}$. An *ω-marking on* P is an element of the set $\mathsf{Mark}^\omega(P) = (\mathbb{N} \cup \{\omega\})^P$. The order $\leq$ on $\mathsf{Mark}(P)$ is naturally extended to this set by letting $n < \omega$ for any $n \in \mathbb{N}$, and $\omega \leq \omega$. Addition and subtraction on $\mathsf{Mark}^\omega(P)$ are obtained using the rules $\omega + n = \omega - n = \omega$ for any $n \in \mathbb{N}$. The ω-marking $(\omega, 0, 0, 2)$ on $P = (p_1, p_2, p_3, p_4)$ is represented by the multiset $\{\omega p_1, 2p_4\}$.

Given two sets Σ_1 and Σ_2, a labelled tree is a tuple $\mathcal{T} = (N, n_0, E, \Lambda)$ where N is the set of nodes, $n_0 \in N$ is the root, $E \subseteq N \times \Sigma_2 \times N$ is the set of edges labelled with elements of Σ_2, and $\Lambda : N \to \Sigma_1$ labels nodes with elements of Σ_1. We extend the mapping Λ to sets of nodes: for $S \subseteq N$, $\Lambda(S) = \{\Lambda(n) \mid n \in S\}$. Given a node $n \in N$, we denote by $\mathsf{Ancestor}_{\mathcal{T}}(n)$ the set of ancestors of n in $\mathcal{T}$ (n included). If n is not the root of $\mathcal{T}$, we denote by $\mathsf{parent}_{\mathcal{T}}(n)$ its first ancestor in $\mathcal{T}$. Finally, given two nodes x and y such that $x \in \mathsf{Ancestor}_{\mathcal{T}}(y)$, we denote by $\mathsf{path}_{\mathcal{T}}(x, y) \in E^*$ the sequence of edges leading from x to y in $\mathcal{T}$. We also denote by $\mathsf{pathlabel}_{\mathcal{T}}(x, y) \in \Sigma_2^*$ the label of this path.

2.2 Petri nets

Definition 1 (Petri net (PN)). *A Petri net $\mathcal{N}$ is a tuple (P, T, I, O, m_0) where P is a finite set of* places*, T is a finite set of* transitions *with $P \cap T = \emptyset$, $I : T \to \mathsf{Mark}(P)$ is the backward incidence mapping, representing the* input *tokens, $O : T \to \mathsf{Mark}(P)$ is the forward incidence mapping, representing* output *tokens, and $m_0 \in \mathsf{Mark}(P)$ is the initial marking.*

The semantics of a PN is usually defined on markings, but can easily be extended to ω-markings. We define the semantics of $\mathcal{N} = (P, T, I, O, m_0)$ by its associated labelled transition system $(\mathsf{Mark}^\omega(P), m_0, \Rightarrow)$ where $\Rightarrow \subseteq \mathsf{Mark}^\omega(P) \times \mathsf{Mark}^\omega(P)$ is the transition relation defined by $m \Rightarrow m'$ iff $\exists t \in T$ s.t. $m \geq I(t) \wedge m' = m - I(t) + O(t)$. For convenience we will write, for $t \in T$, $m \overset{t}{\Rightarrow} m'$ if $m \geq I(t)$ and $m' = m - I(t) + O(t)$. In addition, we also write $m' = \mathsf{Post}(m, t)$, this defines the operator Post which computes the successor of an ω-marking by a transition. We naturally extend this operator to sequences of transitions. Given an ω-marking m and a transition t, we write $m \overset{t}{\Rightarrow} \cdot$ iff there exists $m' \in \mathsf{Mark}^\omega(P)$ such that $m \overset{t}{\Rightarrow} m'$. The relation $\Rightarrow^*$ represents the reflexive and transitive closure of $\Rightarrow$. We say that a marking m is *reachable in* $\mathcal{N}$ iff $m_0 \Rightarrow^* m$. We say that a Petri net is bounded if the set of reachable markings is finite.

Example 1. We consider the Petri net $\mathcal{N}$ depicted on Figure 1, which is the example used in [17]. The initial marking is $\{p_1\}$, depicted by the token in the place p_1. For any integer n, we have $\mathsf{Post}(\{p_1\}, t_1(t_3t_4)^n) = \{p_3, np_5\}$. In particular, this net is not bounded as place p_5 is not. ⌟

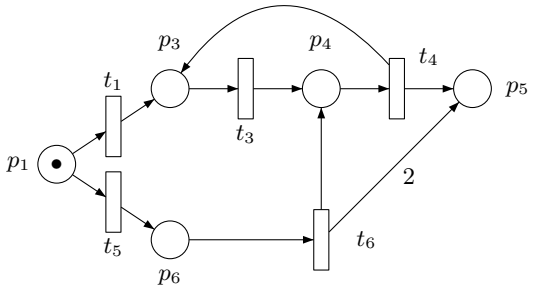


Fig. 1. A Petri net $\mathcal{N}$.

2.3 Minimal Coverability Set of Petri Nets

We recall the definition of minimal coverability set introduced in [5].

Definition 2. *A coverability set of a Petri net $\mathcal{N} = (P, T, I, O, m_0)$ is a finite subset C of $\mathsf{Mark}^\omega(P)$ such that the two following conditions hold:*

1) *for every reachable marking m of $\mathcal{N}$, there exists $m' \in C$ such that $m \leq m'$,*
2) *for every $m' \in C$, either m' is reachable in $\mathcal{N}$ or there exists an infinite strictly increasing sequence of reachable markings $(m_n)_{n\in\mathbb{N}}$ converging to m'.*

A coverability set is minimal iff no proper subset is a coverability set.

One can prove (see [5]) that a PN $\mathcal{N}$ admits a unique minimal coverability set, which we denote by $\mathrm{MCS}(\mathcal{N})$.

Note that every two elements of a minimal coverability set are incomparable. Computing the minimal coverability set from a coverability set is easy. Note also that if the PN is bounded, then the set of reachable markings is finite, and thus the notion of reachable maximal marking is well-defined. In this case, a set of markings is a coverability set iff it contains all maximal reachable markings.

Example 2 (Example 1 continued). The MCS of the Petri net $\mathcal{N}$ is composed of the following ω-markings: $\{p_1\}$, $\{p_6\}$, $\{p_3, \omega p_5\}$, and $\{p_4, \omega p_5\}$. ⌟

3 Presentation of the Monotone-Pruning Algorithm

3.1 Acceleration(s)

Following previous works, as Karp and Miller algorithm, MP algorithm involves an acceleration function. Such a function takes as input a set of ω-markings M and an ω-marking m, and returns an ω-marking m', which can be used to replace m. Several such functions have been considered in the literature. A classical one is the mapping $\mathsf{Acc}_{\mathsf{all}} : 2^{\mathsf{Mark}^\omega(P)} \times \mathsf{Mark}^\omega(P) \to \mathsf{Mark}^\omega(P)$ which is defined as follows:

$$\forall p \in P, \mathsf{Acc}_{\mathsf{all}}(M, m)(p) = \begin{cases} \omega & \text{if } \exists m' \in M \mid m' < m \wedge m'(p) < m(p) < \omega \\ m(p) & \text{otherwise.} \end{cases}$$

A weaker acceleration computes the acceleration w.r.t. a single ω-marking chosen in the set M. The mapping $\mathsf{Acc}_{\mathsf{one}} : 2^{\mathsf{Mark}^\omega(P)} \times \mathsf{Mark}^\omega(P) \to \mathsf{Mark}^\omega(P)$ is defined as follows:

- if there exists $m' \in M$ such that $m' < m$, then we fix one such ω-marking m', and define $\mathsf{Acc}_{\mathsf{one}}(M, m)$ as follows:

$$\forall p \in P, \mathsf{Acc}_{\mathsf{one}}(M,m)(p) = \begin{cases} \omega & \text{if } m'(p) < m(p) < \omega \\ m(p) & \text{otherwise.} \end{cases}$$

- otherwise, we define $\mathsf{Acc}_{\mathsf{one}}(M, m) = m$.

In both functions, the acceleration *uses* one (or several) of the ω-markings in M to build the new ω-marking. Note that these accelerations will always be used along a branch of the tree constructed by the algorithm.

3.2 Definition of the algorithm

The K&M Algorithm uses comparisons along the same branch to compute the acceleration and stop the exploration. We present in this section the Monotone-Pruning Algorithm which includes a comparison (and a pruning) between branches. We denote this algorithm by MP. It has as a parameter an acceleration function Acc as defined in the previous section.

Algorithm 1 Monotone Pruning Algorithm for Petri Nets.

Require: A Petri net $\mathcal{N} = (P, T, I, O, m_0)$ and an acceleration function Acc.
Ensure: A labelled tree $\mathcal{C} = (X, x_0, B, \Lambda)$ with nodes (resp. edges) labelled with elements in $\mathsf{Mark}^\omega(P)$ (resp. T), and a set $\mathsf{Act} \subseteq X$ such that $\Lambda(\mathsf{Act}) = \mathrm{MCS}(\mathcal{N})$.

1: Let x_0 be a new node such that $\Lambda(x_0) = m_0$;
2: $X := \{x_0\}$; $\mathsf{Act} := X$; $\mathsf{Wait} := \{(x_0, t) \mid \Lambda(x_0) \stackrel{t}{\Rightarrow} \cdot\}$; $B := \emptyset$;
3: **while** $\mathsf{Wait} \neq \emptyset$ **do**
4: Pop (n, t) from Wait. $m := \mathsf{Post}(\Lambda(n), t)$;
5: **if** $n \in \mathsf{Act}$ and $m \not\leq \Lambda(\mathsf{Act})$ **then**
6: Let n' be a new node such that $\Lambda(n') = \mathsf{Acc}(\Lambda(\mathsf{Ancestor}_{\mathcal{C}}(n) \cap \mathsf{Act}), m)$;
7: $X := X \cup \{n'\}$; $B := B \cup \{(n, t, n')\}$;
8: $\mathsf{Act} := \mathsf{Act} \setminus \{x \mid \exists y \in \mathsf{Ancestor}_{\mathcal{C}}(x)$ s.t. $\Lambda(y) \leq \Lambda(n') \wedge (y \in \mathsf{Act} \vee y \notin \mathsf{Ancestor}_{\mathcal{C}}(n'))\}$;
9: $\mathsf{Act} := \mathsf{Act} \cup \{n'\}$; $\mathsf{Wait} := \mathsf{Wait} \cup \{(n', t') \mid \Lambda(n') \stackrel{t'}{\Rightarrow} \cdot\}$;
10: **end if**
11: **end while**
12: Return $\mathcal{C} = (X, x_0, B, \Lambda)$ and Act.

As Karp and Miller Algorithm, the MP Algorithm builds a tree $\mathcal{C}$ in which nodes are labelled by ω-markings and edges by transitions of the Petri net. Therefore it proceeds in an exploration of the reachability tree of the Petri net,

and uses acceleration along branches to reach the "limit" markings. In addition, it can prune branches that are covered by nodes on other branches. This additional pruning is the source of efficiency, as it avoids to perform redundant computations. It is also the source of difficulty, as a previous attempt of introduction of such pruning led to an incomplete algorithm (MCT Algorithm [5]). In order to obtain a complete algorithm, nodes of the tree are partitioned into two subsets: active nodes, and inactive ones. Intuitively, active nodes will form the minimal coverability set of the Petri net, while inactive ones are kept to ensure completeness of the algorithm.

Given a pair (n, t) popped from Wait, the introduction in $\mathcal{C}$ of the new node obtained from (n, t) proceeds in the following steps:

1. the "regular" successor marking is computed: $m = \mathsf{Post}(\Lambda(n), t)$ (Line 4) ;
2. node n should be active and marking m should not be covered by some active node (test of Line 5) ;
3. the marking resulting from the acceleration of m w.r.t. the *active ancestors* of node n is computed and associated with a new node n': $\Lambda(n') = \mathsf{Acc}(\Lambda(\mathsf{Ancestor}_{\mathcal{C}}(n) \cap \mathsf{Act}), m)$ (Line 6) ;
4. update of Act: some nodes are "deactivated", i.e. removed from Act (Line 8).
5. the new node n' is declared as active and Wait is updated (Line 9) ;

We detail the update of the set Act. Intuitively, one wants to deactivate nodes (and their descendants) that are covered by the new node n'. In MP Algorithm (see Line 8), node x is deactivated iff its ancestor y is either active ($y \in \mathsf{Act}$), or is not itself an ancestor of n' ($y \notin \mathsf{Ancestor}_{\mathcal{C}}(n')$). In this case, we say that x *is deactivated by* n'. This subtle condition constitutes the main difference between MP and MCT Algorithms.

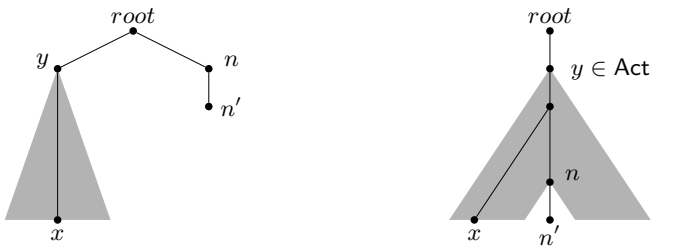
To illustrate the behaviour of MP Algorithm, consider the introduction of a new node n' obtained from $(n, t) \in \mathsf{Wait}$, and a node y such that $\Lambda(y) \leq \Lambda(n')$, y can be used to deactivate nodes in two ways:

- if $y \notin \mathsf{Ancestor}_{\mathcal{C}}(n')$, then no matter whether y is active or not, all its descendants are deactivated (represented in gray on Figure 2(a)),
- if $y \in \mathsf{Ancestor}_{\mathcal{C}}(n')$, then y must be active ($y \in \mathsf{Act}$), and in that case all its descendants are deactivated, except node n' itself as it is added to Act at Line 9 (see Figure 2(b)).

4 Structure of the proof of correction of MP Algorithm

In this section, we describe the overall structure of the proof of [17]. Given an input Petri net $\mathcal{N}$, MP Algorithm returns a set Act of ω-markings. We say that MP Algorithm is:

- *sound* if for every $m \in \mathsf{Act}$, there exists $n \in \mathrm{MCS}(\mathcal{N})$ such that $m \leq n$,
- *complete* if for every $n \in \mathrm{MCS}(\mathcal{N})$, there exists $m \in \mathsf{Act}$ such that $n \leq m$.



(a) $y \notin \mathsf{Ancestor}_{\mathcal{C}}(n')$ (b) $y \in \mathsf{Ancestor}_{\mathcal{C}}(n') \cap \mathsf{Act}$

Fig. 2. Deactivations of MP Algorithm.

It is easy to show that Act is composed of pairwise incomparable ω-markings. Hence, if MP Algorithm is both sound and complete, then it returns exactly the set $\mathrm{MCS}(\mathcal{N})$.

4.1 Widened Petri nets

Our proof involves a widening operation which turns a Petri net into a finite state system. Let P be a finite set, and $\varphi \in \mathsf{Mark}(P)$ be a marking. We consider the *finite* set of ω-markings whose finite components (*i.e.* values different from ω) are less or equal than φ. Formally, we define:

$$\mathsf{Mark}^{\omega}_{\varphi}(P) = \{m \in \mathsf{Mark}^{\omega}(P) \mid \forall p \in P, m(p) \leq \varphi(p) \vee m(p) = \omega\}.$$

The widening operator Widen_{φ} maps an ω-marking to an element of $\mathsf{Mark}^{\omega}_{\varphi}(P)$:

$$\forall m \in \mathsf{Mark}^{\omega}(P), \forall p \in P, \mathsf{Widen}_{\varphi}(m)(p) = \begin{cases} m(p) & \text{if } m(p) \leq \varphi(p) \\ \omega & \text{otherwise.} \end{cases}$$

Note that this operator trivially satisfies $m \leq \mathsf{Widen}_{\varphi}(m)$.

Definition 3 (Widened Petri net). *A widened Petri net (*WPN *for short) is a pair $(\mathcal{N}, \varphi)$ composed of a* PN $\mathcal{N} = (P, T, I, O, m_0)$ *and of a marking $\varphi \in \mathsf{Mark}(P)$ such that $m_0 \leq \varphi$.*

The semantics of $(\mathcal{N}, \varphi)$ is given by its associated labelled transition system $(\mathsf{Mark}^{\omega}_{\varphi}(P), m_0, \Rightarrow_{\varphi})$ where for $m, m' \in \mathsf{Mark}^{\omega}_{\varphi}(P)$, and $t \in T$, we have $m \overset{t}{\Rightarrow}_{\varphi} m'$ iff $m' = \mathsf{Widen}_{\varphi}(\mathsf{Post}(m, t))$. We carry over from PN to WPN the relevant notions, such as reachable marking. We define the operator Post_{φ} by $\mathsf{Post}_{\varphi}(m, t) = \mathsf{Widen}_{\varphi}(\mathsf{Post}(m, t))$. Subscript φ may be omitted when it is clear from the context. Finally, the minimal coverability set of a widened Petri net $(\mathcal{N}, \varphi)$ is simply the set of its maximal reachable states as its reachability set is finite. It is denoted $\mathrm{MCS}(\mathcal{N}, \varphi)$.

Example 3 (Example 1 continued). Consider the mapping φ associating 1 to places p_1, p_3, p_4 and p_6, and 3 to place p_5, and the widened Petri net $(\mathcal{N}, \varphi)$.

Then from marking $\{p_4, 3p_5\}$, the firing of t_4 results in the marking $\{p_3, \omega p_5\}$, instead of the marking $\{p_3, 4p_5\}$ in the standard semantics. One can compute the MCS of this WPN. Due to the choice of φ, it coincides with $\mathrm{MCS}(\mathcal{N})$. ⌟

In the sequel, we will consider the execution of MP Algorithm on widened Petri nets, which we will denote by $\mathrm{MP}_{\mathrm{WPN}}$. Let $(\mathcal{N}, \varphi)$ be a WPN. The only difference is that the operator Post (resp. $\Rightarrow$) must be replaced by the operator Post_φ (resp. $\Rightarrow_\varphi$). Thus, all the ω-markings computed by the algorithm belong to $\mathsf{Mark}^\omega_\varphi(P)$.

4.2 Structure of the proof of correction presented in [17]

The structure of the proof of correction presented in [17] is depicted in Figure 3. In this proof, all results have rather simple proofs, except the completeness of $\mathrm{MP}_{\mathrm{WPN}}$. The proof of this property presented in [17] is approximately ten pages long. The main contribution of this paper is a very short proof of this property. It is presented in the next section, and stated as Theorem 1.

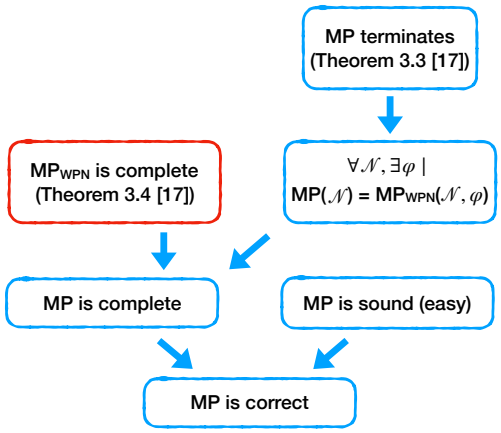


Fig. 3. Structure of the proof of [17]. The main difficulty lies in the completeness of $\mathrm{MP}_{\mathrm{WPN}}$, depicted in red.

5 Completeness of MP Algorithm for WPN

In this section, we present the main contribution of this article, which is a new and simple proof of the completeness of MP Algorithm for WPN.

5.1 Coherence of an acceleration

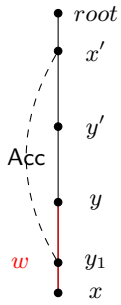
MP Algorithm builds a labelled tree $\mathcal{C}$. In this context, the acceleration is applied along a branch β starting from the root, and leading to a node n whose marking is m. More precisely, there exists a set N of nodes on β, which are active ancestors of node n, and such that M is the set of markings of N. We recall the notion of concretization that "explains" how the accelerated marking is computed, by giving an explicit sequence of transitions leading to the accelerated marking.

Definition 4. *We consider an acceleration* Acc *and a labelled tree* $\mathcal{C} = (X, x_0, B, \Lambda)$ *obtained from a WPN* $(\mathcal{N}, \varphi)$ *using* Acc. *A* concretization function *is a mapping* $\gamma : B^* \to T^*$ *associating to every path in* $\mathcal{C}$ *a sequence of transitions of* $\mathcal{N}$. *In addition, given* $x, y \in X$ *such that* $y \in \mathsf{Ancestor}_{\mathcal{C}}(x)$, *we have* $\Lambda(x) = \mathsf{Post}_{\varphi}(\Lambda(y), \gamma(\mathsf{path}_{\mathcal{C}}(y, x)))$.

In order to have a generic proof, independent of the acceleration considered, we identify a property of the acceleration together with its concretization function which ensures that the algorithm is correct.

Definition 5. *We consider an acceleration function* Acc. *We say that* Acc *is* coherent *if it admits a concretization function* γ *such that the following property holds:* (†) *Let* x, y *in* $\mathcal{C}$ *and* $w = \mathsf{path}_{\mathcal{C}}(y, x) \in B^*$. *Then for every* $\rho_p \preceq \gamma(w)$, *there exist two nodes* x' *and* y' *such that:*

- $\mathsf{Post}(\Lambda(y), \rho_p) \geq \Lambda(y')$,
- x' *is an ancestor of* x, *used by some acceleration for node* y_1 *on the path from* y *to* x,
- y' *lies between* x' *and* y_1.



We prove now that when an acceleration Acc is coherent, then MP Algorithm satisfies a property that we call the coherence of this algorithm.

Lemma 1 (MP Algorithm is coherent). *We consider MP Algorithm with a coherent acceleration* Acc, *with concretization* γ. *Then the following property holds: consider three nodes* x, y, z *such that* $x, z \in \mathsf{Act}$ *and* $y \in \mathsf{Ancestor}_{\mathcal{C}}(x)$, *and define* $\rho = \gamma(\mathsf{path}_{\mathcal{C}}(y, x))$. *If* $\Lambda(z) \geq \mathsf{Post}(\Lambda(y), \rho_p)$ *for some* $\rho_p \preceq \rho$, *then* $y \in \mathsf{Ancestor}_{\mathcal{C}}(z)$.

Proof. As the acceleration is coherent, we fix an adequate concretization function γ. We consider nodes x, y, z and some $\rho_p \preceq \rho = \gamma(\mathsf{path}_{\mathcal{C}}(y, x))$ as in the premises of the statement. Thanks to property (†), there exist two nodes x', y' such that:

- $\mathsf{Post}(\Lambda(y), \rho_p) \geq \Lambda(y')$,
- x' is an ancestor of x, used by some acceleration for node y_1 on the path from y to x,
- y' lies between x' and y_1.

By contradiction, assume that $y' \notin \mathsf{Ancestor}_{\mathcal{C}}(z)$. We have $\Lambda(z) \geq \mathsf{Post}(\Lambda(y), \rho_p)$ and $\mathsf{Post}(\Lambda(y), \rho_p) \geq \Lambda(y')$, hence $\Lambda(z) \geq \Lambda(y')$. Then, by definition of MP Algorithm, x is deactivated by z. This is in contradiction with our assumption that x and z are active. Thus, we have $y' \in \mathsf{Ancestor}_{\mathcal{C}}(z)$.

Assume now that $y_1 \notin \mathsf{Ancestor}_{\mathcal{C}}(z)$. Recall that y_1 used the node x' when the acceleration has been applied. By definition of MP Algorithm, it deactivated everything below x', except itself. As we have that y' is between x' and y_1, y' is an ancestor of z, and y_1 is not an ancestor of z, this entails that y_1 deactivated z, which is a contradiction. Thus, we have $y_1 \in \mathsf{Ancestor}_{\mathcal{C}}(z)$.

In particular, this implies $y \in \mathsf{Ancestor}_{\mathcal{C}}(z)$, as expected. □

5.2 New proof

Our new proof relies on a simple invariant of the algorithm, from which completeness easily follows. This invariant is defined as the following property $(\mathcal{P})$:

$$\forall m \in \mathsf{Reach}(\mathcal{N}), \exists (x, \rho) \in \mathsf{Act} \times T^* \text{ such that:}$$

$$\begin{cases} (1) & \mathsf{Post}(\Lambda(x), \rho) \geq m \\ (2) & \rho \neq \epsilon \Rightarrow (x, \mathsf{first}(\rho)) \in \mathsf{Wait} \\ (3) & \forall \epsilon \neq \rho_p \preceq \rho, \neg \exists z \in \mathsf{Act}. \Lambda(z) \geq \mathsf{Post}(\Lambda(x), \rho_p) \end{cases}$$

Intuitively, the invariant states that for every reachable marking m, there exists a pair (x, ρ) which allows to cover m (property (1)), whose exploration is still in the waiting list (property (2)), and whose exploration will not be stopped by another active node (property (3)).

We now prove that the MP Algorithm satisfies the invariant $(\mathcal{P})$:

Lemma 2. *When used with a coherent acceleration, the MP Algorithm satisfies the property $(\mathcal{P})$ at every step of its execution.*

Proof. We proceed by induction on the number of steps of the algorithm.

Base case. Initially, the invariant is trivially satisfied as there is a single active node corresponding to the initial marking.

Induction. $\mathcal{P}(k) \Rightarrow \mathcal{P}(k+1)$

Let $m \in \mathsf{Reach}(\mathcal{N})$. By $\mathcal{P}(k)$, there exists (x, ρ) as given by $\mathcal{P}$.

We consider different cases depending on what happens in the While loop of the algorithm. At Line 4, a pair (n, t) is popped from the waiting list. If this pair does not pass the test of Line 5, then nothing changes and the pair (x, ρ) still satisfies the properties. The interesting case is when this pair passes the test of Line 5. We distinguish three cases:

1. if x is not deactivated and no successor of x by prefixes of ρ is covered by n', then we can simply choose the pair (x, ρ).
2. otherwise, assume that some successor of x by a (possibly empty) prefix of ρ is covered by n'. Then, let ρ_1 be the longest prefix of ρ such that $\Lambda(n') \geq \mathsf{Post}(\Lambda(x), \rho_1)$. We claim that we can choose the pair (n', ρ') where $\rho' = \rho_1^{-1}.\rho$. Indeed:
   - $n' \in \mathsf{Act}$ (Line 9),

– Property (1) follows from monotonicity of Petri nets and from $\Lambda(n') \geq \mathsf{Post}(\Lambda(x), \rho_1)$,
– Property (2) follows from Line 9,
– In order to show that Property (3) holds, we proceed by contradiction. Assume that there exists ρ'_p a non-empty prefix of ρ' and an active node z such that $\Lambda(z) \geq \mathsf{Post}(\Lambda(n'), \rho'_p)$. Then we have:

$$\mathsf{Post}(\Lambda(x), \rho_1\rho'_p) \leq \mathsf{Post}(\Lambda(n'), \rho'_p) \leq \Lambda(z)$$

As $\rho_1\rho'_p \neq \epsilon$ and $\rho_1\rho'_p \preceq \rho$, Property (3) of our invariant for (x, ρ) implies that z is a new active node, *i.e.* $z = n'$. This is a contradiction with our choice of ρ_1 of maximal length.

3. otherwise, x is deactivated by n': n' dominates a strict ancestor y of x such that $y \in \mathsf{Act}$ or $y \notin \mathsf{Ancestor}_{\mathcal{C}}(n')$ (see Line 8 of the algorithm). We fix such a node y and let $w = \mathsf{path}_{\mathcal{C}}(y, x)$. We define $\rho_0 = \gamma(w)$ and ρ_1 as the longest prefix of ρ_0 such that $\Lambda(n') \geq \mathsf{Post}(\Lambda(n'), \rho_1)$. We write $\rho_0 = \rho_1\rho_2$ and claim that the pair $(n', \rho_2\rho)$ satisfies the properties of the invariant. Property (1) follows directly from monotonicity of Petri nets. Property (2) follows from the fact that n' has just been added to $\mathcal{C}$.
We prove now Property (3). By contradiction, assume that there exists ρ_p a non-empty prefix of $\rho_2\rho$ and an active node z such that $\Lambda(z) \geq \mathsf{Post}(\Lambda(n'), \rho_p)$. Then we also have, by monotonicity, $\Lambda(z) \geq \mathsf{Post}(\Lambda(y), \rho_p)$.
First case: $z = n'$. As we are not in Case 2, ρ_p should be a prefix of ρ_2. But this in contradiction with the definition of ρ_2.
Second case: $z \neq n'$. In particular, z is already active at the previous iteration of the algorithm. By Property (3) of the invariant for (x, ρ), ρ_p is a prefix of ρ_2.
We consider the prefix $\rho_1\rho_p$ of $\rho_0 = \gamma(w)$. We can apply Lemma 1 and deduce that $y \in \mathsf{Ancestor}_{\mathcal{C}}(z)$. Thus z is deactivated by the construction of n' (see Line 8 of the algorithm), yielding the contradiction as we supposed z is active.

□

Theorem 1. *If* Acc *is coherent, then* MP *Algorithm for* WPN *is complete.*

Proof. The result directly follows from Lemma 2 and from the termination of the algorithm. Consider some reachable marking m and the set Act returned by MP Algorithm after its termination. Thanks to Lemma 2, there exists some pair (x, ρ) as given by property $(\mathcal{P})$. As the waiting list is empty, we have $\rho = \epsilon$, hence property (1) directly gives the completeness of Act. □

6 Coherence of the acceleration $\mathsf{Acc}_{\mathsf{one}}$

In this section, we exhibit a concretization function for the acceleration $\mathsf{Acc}_{\mathsf{one}}$ which allows to show that this acceleration is coherent.

Definition 6 (Concretization function for $\mathsf{Acc}_{\mathsf{one}}$). *The concretization function is a morphism γ_{one} from B^* to T^*. We let $M = \max\{\varphi(p) \mid p \in P\} + 1$.*

Let $b = (n, t, n') \in B$. We assume γ_{one} is defined on all edges $(x, u, y) \in B$ such that $y \in \mathsf{Ancestor}_{\mathcal{C}}(n)$.

Let $m = \mathsf{Post}_{\varphi}(\Lambda(n), t)$, then there are two cases, either :

1. *$\Lambda(n') = m$ (t is not accelerated), then we define $\gamma(b) = t$, or*
2. *$\Lambda(n') > m$. Let x be the ancestor of n used for this acceleration, and $w = \mathsf{path}_{\mathcal{C}}(x, n) \in B^*$. Then we define:*
   $\gamma_{\mathsf{one}}(b) = t.(\gamma_{\mathsf{one}}(w).t)^M$

The following property can easily be proved by induction:

Lemma 3. *The mapping γ_{one} is a concretization of the acceleration $\mathsf{Acc}_{\mathsf{one}}$.*

By reasoning on γ_{one} and using again an induction, we can show the existence of adequate ancestors, to prove the following property:

Lemma 4. *The acceleration $\mathsf{Acc}_{\mathsf{one}}$ is coherent.*

7 Conclusion

In this paper, we have provided a new proof of the completeness of MP Algorithm, an algorithm introduced in [17] to compute the minimal coverability set of a Petri net. The new proof relies on an original invariant, is considerably shorter than that of [17], and allows to identify the property that the acceleration should meet to ensure the completeness of the algorithm.

As future work, we would like to extend MP Algorithm to more general classes of well-structured transition systems. To this end, we plan to rely on the representation of downward-closed sets using finite unions of ideals, as introduced in [6, 7]. This setting has recently been used to develop an *ideal Karp and Miller* algorithm in [1], which should be a good basis for extending MP Algorithm to well-structured transition systems.

References

1. M. Blondin, A. Finkel, and J. Goubault-Larrecq. Forward analysis for wsts, part III: karp-miller trees. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017*, volume 93 of *LIPIcs*, pages 16:1–16:15. Leibniz-Zentrum fuer Informatik, 2017.
2. M. Blondin, A. Finkel, C. Haase, and S. Haddad. Approaching the coverability problem continuously. In *Proc. of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2016*, volume 9636 of *Lecture Notes in Computer Science*, pages 480–496. Springer, 2016.
3. E. Cardoza, R. J. Lipton, and A. R. Meyer. Exponential space complete problems for petri nets and commutative semigroups: Preliminary report. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*, pages 50–54. ACM, 1976.

4. A. Finkel. A generalization of the procedure of Karp and Miller to well structured transition system. In *Proc. ICALP'87*, volume 267 of *LNCS*, pages 499–508. Springer, 1987.
5. A. Finkel. The minimal coverability graph for Petri nets. In *Proc. ICATPN'91*, volume 674 of *LNCS*, pages 210–243. Springer, 1993.
6. A. Finkel and J. Goubault-Larrecq. Forward analysis for WSTS, part I: Completions. In *Proc. STACS'09*, volume 3 of *LIPIcs*, pages 433–444. Leibniz-Zentrum für Informatik, 2009.
7. A. Finkel and J. Goubault-Larrecq. Forward analysis for wsts, part II: complete WSTS. *Logical Methods in Computer Science*, 8(3), 2012.
8. G. Geeraerts, J. Raskin, and L. V. Begin. Expand, enlarge and check: New algorithms for the coverability problem of WSTS. *J. Comput. Syst. Sci.*, 72(1):180–203, 2006.
9. G. Geeraerts, J.-F. Raskin, and L. Van Begin. On the efficient computation of the coverability set for petri nets. *International Journal of Foundations of Computer Science*, 21(2):135–165, 2010.
10. R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.
11. J. Kloos, R. Majumdar, F. Niksic, and R. Piskac. Incremental, inductive coverability. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 2013.
12. Y. Li, A. Deutsch, and V. Vianu. Verifas: A practical verifier for artifact systems. *Proc. VLDB Endow.*, 11(3):283–296, Nov. 2017.
13. K. Lüttge. Zustandsgraphen von Petri-Netzen. Master's thesis, Humboldt-Universität, 1995.
14. C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.
15. A. Piipponen and A. Valmari. Constructing minimal coverability sets. *Fundam. Inform.*, 143(3-4):393–414, 2016.
16. C. Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978.
17. P.-A. Reynier and F. Servais. Minimal coverability set for Petri nets: Karp and Miller algorithm with pruning. *Fundam. Inform.*, 122(1-2):1–30, 2013.
18. K. Schmidt. Model-checking with coverability graphs. *Form. Methods Syst. Des.*, 15(3):239–254, 1999.