A counter-example to the minimal coverability tree algorithm

A. Finkel, G. Geeraerts, J.-F. Raskin and L. Van Begin

Abstract

In [1], an algorithm to compute a *minimal coverability tree* for Petri nets has been presented. This document demonstrates, thanks to a simple counter-example, that this algorithm may compute an under-approximation of a coverability tree, i.e. a tree whose set of nodes is not sufficient to cover all the reachable markings.

1 Preliminaries

Definition 1 *A* **Petri Net** *(PN for short) $\mathcal{N}$ is a tuple $\langle \mathcal{P}, \mathcal{T}, \mathbf{m}_0 \rangle$, where $\mathcal{P} = \{p_1, p_2, \ldots p_n\}$ is a finite set of places, $\mathcal{T}$ is finite set of transitions. A* **marking** *of the places is a function $\mathbf{m} : \mathcal{P} \mapsto \mathbb{N}$. A marking $\mathbf{m}$ can also be seen as a vector v such that $v^T = [\mathbf{m}(p_1), \mathbf{m}(p_2), \ldots, \mathbf{m}(p_n)]$. $\mathbf{m}_0$ is the initial marking. A transition $t \in \mathcal{T}$ is a pair $\langle I, O \rangle$ where $I : \mathcal{P} \mapsto \mathbb{N}$ and $O : \mathcal{P} \mapsto \mathbb{N}$.*

Given a Petri net $N = \langle \mathcal{P}, \mathcal{T}, \mathbf{m}_0 \rangle$, an **$\omega$-marking** is a function $\mathbf{m}$ that associates to each place of $\mathcal{P}$ either a natural number, or ω. Notice that markings are particular cases of ω-markings. We define $\omega + c = \omega$ and $\omega - c = \omega$ for all $c \in \mathbb{N}$.

Definition 2 *Given a PN $\mathcal{N} = \langle \mathcal{P}, \mathcal{T}, \mathbf{m}_0 \rangle$, and an ω-marking $\mathbf{m}$ of $\mathcal{N}$, a transition $t = \langle I, O \rangle$ is said to be* enabled in $\mathbf{m}$ *iff $\forall p \in \mathcal{P} : \mathbf{m}(p) \geq I(p)$. An enabled transition $t = \langle I, O \rangle$ can* occur*, which transforms the ω-marking $\mathbf{m}$ into a new ω-marking $\mathbf{m}'$ (we denote this by $\mathbf{m} \xrightarrow{t} \mathbf{m}'$). $\mathbf{m}'$ is computed as follows: $\forall p \in \mathcal{P} : \mathbf{m}'(p) = \mathbf{m}(p) - I(p) + O(p)$. We note $\mathbf{m} \xrightarrow{*} \mathbf{m}'$ when there exists a sequence of transitions $t_1, \ldots, t_k$ and a sequence of ω-markings $\mathbf{m}_1, \ldots, \mathbf{m}_{k-1}$ such that $\mathbf{m} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \ldots \xrightarrow{t_{k-1}} \mathbf{m}_{k-1} \xrightarrow{k} \mathbf{m}'$. The set of reachable markings of $\mathcal{N}$, noted $\mathsf{RS}(\mathcal{N})$, is the set $\{\mathbf{m} \mid \mathbf{m}_0 \xrightarrow{*} \mathbf{m}\}$.*

Given two ω-markings m_1 and m_2 ranging over set of places P, we have $m_1 \preccurlyeq m_2$ iff $\forall p \in P : m_1(p) \leq m_2(p)$. In particular: $i < \omega$, for any $i \in \mathbb{N}$. Moreover $m_1 \prec m_2$ iff $m_1 \preccurlyeq m_2$ and $m_2 \not\preccurlyeq m_1$.

A *coverability set* of a PN $\mathcal{N} = \langle \mathcal{P}, \mathcal{T}, \mathbf{m}_0 \rangle$ is a set of ω-markings $\mathsf{CS}(\mathcal{N})$ such that $\mathsf{RS}(\mathcal{N}) = \{\mathbf{m} \in \mathbb{N}^{|\mathcal{P}|} \mid \exists \mathbf{m}' \in \mathsf{CS}(\mathcal{N}) : \mathbf{m} \preccurlyeq \mathbf{m}'\}$. A coverability set $\mathsf{CS}(\mathcal{N})$ is *minimal* if and only if $\forall \mathbf{m}, \mathbf{m}' \in \mathsf{CS}(\mathcal{N}) : \mathbf{m} \preccurlyeq \mathbf{m}'$ implies $\mathbf{m} = \mathbf{m}'$.

Proposition 1 *For any PN $\mathcal{N}$, there exists a minimal coverability set $\mathsf{CS}(\mathcal{N})$ such that (i) $\mathsf{CS}(\mathcal{N})$ is unique and (ii) $\mathsf{CS}(\mathcal{N})$ is finite.*

Definition 3 A *labelled tree* of a Petri net $\mathcal{N} = \langle \mathcal{P}, \mathcal{T}, \mathbf{m}_0 \rangle$ is a directed acyclic graph $\langle N, root, B, \Lambda \rangle$ where N is a finite set of nodes, $root$ is the root node, $B \subseteq N \times N$ is a transition relation and $\Lambda : B \mapsto (\mathbb{N} \cup \{\omega\})^{|\mathcal{N}|}$ is a function labelling the nodes by ω-markings of $\mathcal{N}$. B is such that (i) for all $n \in N \setminus \{root\}$, there exists one and only one $n' \in N$ such that $B(n', n)$ and (ii) there does not exist $n \in N$ such that $B(n, root)$.

A labelled tree $\mathcal{T} = \langle N, root, B, \Lambda \rangle$ is a *coverability tree* for the Petri net $\mathcal{N} = \langle \mathcal{P}, \mathcal{T}, \mathbf{m}_0 \rangle$ if the set $\{\mathbf{m} \mid \exists n \in N : \Lambda(n) = \mathbf{m}\}$ is a coverability set of $\mathcal{N}$. If that set is the minimal coverability set of $\mathcal{N}$, we say that $\mathcal{T}$ is a *minimal coverability tree*.

2 The minimal coverability tree algorithm [1]

In this section we briefly recall the main ideas of the algorithm to compute a *minimal coverability tree*, as presented by Finkel in [1]. The algorithm is given at Algorithm 1.

Let us first detail several auxiliary function we use in the algorithm. Given a labelled tree $\mathcal{T}$, $\mathsf{nodes}(\mathcal{T})$ returns the set of all the nodes of $\mathcal{T}$. Given a node n, $\mathsf{subtree}(n)$ returns the maximal subtree rooted at n.

Algorithm 1 is essentially a refinement of the classical Karp&Miller algorithm, see [2]. A set *to_treat* holds the nodes that are waiting to be processed (initially, this sets contains the root node of the labelled tree, whose label is the initial marking of the net). The processing of a node consist in trying to apply several *reduction rules*, which are detailed along the various cases of the **if**. Whenever there exists in the labelled tree computed so far a node n_1 that is *larger* than the node n currently processed, the algorithm decides to discard n. This node gets suppressed from the labelled tree. When one finds in the labelled tree computed so far a node n_1 that is *smaller* than n, the **Accelerate** function, which is the same as in the K&M algorithm, is first called. Remark that in the case where n has no ancestor smaller than itself, the function simply returns $\Lambda(n)$.

Accelerate(n)
begin
    $m_\omega \leftarrow \Lambda(n)$;
    $m \leftarrow \Lambda(n)$;
    foreach *ancestor* n_1 *of* n *s.t.* $\Lambda(n_1) \prec m$ **do**
        $m_1 \leftarrow \Lambda(n_1)$;
        foreach *place* p *s.t.* $m_1(p) < m(p)$ **do**
            $m_\omega(p) \leftarrow \omega$;
    return(m_ω) ;
end

Then, we check whether there exists an ancestor n_1 of n that is smaller than n. In this case, the marking returned by **Accelerate** contains ω's, and we place it as high as possible in the labelled tree. Finally, we look at all the other nodes that are smaller than n (without being one of its ancestors). Of course, all these nodes have to be forgotten.

Finally, in the case where none of the reductions could be applied, we simply develop the successors of the node n that has been taken out of *to_treat*, and add them to the labelled tree, as well as to *to_treat*.

3 Counter-example to the algorithm

This section presents our counter-example to the algorithm recalled in the previous section. The algorithm presented in [1] is supposed to compute the minimal coverability set of any

Algorithm 1: The solution of [1] to compute a minimal coverability tree of a Petri net.

Data : A Petri net $\mathcal{P}$ with initial marking m_0.

Result : a minimal coverability tree of $\mathcal{P}$.

begin

  Let $\mathcal{T} = \langle N, n_0, B, \Lambda \rangle$ be the labelled tree computed as follows:

  $to\_treat = \{n_0\}$ such that $\Lambda(n_0) = m_0$, $N = \{n_0\}$, $B = \emptyset$;

  while $to\_treat \neq \emptyset$ **do**

    chose and **remove** n in to_treat ;

    if *There is* $n_1 \in N$ *s.t.* $\Lambda(n) = \Lambda(n_1)$ **then**

      /* *Nothing to do* */

    else if *There is* $n_1 \in N$ *s.t.* $\Lambda(n) \prec \Lambda(n_1)$ **then**

      /* n *being smaller, we can forget it* */

      $N \leftarrow N \setminus \{n\}$;

      Let n' be the direct ancestor of n ;

      $B \leftarrow B \setminus (n', n)$;

    else if *There is* $n_1 \in N$ *s.t.* $\Lambda(n_1) \prec \Lambda(n)$ **then**

      /* *The classical Karp& Miller acceleration* */

      $\mathbf{m} \leftarrow$**Accelerate**(n) ;

      /* *Ancestors are treated first* */

      if *There is an ancestor* n' *of* n *s.t.* $\Lambda(n') \prec m$ **then**

        Let n' be the highest such ancestor in $\mathcal{T}$;

        $\Lambda(n') \leftarrow m$;

        $to\_treat \leftarrow to\_treat \setminus \mathsf{nodes}(\mathsf{subtree}(n'))$;

        Remove $\mathsf{subtree}(n')$ from $\mathcal{T}$ (but keep n') ;

        $to\_treat \leftarrow to\_treat \cup \{n'\}$;

      else $to\_treat \leftarrow to\_treat \cup \{n\}$;

      /* *Then the other nodes that are smaller than* $\mathbf{m}$ */

      foreach $n' \in N$ *s.t.* $\Lambda(n') \prec \mathbf{m}$ **do**

        $to\_treat \leftarrow to\_treat \setminus \mathsf{nodes}(\mathsf{subtree}(n'))$;

        Remove $\mathsf{subtree}(n')$ from $\mathcal{T}$ (including n') ;

    else

      /* *When we couldn't apply reductions, we compute the successors of* n */

      foreach *marking* $\mathbf{m}'$ *successor of* $\Lambda(\mathbf{m})$ **do**

        Let n' be a new node s.t. $\Lambda(n') = \mathbf{m}'$;

        $N \leftarrow N \cup \{n'\}$;

        $B \leftarrow B \cup \{(n, n')\}$;

        $to\_treat \leftarrow to\_treat \cup \{n'\}$;

  Return$(\mathcal{T})$;

end

Petri net. The counter-example relies on the Petri net of Fig. 1. Its analysis by Algorithm 1 is presented at Fig. 2 and 3. It is not difficult to see that not all the reachable markings in the net of Fig. 1 are covered by the labelled tree (Fig.3(b)) obtained at the end of the algorithm.

Let us further comment on these figures:

Step 1 At the first step (Fig. 2(a)), we first compute the three successors of the initial marking, and add them to the labelled tree. Then, the node that is picked up from *to_treat* is $\langle 0,1,0,0,0,0,0\rangle$. We unroll its successors in a branch of the labelled tree by firing t_2 and t_3. Remark that all the markings obtained so far are incomparable. After the development of t_4, we obtain $\langle 0,0,1,0,1,0,0\rangle$ which is strictly greater than its ancestor $\langle 0,0,1,0,0,0,0\rangle$. The **Accelerate** function returns the marking $\langle 0,0,1,0,\omega,0,0\rangle$.

Step 2 Fig. 2(b) shows the labelled tree obtained after: *(i)* having thoroughly applied the acceleration and *(ii)* having successively picked up the nodes $\langle 0,0,0,0,0,1,0\rangle$ and $\langle 0,0,0,1,2,0,0\rangle$ from *to_treat*. One then obtains the marking $\mathbf{m}_1 = \langle 0,0,1,0,3,0,0\rangle$ which is strictly smaller than $\langle 0,0,1,0,\omega,0,0\rangle$. Hence $\mathbf{m}_1$ is removed from the labelled tree and its successors won't be explored any further.

Step 3 At that point, we pick up the node $\langle 0,0,0,0,0,0,1\rangle$ from *to_treat* and compute its unique successor, by firing t_8. This is showed at Fig 2(c). The successor is $\langle 0,1,0,0,1,0,0\rangle$ and is strictly smaller than the previously obtained marking $\mathbf{m}_2 = \langle 0,1,0,0,0,0,0\rangle$. Thus, $\mathbf{m}_2$ and its whole subtree (including the marking resulting from the acceleration) disappear from the labelled tree.

Step 4 $\langle 0,1,0,0,1,1,0\rangle$ is the next marking to be looked at. From this marking, we can compute two successive successors by unrolling the branch labelled $t_2 \cdot t_3$. This is shown at Fig.3(b). We obtain $\langle 0,0,0,1,1,0,0\rangle$ which is strictly smaller than $\langle 0,0,0,1,2,0,0\rangle$, and thus removed from the labelled tree. At that point, the set *to_treat* is empty and the algorithm terminates. The labelled tree computed by the algorithm is shown at Fig.3(b). However, some reachable markings are not covered by any nodes of this labelled tree. Indeed, if it was the case, we could conclude that the place p_5 is bounded, which is obviously not the case: the sequence $t_1 \cdot t_2 \cdot (t_3 \cdot t_4)^n$, which puts n tokens in p_5, can be fired for any $n \geq 0$.

References

[1] A. Finkel. The minimal coverability graph for Petri nets. In *Proceedings of Advances in Petri Nets*, volume 674 of *LNCS*, pages 210–243. Springer, 1993.

[2] R. M. Karp and R. E. Miller. Parallel Program Schemata. *Journal of Computer and System Sciences*, 3:147–195, 1969.

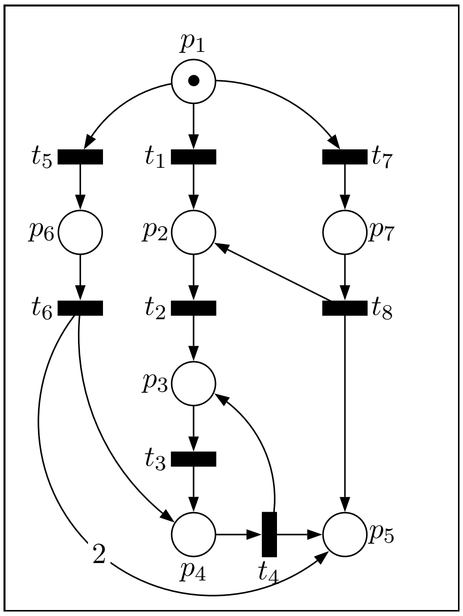
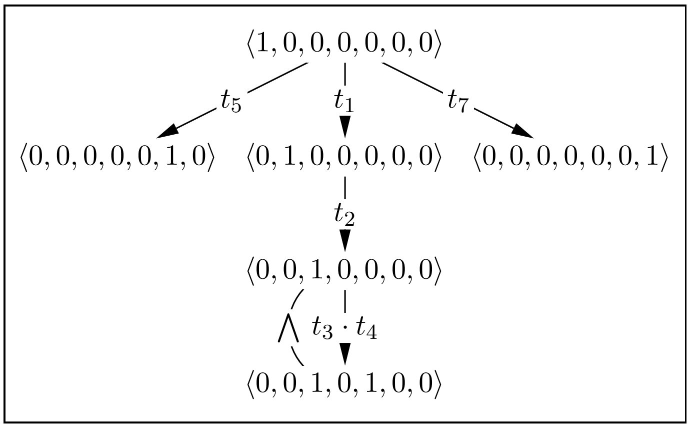
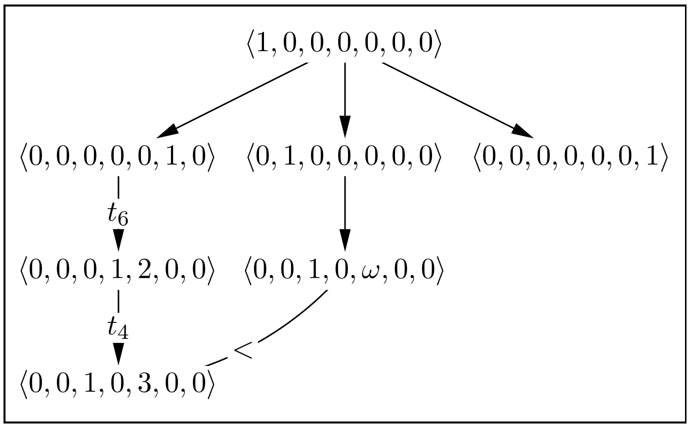


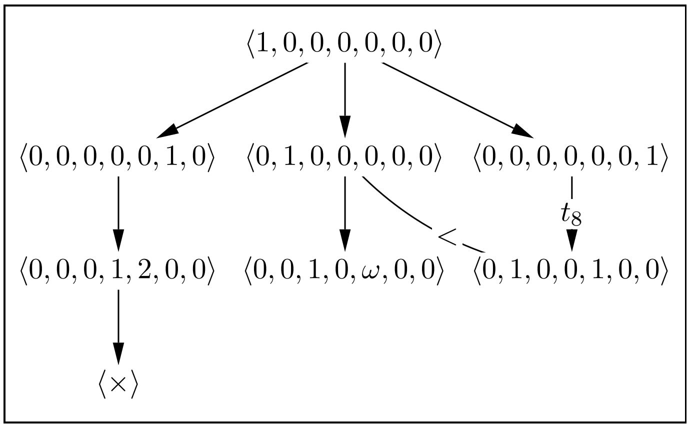
Figure 1: The Petri net on which the algorithm proposed in [1] may not compute the whole coverability set



(a) Step 1

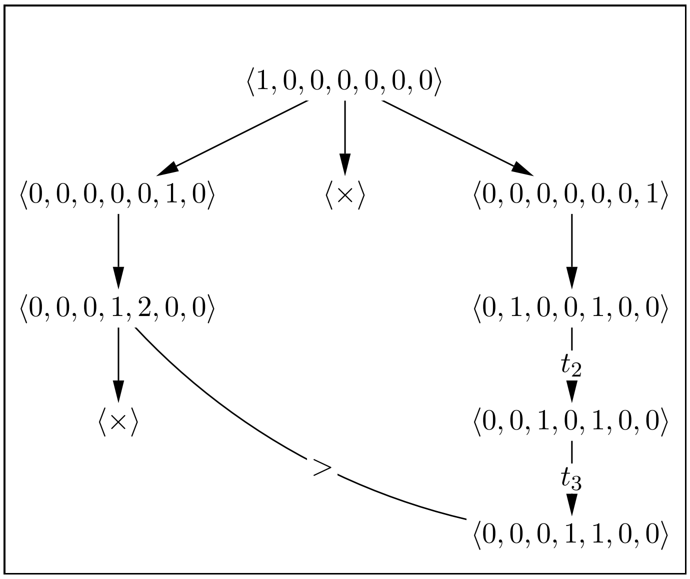


(b) Step 2

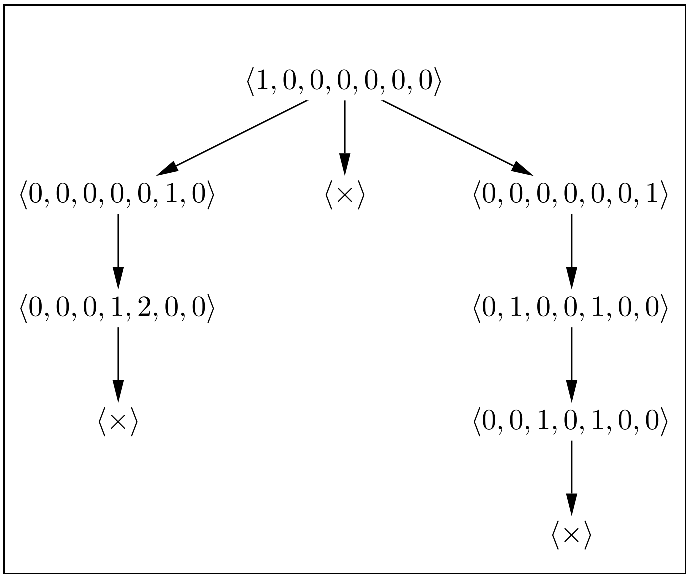


(c) Step 3

Figure 2: A counter-example to Finkel's algorithm



(a) Step 4



(b) The result of the algorithm

Figure 3: A counter-example to Finkel's algorithm (cont'd)