

Old and New Algorithms for Minimal Coverability Sets

Antti Valmari and Henri Hansen

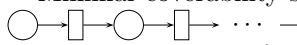
Department of Software Systems, Tampere University of Technology
P.O. Box 553, FI-33101 Tampere, Finland
{antti.valmari,henri.hansen}@tut.fi

Abstract. Many algorithms for computing minimal coverability sets for Petri nets *prune futures*. That is, if a new marking strictly covers an old one, then not just the old marking but also some subset of its subsequent markings is discarded from search. In this publication, a simpler algorithm that lacks future pruning is presented and proven correct. Then its performance is compared with future pruning. It is demonstrated, using examples, that neither approach is systematically better than the other. However, the simple algorithm has some attractive features. It never needs to re-construct pruned parts of the minimal coverability set. If the minimal coverability set is constructed in depth-first or most tokens first order, and if so-called *history merging* is applied, then most of the advantage of future pruning is automatic. Some implementation aspects of minimal coverability set construction are also discussed.

1 Introduction

The set of reachable markings of a finite Petri net is not necessarily finite. However, Karp and Miller [4] showed that an abstracted version of it, the *coverability set*, is always finite. The coverability set constructed by the algorithm of Karp and Miller is not unique. Finkel defined a unique *minimal coverability set* and presented an algorithm for constructing it [1]. Surprisingly, more than a decade later an error was found in his algorithm [2]. This inspired new interest in coverability set algorithms.

Proposals for solving the problem correctly, and more efficiently than the original, have been made [3,6]. Some recent work also exists on incremental construction of coverability graphs, that is, mapping transformations of a Petri net to transformations of its coverability graph [5].

Minimal coverability sets may be huge. For instance, the “linear” Petri net  with n places, $n-1$ tokens in the first place, and no tokens elsewhere, has $(2n-2)!/(n-1)!^2 \approx 2^{2n-2}/\sqrt{\pi(n-1)}$ maximal markings.

The algorithms of [4,1,6] are all based on the idea of building a tree of markings with acceleration or ω -addition, that is, replacing unbounded markings of a place with an ω , based on the history of a newly discovered marking. The two latter algorithms also *prune* the tree by excluding the already constructed descendants of the covered nodes from future exploration, in an attempt to make

the computation more efficient. With this pruning, correctness or termination of the algorithm is jeopardised, and proving either becomes very hard, as is evidenced by the fact that the algorithm of [1] was long thought to be correct.

Pruning takes place when a sequence $M_0, t_1, \dots, t_n, M_n$ of markings and transitions has been found, where each M_i is obtained by firing t_i from M_{i-1} and then possibly adding ω -symbols, and then an M'_0 such that $M_0 < M'_0$ is found. Then M_0 is clearly unnecessary in the coverability set. If $M_0 [t_1 \cdots t_n] M_n$ and $M_0 < M'_0$, then there is an M'_n such that $M'_0 [t_1 \cdots t_n] M'_n$ and $M_n < M'_n$. Inspired by this, future pruning passivates also M_1, \dots, M_n simultaneously with M_0 . However, when ω -symbols are added along the path from M_0 to M_n , it is possible that $M_n = M'_n$. Then it is possible that after M_n has been pruned, it may have to be activated anew. We say that the pruning of M_n is *overeager* if and only if $M'_0 [t_1 \cdots t_n] M_n$.

To provide a firm foundation for the discussion in this publication, in Section 2 we recall the basic facts of minimal coverability sets. Then, in Section 3, we propose an algorithm for creating minimal coverability sets that does not prune futures, and show that it is correct. We also comment on some implementation issues. Section 4 is devoted to a discussion of the order in which the set is constructed.

The simple approach is compared with future pruning in Section 5. We demonstrate that future pruning is vulnerable to having to construct large subsets more than once. Then we prove that if the minimal coverability set is constructed in depth-first order or what we call most tokens first order, and if what we call history merging is applied, then, even without explicit pruning, the algorithm automatically avoids the investigation of transitions from those markings that future pruning would prune but not in the overeager way. The last section presents our conclusions and some measurements.

2 Minimal Coverability Sets

The basic facts of coverability sets are more or less widely known, but their published proofs tend to be unclear and tied to individual algorithms. Therefore, we prove them anew in this section, independently of even the notion of transition. The set of natural numbers (including 0) is denoted with \mathbb{N} . Let P be any finite set. Its elements are called *places*. We start with the notion of markings that may also use ω as the marking of a place. Intuitively, ω denotes unbounded.

Definition 1. *An ω -marking is a function from P to $\mathbb{N} \cup \{\omega\}$. If $n \in \mathbb{N}$, we define $n < \omega$ and $\omega + n = \omega - n = \omega$. Let M and M' be ω -markings. We define that M' covers M and write $M \leq M'$ if and only if $M(p) \leq M'(p)$ for every $p \in P$. Furthermore, M' covers M strictly if and only if $M < M'$, that is, $M \leq M'$ and $M \neq M'$. A (finite or infinite) sequence M_1, M_2, \dots of ω -markings is growing if and only if $M_1 \leq M_2 \leq \dots$ and strictly growing if and only if $M_1 < M_2 < \dots$*

Note that (ordinary) markings are ω -markings. An ω -marking is a marking if and only if it does not contain ω -symbols.

Given any infinite growing sequence M_i of ω -markings and any place p , $M_i(p)$ either reaches a maximum value and stays there or grows without limit. The latter means that for each $n \in \mathbb{N}$, there is an i such that $M_i(p) \geq n$. Therefore, the following notion of the limit of the sequence is well-defined.

Definition 2. Let M_1, M_2, \dots be ω -markings such that $M_1 \leq M_2 \leq \dots$. Their limit is the ω -marking $M = \lim_{i \rightarrow \infty} M_i$ such that for each $p \in P$, either $M(p) = M_i(p) = M_{i+1}(p) = M_{i+2}(p) = \dots$ for some i , or $M(p) = \omega$ and $M_i(p)$ grows without limit as i grows.

Clearly $M_i \leq \lim_{i \rightarrow \infty} M_i$ for each i . The ω -markings in a sequence need not be different from each other. So also the sequence M, M, M, \dots has a limit. It is M . It is also worth pointing out that in this publication, the either-part of the definition does not require that $M(p) < \omega$. Therefore, $M(p) = \omega$ is possible in two ways: either $M_i(p) = \omega$ from some i on, or $M_i(p)$ grows without limit.

The following lemma is an immediate consequence of Definition 2. It says that given an arbitrary finite value, the places marked with ω in the limit may *simultaneously* get at least that value, while the other places get their limit values. In this publication, $M_i(p)$ may also be ω , but then trivially $M_i(p) \geq n$.

Lemma 1. If $M = \lim_{i \rightarrow \infty} M_i$, then for every $n \in \mathbb{N}$ there is an i such that for each $p \in P$, either $M_i(p) = M(p) < \omega$ or $M_i(p) \geq n$ and $M(p) = \omega$.

For convenience, we also define a limit of a set of ω -markings as any limit of any infinite growing sequence of elements of the set. The limit of a set is not necessarily unique. Actually, each element of the set is its limit.

Definition 3. Let \mathcal{M} be a set of ω -markings. Then M is a limit of \mathcal{M} if and only if there are $M_1 \in \mathcal{M}, M_2 \in \mathcal{M}, \dots$ such that $M_1 \leq M_2 \leq \dots$ and $M = \lim_{i \rightarrow \infty} M_i$.

The next lemma follows from Dickson’s lemma, but is easier to prove directly.

Lemma 2. Every infinite sequence of ω -markings has an infinite growing subsequence.

Proof. Let M_i be the sequence and $P = \{p_1, p_2, \dots, p_{|P|}\}$. We show that for each $0 \leq j \leq |P|$, M_i has an infinite subsequence $M_{j,i}$ such that $M_{j,1}(p_k) \leq M_{j,2}(p_k) \leq \dots$ for $1 \leq k \leq j$.

Clearly M_i qualifies as the $M_{0,i}$. Let $j > 0$. If $M_{j-1,i}(p_j)$ only gets a finite number of different values for $i \in \mathbb{N}$, then some value v occurs infinitely many times. We let $M_{j,i}$ be the infinite subsequence obtained by picking those $M_{j-1,i}$ that have $M_{j-1,i}(p_j) = v$. Otherwise $M_{j-1,i}(p_j)$ gets infinitely many different values. Then $M_{j-1,i}$ has an infinite subsequence where $M_{j-1,i}(p_j)$ grows. It qualifies as the $M_{j,i}$.

Finally $M_{|P|,i}$ qualifies as the sequence in the claim of the lemma. □

We are ready to define coverability sets. The idea is that for a given set \mathcal{M} of markings, the ω -markings in its coverability set \mathcal{M}' cover every marking in \mathcal{M} without using bigger ω -markings than necessary. The goal is to get finite coverability sets. However, if $M(p)$ obtains infinitely many different values in \mathcal{M} , then to cover them all with finitely many ω -markings it is necessary to let $M'(p) = \omega$ in at least one $M' \in \mathcal{M}'$. More generally, it may be that many places must simultaneously have $M'(p) = \omega$ to cover some subset of \mathcal{M} with finitely many ω -markings. Part 2 of the definition says that this is the only justification for the introduction of ω -symbols. The concept of antichain is important, because we will later show that a coverability set is minimal if and only if it is an antichain.

Definition 4. Let \mathcal{M} be a set of markings and \mathcal{M}' a set of ω -markings. We define that \mathcal{M}' is a coverability set for \mathcal{M} , if and only if

1. For every $M \in \mathcal{M}$, there is an $M' \in \mathcal{M}'$ such that $M \leq M'$.
2. Each $M' \in \mathcal{M}'$ is a limit of \mathcal{M} .

A coverability set is an antichain, if and only if it does not contain two ω -markings M_1 and M_2 such that $M_1 < M_2$.

Every $M \in \mathcal{M}$ is the limit of the infinite growing sequence M, M, M, \dots . Thus \mathcal{M} is its own coverability set. However, it is not necessarily finite. To prove that each set of markings has a finite coverability set, we first show that the limit of an infinite growing sequence of limits is a limit of the original set.

Lemma 3. Let \mathcal{M} be a set of markings. For each $i > 0$, let M_i be any limit of \mathcal{M} such that $M_1 \leq M_2 \leq \dots$. Then also $\lim_{i \rightarrow \infty} M_i$ is a limit of \mathcal{M} .

Proof. For each i , let $M_{j,i}$ be an infinite growing sequence of elements of \mathcal{M} such that $\lim_{j \rightarrow \infty} M_{j,i} = M_i$. Let $M'_1 = M_{1,1}$. When $i > 1$, let M'_i be the first element of $M_{j,i}$ such that for each $p \in P$, either $M'_i(p) = M_i(p) < \omega$ or $M'_{i-1}(p) \leq M'_i(p) \geq i$ and $M_i(p) = \omega$. It exists by Lemma 1. Furthermore, $M'_{i-1}(p) \leq M_{i-1}(p) \leq M_i(p)$, so if $M'_i(p) = M_i(p)$, then $M'_{i-1}(p) \leq M'_i(p)$. Thus M'_i is an infinite growing sequence of elements of \mathcal{M} .

If $M_i(p) < \omega$ for each i , then $M'_i(p) = M_i(p)$ for each $i > 1$, so $M'_i(p)$ has the same limit as $M_i(p)$. Otherwise, from some value of i on, $M_i(p) = \omega$ and $M'_i(p) \geq i$. Then the limit of $M'_i(p)$ is ω , which is also the limit of $M_i(p)$. As a consequence, $\lim_{i \rightarrow \infty} M'_i = \lim_{i \rightarrow \infty} M_i$. □

We say that an element a of a set A is *maximal*, if and only if there is no $b \in A$ such that $a < b$. Let $[\mathcal{M}]$ denote the set of all limits of \mathcal{M} , and let $\lceil \mathcal{M} \rceil$ denote the set of the maximal elements of $[\mathcal{M}]$. We are ready to prove the central result of this section.

Theorem 1. Each set \mathcal{M} of markings has a coverability set that is an antichain. It is finite and unique. It consists of the maximal elements of the limits of \mathcal{M} .

Proof. Obviously $[\mathcal{M}]$ satisfies part 2 of Definition 4. It also satisfies part 1, because each $M \in \mathcal{M}$ is the limit of M, M, M, \dots

We prove next that for every $M \in [\mathcal{M}]$, there is an $M' \in [\mathcal{M}]$ such that $M \leq M'$. Let $M_{0,1} = M$ and $j = 0$. For each $i > 1$ such that $M_{j,i-1}$ is not maximal in $[\mathcal{M}]$, there is an $M_{j,i} \in [\mathcal{M}]$ such that $M_{j,i-1} < M_{j,i}$. If this sequence ends, then the last $M_{j,i}$ qualifies as the M' . Otherwise, let $M_{j+1,1} = \lim_{i \rightarrow \infty} M_{j,i}$. Clearly $M_{j+1,1} \geq M_{j,1} \geq M$. By Lemma 3, $M_{j+1,1} \in [\mathcal{M}]$. We repeat this reasoning with $j = 1, j = 2$, and so on as long as possible. Because $M_{j,i}$ is strictly growing as i grows, $M_{j+1,1}$ has more ω -symbols than $M_{j,1}$. Therefore, $M_{j,i}$ has at least j ω -symbols. This implies that j cannot grow beyond $|P|$. So a maximal element is eventually encountered.

Thanks to this, $[\mathcal{M}]$ satisfies part 1 of Definition 4. It clearly also satisfies the rest of Definition 4. So an antichain coverability set exists.

If \mathcal{M}' is an infinite coverability set of \mathcal{M} , then it is possible to pick an infinite sequence of distinct elements from \mathcal{M}' . By Lemma 2, it has an infinite growing subsequence M_i . Because all its elements are distinct, we have $M_1 < M_2$. Therefore, infinite coverability sets are not antichains.

It remains to be proven that there are no other antichain coverability sets. We have already ruled out infinite sets, so let \mathcal{M}' be any finite coverability set of \mathcal{M} . Part 2 of Definition 4 yields $\mathcal{M}' \subseteq [\mathcal{M}]$.

For any $M \in [\mathcal{M}]$, let M_i be a sequence of elements of \mathcal{M} whose limit is M . Because \mathcal{M}' is finite, it must cover every M_i only using a finite number of ω -markings. Thus \mathcal{M}' must contain an ω -marking M' that covers infinitely many of M_i . Definition 2 implies that $M \leq M'$. We have $M' \in \mathcal{M}' \subseteq [\mathcal{M}]$, and $M \in [\mathcal{M}]$ makes $M < M'$ impossible. Therefore, $M' = M$.

We have proven that every finite coverability set \mathcal{M}' satisfies $[\mathcal{M}] \subseteq \mathcal{M}' \subseteq [\mathcal{M}]$. If there is an M such that $M \in \mathcal{M}' \setminus [\mathcal{M}]$, then there is an $M' \in [\mathcal{M}] \subseteq \mathcal{M}'$ such that $M < M'$, so \mathcal{M}' is not an antichain. As a conclusion, there is only one antichain coverability set. □

A coverability set is *minimal* if and only if none of its proper subsets is a coverability set. Next we will show that a coverability set is minimal if and only if it is an antichain. This will immediately yield the existence, finiteness, and uniqueness of minimal coverability sets.

Corollary 1. *Each set of markings has precisely one minimal coverability set. It is finite. It is the antichain coverability set.*

Proof. The claims follow by Theorem 1, if we prove that a coverability set is minimal if and only if it is an antichain. It is clear that a coverability set that is not an antichain has a proper subset that is a coverability set, because M_1 could be left out. Consider the antichain coverability set $[\mathcal{M}]$. Because it is finite, it cannot have any infinite set as a proper subset. By the proof of Theorem 1, every finite coverability set has $[\mathcal{M}]$ as a subset. So $[\mathcal{M}]$ cannot have a proper subset that is a coverability set. □

```

1   $F := \{\hat{M}\}; A := \{\hat{M}\}; W := \{\hat{M}\} \times T; \hat{M}.B := \text{nil}$ 
2  while  $W \neq \emptyset$  do
3       $(M, t) :=$  any element of  $W; W := W \setminus \{(M, t)\}$ 
4      if  $\neg M[t]$  then continue
5       $M' :=$  the  $\omega$ -marking such that  $M[t]M'$ 
6      if  $M' \in F$  then continue
7      Add- $\omega(M, M')$ 
8      if  $\omega$  was added then if  $M' \in F$  then continue
9      Cover-check( $M'$ ) // may update  $A$  and  $W$ 
10     if  $M'$  is covered then continue
11      $F := F \cup \{M'\}; A := A \cup \{M'\}; W := W \cup (\{M'\} \times T); M'.B := M$ 

```

Fig. 1. The basic coverability set algorithm

3 Basic Algorithm

In this section we present, discuss, and prove correct the simplest of the algorithms in this paper, and its variant that uses what we call history merging. We believe that they are new.

A *Petri net* is a tuple (P, T, W, \hat{M}) such that $P \cap T = \emptyset$, \hat{M} is a function from P to \mathbb{N} , and W is a function from $(P \times T) \cup (T \times P)$ to \mathbb{N} . For this publication, we assume that P and T are always finite. The elements of P , T , W , and \hat{M} are called *places*, *transitions*, *weights*, and *initial marking* respectively. The firing rule for ω -markings is the same as with markings: $M[t]M'$ if and only if for each $p \in P$, $M(p) \geq W(p, t)$ and $M'(p) = M(p) - W(p, t) + W(t, p)$. Whether or not $M[t]$ holds, is determined by the places for which $M(p) < \omega$. If $M(p) < \omega$, then $M'(p) = M(p) - W(p, t) + W(t, p) < \omega$. If $M(p) = \omega$, then also $M'(p) = \omega$. We define $M[t_1 t_2 \dots t_n]M'$ in the usual way.

Overview. The algorithm is shown in Fig. 1. The ω -markings that have been generated and taken into consideration, are stored in the set F . We call these *found* ω -markings. In our test implementation, F is presented as a hash table. There is a base table of pointers to ω -markings that is indexed by the hash value of the ω -marking. Each ω -marking has a pointer to the next ω -marking in the hash list.

The set of found ω -markings is divided to sets of *active* and *passive* ω -markings. The set of active ω -markings is denoted with A , and passive are those that are in F but not in A . In our test implementation, A is represented by a linked list, maintained by another pointer in the ω -marking structure.

Each ω -marking M' has a *back pointer* $M'.B$ that points to the ω -marking M such that M' was first found by firing a transition from M , except that it points to nowhere in the case of the initial marking. Using the back pointers one can scan the history of M' up to the initial marking.

Finally, W is a *workset* that keeps track of the work to be done. The minimal coverability set can be constructed in many different orders, including breadth-first, depth-first, and what we call “most tokens first”. To model this generality,

	Add- $\omega(M, M')$
1	$last := M; now := M; added := False$
2	repeat
3	if $now < M' \wedge \exists p \in P : now(p) < M'(p) < \omega$ then
4	$added := True; last := now$
5	for each $p \in P$ such that $now(p) < M'(p) < \omega$ do
6	$M'(p) := \omega$
7	if $now.B = nil$ then $now := M$ else $now := now.B$
8	until $now = last$

Fig. 2. The basic version of ω -addition

in Fig. 1, W contains pairs consisting of an ω -marking and a transition. In practice it suffices to store ω -markings instead of pairs. In our test implementation, the workset is a queue, stack, or heap containing pointers to ω -markings, and each ω -marking has an integer attribute *next_tr* containing a number of a transition. If M is in the workset of the implementation, the pairs (M, t) in the W of Fig. 1 are the ones where the number of t is at least $M.next_tr$. When we say that M is in the workset, we mean that $(M, t) \in W$ for some $t \in T$.

Initially the initial marking \hat{M} has been found and is active, and the workset contains \hat{M} paired with every transition. The algorithm runs until the workset is empty. Intuitively, the workset contains the ω -markings which still may contain something of interest for the minimal coverability set. In each iteration of the main loop, the algorithm selects and removes one pair (M, t) from the workset. Then it tries to fire t from M . If t cannot be fired from M , then the main loop rejects the pair and goes to the next pair. This is shown in the figure with the word “continue” that means a jump to the test of the **while**-loop.

If the firing of t from M succeeds, the algorithm checks whether the resulting ω -marking M' has already been found. If found, M' is rejected. Otherwise the operation Add- ω is applied to M' . It adds ω -symbols to M' as justified by the history of M' . We will discuss the operation in more details soon.

If M' was changed by Add- ω , then the algorithm tests again whether the resulting ω -marking has already been found and rejects it if it is. If M' survived or avoided this test, one more test is applied to it. Cover-check(M') finds out if M' is covered by any ω -marking in A . It also removes from A those ω -markings that M' covers strictly. Therefore, A is always the set of maximal found ω -markings. We will soon discuss this in more details. Cover-check also removes from W each pair whose first component was removed from A .

If M' passes all these tests, it is added to the found ω -markings and made active. It is also added to the workset paired with every transition. Its back pointer is made to point to the previous ω -marking.

Add- ω . Add- ω is shown in Fig. 2. It scans the history of M towards the initial marking at least once. It tries to find an ω -marking $M'' = now$ that is strictly covered by M' , in such a way that M' does not yet have ω in all those places where $M''(p) < M'(p)$. Whether such a covered ω -marking was found is recorded

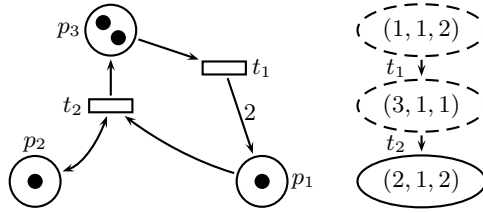


Fig. 3. The advantage of repeated scanning of history

in the boolean variable *added*. The operation then sets $M'(p) = \omega$ for places where previously $M''(p) < M'(p) < \omega$. Then it moves to the next (or perhaps one should say previous) ω -marking, and so on. The operation terminates when it has fully tested the history without being able to add new ω -symbols.

The purpose of repeated scanning of the history is to add as many ω -symbols as possible to M' . For example (see Fig. 3), let $(1, 1, 2) [t_1] (3, 1, 1) [t_2] (2, 1, 2)$. For each $n \in \mathbb{N}$ except 0 and 1 we have $(1, 1, 2) [(t_1 t_2)^{2n-3} t_2^{n-2}] (n, 1, n)$, where σ^i means σ repeated i times. So $(\omega, 1, \omega)$ is a limit of reachable markings. Add- ω checks whether $(2, 1, 2)$ covers $(3, 1, 1)$. It does not. Then it checks whether $(2, 1, 2)$ covers $(1, 1, 2)$. It does, so Add- ω converts $(2, 1, 2)$ to $(\omega, 1, 2)$. Then it has found the end (or beginning) of the history. If it terminated there, the result would be $(\omega, 1, 2)$. However, Add- ω starts anew at $(3, 1, 1)$ and sees that $(\omega, 1, 2)$ covers it. Therefore, it converts $(\omega, 1, 2)$ to $(\omega, 1, \omega)$.

We will later see that inserting an ω -marking to the data structures is an expensive operation. By Corollary 1 and Theorem 1, the algorithm only has to maintain *maximal* ω -markings. Therefore, first inserting $(\omega, 1, 2)$ and later removing it and inserting $(\omega, 1, \omega)$ is disadvantageous compared to just inserting $(\omega, 1, \omega)$. In the worst case, fully testing the history after the last addition of ω doubles the running time of Add- ω , which is a relatively small price.

After each addition of an ω -symbol, scanning is continued from where it was instead of starting anew at M , because intuition suggests that the least recently tried ω -markings have the best chance of success. However, this is not a theorem but a heuristic.

We write $M[t]^\omega M'$ to denote that M' is obtained by firing t from M and then executing Add- $\omega(M, M')$. This notion depends on not only M and t , but also on the history of M .

Cover-Check. The purpose of Cover-check(M') is to ensure that A always consists of the maximal ω -markings in F . In our test implementation, A is represented as a linked list, which is scanned by Cover-check. If it finds an element M''' that is strictly covered by M' , it removes M''' from the list and removes (M''', t) from the workset for every $t \in T$. In our test implementation, the latter is done simply by assigning to M''' .*next_tr* a number that is greater than the number of any transition. Then Cover-check continues scanning.

If Cover-check finds that M' is covered by an element M'' of the list, it terminates immediately, because then it is not possible that M' covers strictly any element in the list. This is because if $M' > M'''$ and M''' is in the list, then $M'' \geq M' > M'''$, so the list does not consist of only maximal elements.

While the test whether a given ω -marking has already been found can be performed very efficiently with hash tables, testing whether a given ω -marking strictly covers any found ω -marking seems much more difficult. We are not aware of essentially better approaches than comparing the new ω -marking one by one to each old ω -marking, with some heuristics to speed the procedure up a little. Therefore, it makes sense to try to reduce the number of times Cover-check is called. It also makes sense to try to keep A small, because the cost of the call is often and at most proportional to the size of A . For both goals, it seems advantageous to get as many ω -symbols as possible to the ω -markings as early as possible. However, this is not a theorem but an intuitive heuristic.

This is also the reason for the presence of F in the algorithm. Correctness does not require it, because Cover-check and A suffice for filtering out later instances of any found ω -marking M . If the earlier instance of M has been removed from A , it happened in favor of some M' that strictly covers M , so the filtering effect remains. However, detecting repeated instances of the same ω -marking with a hash table costs next to nothing, while the cost of the coverability check is significant. Repeated instances of the same ω -marking are common with Petri nets, because if $M[t_1 t_2]M_{12}$ and $M[t_2 t_1]M_{21}$, then $M_{12} = M_{21}$. Therefore, it makes sense to implement a special mechanism for them that is much faster than the general mechanism.

Also Add- ω is costly compared to the test whether $M' \in F$. Performing the test before Add- ω and after each addition of ω -symbols inside Add- ω would speed Add- ω up, but would also globally slow down the adding of ω -symbols, because the new instance of the same ω -marking usually has a different history and may thus introduce ω -symbols to different places. We believe that most calls of Add- ω do not lead to the addition of ω -symbols, and therefore we believe that it is advantageous to test $M' \in F$ before calling Add- ω . On the other hand, if Add- ω has already succeeded in adding an ω -symbol, then the chances of finding a new maximal ω -marking are improved, so it seems better to let it continue. Again, this is a heuristic argument, and we do not really know the actual effect.

History Merging. History merging is a variant of the basic algorithm. In it, $M.B$ is a set of (pointers to) any number of predecessor ω -markings, instead of being a single ω -marking. This mirrors the fact that the same ω -marking may be reached in multiple ways, any of which may justify the addition of ω -symbols.

If M' is rejected on line 6 or 8 of Fig. 1, then it already has a representation in F . In history merging, the program inserts M to the predecessor set of M' . Thus the predecessor set collects pointers to all ω -markings from which M' was reached by firing one transition and possibly executing Add- ω .

Consider the example in Fig. 4. Suppose the algorithm proceeds in a breadth-first manner. It finds the ω -markings $(0, 1, 0, 0)$ and $(0, 0, 1, 0)$, by firing t_1 and t_2 from $(1, 0, 0, 0)$. It then generates $(0, 0, 0, 1)$ by firing t_3 from $(0, 1, 0, 0)$.

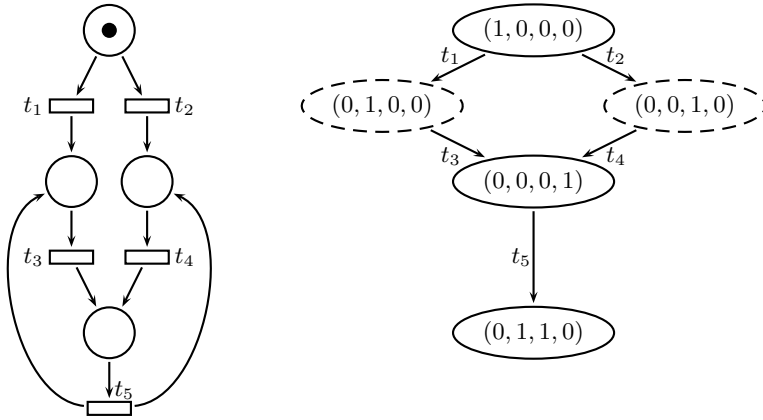


Fig. 4. How history merging helps

The same ω -marking is also found by firing t_4 from $(0, 0, 1, 0)$, at which time we can update the history of $(0, 0, 0, 1)$ to contain both ω -markings. The benefit comes when we add ω -symbols to $(0, 1, 1, 0)$: as both these ω -markings are in its history, we can add two omegas, giving $(0, \omega, \omega, 0)$.

History merging may also be applied on line 10 of Fig. 1, if for each $p \in P$ we have $M''(p) = M'(p)$ or $M''(p) = \omega$, where M'' is the ω -marking that covers M' . If the condition holds in the opposite direction, the histories of those ω -markings that M' is found to strictly cover on line 9, can be merged into the history of M' . In these cases, the changes in the numbers of tokens in those places whose marking is not ω are represented correctly along any path despite the mergings. For the remaining places, the changes do not matter, because their ω -marking is ω also in all later ω -markings. Our test implementation does not yet have this feature.

With history merging, the history of an ω -marking forms a directed graph that is partially shared by the histories of other ω -markings. It can be scanned in time that is linear in its size. After each addition of ω -symbols, our test implementation starts a new scan where it was and continues it at M analogously to Fig. 2, to guarantee that at termination no ω -marking in the history can justify the addition of more ω -symbols to M' .

Repeated scans require repeated resetting of “found” information, which may become a performance problem if implemented naïvely. In our test implementation, each ω -marking has an attribute *search_nr* and there is a global variable *search_now*. When an ω -marking is found, *search_now* is assigned to its *search_nr*. In the beginning of each search, *search_now* is incremented, and if it overflows its type, it is set to 1 and the *search_nr* of every found ω -marking is set to 0 by scanning the hash table that implements F .

Correctness. The correctness of the algorithm consists of four issues, three of which correspond to the three conditions in Definition 4 and the fourth is the termination of the algorithm. We present a lemma for each. In the proofs, we will use the following obvious fact: if $M[t]M'$ and $M \leq M_1$, then there is an M'_1 such that $M_1[t]M'_1$ and $M' \leq M'_1$.

Lemma 4. *After termination, for every reachable marking M of the Petri net, A contains an ω -marking M' such that $M \leq M'$.*

Proof. Each time when an ω -marking is inserted to F , it is also inserted to A . Each time when an M is removed from A , an M' such that $M < M'$ is inserted to A . Therefore, the algorithm maintains the following invariant:

I1: For each $M \in F$, there is an $M' \in A$ such that $M \leq M'$.

Each time when an M is added to A , (M, t) is added to W for every $t \in T$. Each time when a pair (M, t) is removed from W , either $\neg M[t]$, or there is an M' such that $M[t]M'$. In the latter case, the set F either contains an ω -marking M'' such that $M' \leq M''$, or such an M'' is inserted to F . Therefore, the algorithm also maintains the following invariant:

I2: For each $M \in A$ and $t \in T$, either $(M, t) \in W$, $\neg M[t]$, or for the M' such that $M[t]M'$ there is an $M'' \in F$ such that $M' \leq M''$.

Let R be the set of the reachable markings of the Petri net. If $M \in R$, then there is a sequence $M_0 [t_1] M_1 [t_2] \dots [t_n] M_n$ such that $M_0 = \hat{M}$ and $M_n = M$. We prove by induction that for each $0 \leq i \leq n$ there is an $M'_i \in A$ such that $M_i \leq M'_i$. The claim holds for $i = 0$ by I1, because \hat{M} is found initially. After termination $(M'_{i-1}, t_i) \notin W$, because then $W = \emptyset$. We also cannot have $\neg M'_{i-1}[t_i]$, because $M_{i-1}[t_i]M_i$ and $M_{i-1} \leq M'_{i-1}$. Let M'''_i be such that $M'_{i-1}[t_i]M'''_i$. By I2 there is an $M''_i \in F$ and by I1 an $M'_i \in A$ such that $M_i \leq M'''_i \leq M''_i \leq M'_i$. □

Lemma 5. *Every element of F (and thus of A) is a limit of the set of the reachable markings of the Petri net.*

Proof. Let R be the set of the reachable markings of the Petri net.

We show first that if $M[t]M'$ and M is a limit of R , then also M' is a limit of R . We have $M(p) = \omega$ if and only if $M'(p) = \omega$. Let $P_\omega = \{p \in P \mid M(p) = \omega\} = \{p \in P \mid M'(p) = \omega\}$. Let d be the minimum of $W(t, p) - W(p, t)$ over $p \in P_\omega$. By Lemma 1, for every $n \in \mathbb{N}$, there is an $M_i \in R$ such that $W(p, t) \leq M_i(p) \geq n - d$ for every $p \in P_\omega$ and $M_i(p) = M(p)$ for every $p \in P \setminus P_\omega$. We have $M_i[t]$. If M'_i is such that $M_i[t]M'_i$, then $M'_i(p) = M_i(p) - W(p, t) + W(t, p) \geq M_i(p) + d \geq n$ for every $p \in P_\omega$ and $M'_i(p) = M(p) - W(p, t) + W(t, p) = M'(p)$ for every $p \in P \setminus P_\omega$. So M' is the limit of M'_i and thus a limit of R .

We show next that if M' is a limit of R and M'' is the result of applying Add_ω to it, then M'' is a limit of R . Consider any ω -marking $M = \text{now}$ that triggers addition of ω -symbols to M' . Let t_1, \dots, t_k be the transitions from M to M' . Let

d be the minimum of $\sum_{i=1}^k W(t_i, p) - W(p, t_i)$ over $p \in P$. Let e be the maximum of

$\sum_{i=1}^k W(p, t_i)$ over $p \in P$. For each $p \in P$ we have either (1) $M'(p) = M''(p) = \omega$, (2) $M(p) = M'(p) = M''(p) < \omega$, or (3) $M(p) < M'(p) < M''(p) = \omega$.

By Lemma 1, for every $n \in \mathbb{N}$ there is an $M_i \in R$ such that $M_i(p) \geq n(1 - d)$ and $M_i(p) \geq ne$ for every p of kind 1, and $M_i(p) = M'(p)$ for the remaining places. For places of kind 1, ne suffices for firing $t_1 \cdots t_k$ n times in a row starting at M_i , and the result satisfies $M'_i(p) \geq M_i(p) + nd \geq n$. For places of kinds 2 and 3, $t_1 \cdots t_k$ can be fired once from M_i because it was possible to fire it from M . For kind 2, $M_i(p) = M(p) = M'(p) < \omega$, so $t_1 \cdots t_k$ can be fired n times and the result is $M'_i(p) = M_i(p) = M''(p)$. For kind 3, $M(p) < M_i(p) = M'(p) < \omega$, so $t_1 \cdots t_k$ can be fired repeatedly, each time adding at least one token to p . After n repetitions, $M'_i(p) \geq n$. We conclude that $M'' = \lim_{i \rightarrow \infty} M'_i$. Furthermore, $M_i [(t_1 \cdots t_k)^n] M'_i$, so $M'_i \in R$ and M'' is a limit of R .

We have shown that each operation of the algorithm that introduces or modifies ω -markings yields a limit of R , if its input ω -markings are limits of R . Originally there is only the initial marking \hat{M} . It is obviously reachable and the limit of $\hat{M}, \hat{M}, \hat{M}, \dots$. So all ω -markings found by the algorithm are limits of R . \square

Lemma 6. *The set A is always an antichain.*

Proof. This is trivial, because it is explicitly ensured by lines 9 to 11 of Fig. 1, $\{\hat{M}\}$ is an antichain, and no other operation modifies the contents of A . \square

Lemma 7. *The algorithm terminates.*

Proof. Termination of loops other than the main loop of the algorithm and Add- ω are obvious. Add- ω stops adding ω -symbols to M' at the latest when $M'(p) = \omega$ for every $p \in P$, so each call of Add- ω terminates.

The only way in which the main loop of the algorithm gets new work to do is that a new ω -marking is found that is different from all earlier ones. Each old ω -marking and each transition give rise to at most one new ω -marking. Therefore, if the longest acyclic history of any found ω -marking is of length ℓ , then at most $1 + |T| + |T|^2 + \dots + |T|^\ell$ ω -markings are found. This is a finite number.

We conclude that failure of termination requires the existence of an infinite sequence $\hat{M} = M_0 [t_1]^\omega M_1 [t_2]^\omega M_2 [t_3]^\omega \dots$ such that each M_i is first found by firing t_i from M_{i-1} , and then possibly adding ω -symbols with Add- ω . The M_i are distinct because of the tests $M' \in F$. By Lemma 2, M_0, M_1, M_2, \dots has an infinite strictly growing subsequence $M'_0 < M'_1 < M'_2 < \dots$. Thanks to Add- ω , each M'_{i+1} has at least one ω -symbol more than M'_i . However, there are only $|P|$ places, so we run out of places where to add new ω -symbols at $M'_{|P|+1}$, if not earlier. This is a contradiction. So failure of termination is impossible. \square

Together Lemmas 4 to 7 yield the following theorem.

Theorem 2. *The algorithm in Fig. 1 terminates, and then A contains the minimal coverability set of the Petri net.*

```

 $F := \{\hat{M}\}; A := \{\hat{M}\}; Q := \{\hat{M}\}; \hat{M}.B := \text{nil}$ 
while  $Q \neq \emptyset$  do
   $M := \text{the first element of } Q; Q := Q \setminus \{M\}$ 
  for  $t \in T$  do
    if  $M \notin A$  then continue
    lines 4, ..., 10 of Fig 1
     $F := F \cup \{M'\}; A := A \cup \{M'\}; \text{add } M' \text{ to the end of } Q; M'.B := M$ 

```

Fig. 5. Breadth-first discipline

4 Construction Order

The algorithm in the previous section does not specify the order in which the pairs (M, t) are picked from W , and the correctness of the algorithm does not depend on it. In this section we discuss some possible orderings.

The *age* of a pair is defined as the time (for example, the number of iterations of the main loop) elapsed since the pair was inserted to W . The age of (M, t) is determined by M , because the pairs (M, t) for every $t \in T$ are inserted simultaneously to W .

Many state space verification algorithms require traversing the state space in a specific order. With ordinary state spaces, it is customary to construct the state space in that order, so as to enable running the verification algorithm on-the-fly. With coverability sets, however, due to the high cost of unnecessary construction of non-maximal ω -markings, it may be better to construct the set in the order best suited for coverability sets and then, using the minimal coverability set as a starting point, re-generate the transitions in the order required by the verification algorithm.

Breadth-First. Breadth-first discipline is obtained by always picking one of the oldest pairs from W . One possible implementation is described in Fig. 5, where Q (to reflect the fact that it is a queue) is used instead of W of Fig. 1. With this implementation, the attribute *next_tr* is not needed. To save more memory, A and Q can actually be in the same linked list. The list contains first those elements of A that are not in Q , and then the elements of Q . This implementation automatically retains the property $Q \subseteq A$ when an element is removed from A . There are three common pointers: to the beginning, to the beginning of Q , and to the end.

New ω -markings are added to the end, and $Q := Q \setminus \{M\}$ is implemented by moving the middle pointer (the beginning of Q) one step forward.

Let $M.\text{next}_A$ denote the pointer of M that points to the next ω -marking in the list. The test $M \notin A$ can be done in constant time: $M.\text{next}_A$ is made to point to M after M is removed from A . Actually, it would be correct to skip the test, but then the algorithm would unnecessarily fire transitions from ω -markings that are no longer maximal.

From the above we deduce that the breadth-first discipline has a simple and memory-saving implementation. Furthermore, breadth-first is usually more

amenable to parallel implementation than other common disciplines. However, it seems intuitively that it typically adds ω -markings later and thus should have longer running time than other common disciplines. In our measurements (see Section 6), breadth-first was never clearly the fastest but was often clearly the slowest. So we do not recommend breadth-first. Like in many other arguments in this publication, this is a heuristic and not a theorem.

Indeed, it is possible to construct a situation where breadth-first works better than any other approach. Consider an arbitrary Petri net (P, T, W, \hat{M}) , for which the construction of the minimal coverability set takes some considerable time to finish. We add to it one place p_1 and two transitions t and t' in the following way: $W(p, t) = 0$ and $W(t, p) = 1$ for every $p \in P$. $W(p, t') = 0$ and $W(t', p) = \hat{M}(p)$ for $p \in P$, and $W(t', p_1) = 0$. $W(p_1, t) = W(p_1, t') = W(t, p_1) = 1$. A new initial marking \hat{M}' is such that $\hat{M}'(p_1) = 1$, and all other places are empty. The ordering of transitions is such that t' is the first transition to be fired, and t is the second.

Now, breadth-first fires t as the second transition from the initial marking, resulting in (ω, \dots, ω) and quick termination of the algorithm. With many other disciplines, such as depth-first, the algorithm fires t' , which “primes” the original Petri net, after which the algorithm runs its course exactly as with the original Petri net. It explores t only as the very last transition.

Depth-First. Depth-first discipline is obtained by always picking a youngest pair from W . It can be implemented by storing the ω -markings of the pairs in a stack; returning $(M, M.next_tr)$ as the pair, where M is the top element of the stack; and popping the top element when it has no more unused transitions.

Depth-first also has a well-known recursive implementation. It has the advantage that $next_tr$ is not needed, making it conceptually simpler. On the other hand, each recursion level consumes some memory, and the recursive calls consume some time. Therefore, the recursive implementation is likely to be at least marginally less efficient.

Intuitively, depth-first typically adds ω -markings early on, because of the following result. Thus it should have a good running time. In our measurements it was seldom the fastest and seldom much worse than the fastest.

We say that M' is a *successor* of M if and only if there is a t such that the algorithm at some point fires $M[t]^\omega M'$ and either puts M' into F or detects that it is there already. A *descendant* of an ω -marking is the ω -marking itself, its successor, or a descendant of its successor.

Lemma 8. *If the construction order is depth-first and M has more ω -symbols than the ω -marking via which it was first found, then the algorithm will not backtrack from M before it has investigated all descendants of M .*

Proof. Let an ω -marking be *black*, if it has been backtracked from; *grey*, if it has been found but not yet backtracked from; and *white*, if it has not been found. Depth-first search has the property that the set of grey ω -markings and the transitions via which they were first found, constitute a path from the initial

marking to the current ω -marking. We call any contiguous sub-path of this path a *grey path*.

Each black ω -marking has been removed from W . So the successors of any black ω -marking are grey or black. A black ω -marking may have white descendants, but each path to any of them goes via at least one grey ω -marking.

If M_2 is a descendant of M_1 , then M_2 has ω -symbols in at least the same places as M_1 . Assume that the algorithm is about to backtrack from M to M' , where M has more ω -symbols than M' . Then no descendant of M can be along the grey path from \hat{M} to M' . Thus none of them can be grey, implying that none of them can be white either. Hence, they are all black. \square

Most Tokens First. The desire to add ω -symbols as early as possible naturally leads to the heuristic of always trying next the ω -marking that has the most tokens. The ω -marking with the maximal number of ω -symbols is preferred, and if it is not unique, then the total number of tokens in the places whose marking is not ω is used as the criterion. Like before, only ω -markings are stored in the workset, and *next_tr* is used to get the transition component of the pair (M, t) . If the workset is implemented as a heap and contains w ω -markings, then each operation on it takes $O(\log w)$ time.

In our measurements, this discipline was often both the fastest and constructed the smallest number of ω -markings. It lost to depth-first a small number of times, and often there was no clear difference. It may be remarkable that it lost in the biggest example. However, our set of measurements is far too small for firm conclusions.

5 To Prune or Not to Prune

In this section we discuss pruning of active ω -markings and whether it is better than the algorithm in Section 3.

Pumping Cycle Passivation. Consider M_0 in the history of M_n such that M_0 triggered the addition of at least one ω -symbol to M_n along the path $M_0 [t_1]^\omega M_1 [t_2]^\omega \dots [t_n]^\omega M_n$. Then $M_0 < M_n$ and $M_n [t_1 \dots t_n]$. When $0 \leq i \leq n$, let M'_i be the ω -marking such that $M_n [t_1 \dots t_i] M'_i$. Clearly $M_i < M'_i$ for each $0 \leq i < n$. So eventually M_0, \dots, M_{n-1} will not be maximal. The firing of those transitions from them that have not already been fired seems wasted work.

Therefore, it seems a good idea to passivate or remove M_0, \dots, M_{n-1} altogether, when ω -symbols are added to M_n . By *passivation* we mean the removal from W and A , but not from F . (The removal of M from W means the removal of (M, t) from W for every $t \in T$.) By *removal* we mean the removal from all data structures. The algorithm in [1] removes M_0, \dots, M_{n-1} .

We argue that the removal of M_0, \dots, M_{n-1} is not a good idea in general. Firstly, keeping them in F costs very little, but prevents the algorithm from constructing their futures, if they are constructed anew. As we have pointed out, reaching the same marking many times is common with Petri nets.

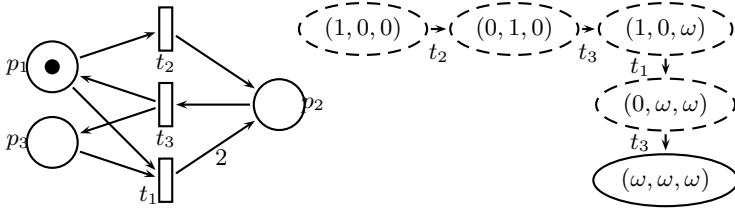


Fig. 6. A pumping ω example

Secondly, the removal may slow down the addition of ω -symbols. Consider the example in Fig. 6, assuming depth-first discipline. The algorithm in Fig. 1 fires $(1, 0, 0) [t_2] (0, 1, 0) [t_3] (1, 0, 1)$ and converts $(1, 0, 1)$ to $(1, 0, \omega)$. Then it fires $(1, 0, \omega) [t_1] (0, 2, \omega)$ and converts $(0, 2, \omega)$ to $(0, \omega, \omega)$, because it covers $(0, 1, 0)$. Finally $(0, \omega, \omega) [t_3] (1, \omega, \omega) > (0, \omega, \omega)$, yielding (ω, ω, ω) . Transitions were fired altogether four times. However, if M_0, \dots, M_{n-1} are removed, then $(1, 0, 0)$ and $(0, 1, 0)$ are removed after constructing $(1, 0, \omega)$. Next $(1, 0, \omega) [t_1] (0, 2, \omega) [t_3] (1, 1, \omega)$ are fired, yielding $(1, \omega, \omega)$. Again, all other ω -markings are removed. Firing t_1 and t_2 from $(1, \omega, \omega)$ do not yield new maximal ω -markings, but t_3 yields (ω, ω, ω) . So seven transition firings were needed.

Future Pruning. Pruning of futures refers to the passivation or removal of some or all found ω -markings whose histories contain an ω -marking that was strictly covered by a newly found ω -marking. Pumping cycle passivation can be considered as a special case of future pruning. The algorithms in [1] and [6] both perform some more general form of future pruning.

Correctness of future pruning is tricky, and not all forms are correct. The counter-example presented in [2] reveals a flaw in the future pruning of the algorithm in [1]. In the counter-example, an ω -marking M_1 first triggers pumping cycle removal. Then another ω -marking M_2 with a different history is found, and its successor ω -markings are covered by M_1 . Therefore, M_2 remains active but does not lead to any new ω -markings. An ω -marking in the (removed) pumping cycle is covered by M_2 , but the algorithm fails to notice this, since the cycle's ω -markings have been removed. Finally, a third ω -marking M_3 is found that covers strictly some ω -marking in the history of M_1 , and M_1 is removed. The firing of transitions from M_3 leads to an ω -marking that is covered by M_2 , and exploration stops short of finding an ω -marking that covers M_1 .

In [6], the algorithm never removes, only passivates ω -markings. The presence of these passive ω -markings in the histories of active ω -markings means that pruning happens differently from [1]. When a pumping cycle is found, the intermediate ω -markings are passivated, but they remain in the history of the new ω -marking. Whenever a new ω -marking M covers some ω -marking not in its own history, the whole branch starting from that ω -marking is passivated, even if the covered ω -marking is passive.

This avoids the behaviour described above. When M_1 in the counter-example triggers pumping cycle passivation, the intermediate ω -markings remain in a tree. On the way to M_2 the algorithm encounters another ω -marking, M , that covers a passive ancestor of M_1 . The algorithm passivates the branch of M_1 when adding M , so by the time it gets to M_2 , M_1 is no longer active, and the search will continue from M_2 .

Unfortunately, this technique requires checking whether the new ω -marking M strictly covers any element in F (excluding the history of M). This is a disadvantage, because otherwise checking coverage against A would suffice, A may be much smaller than F , and checking coverage is expensive.

Is It Worth the Effort? Our first observation is that the running time may depend heavily on finding a certain ω -marking early on. By exploiting this, it is possible to design Petri nets so that either algorithm is faster in the particular case. This is illustrated by the tables below. The arc weights are shown in the table in the format $-W(p, t), W(t, p)$. The initial marking is $(1, 0, 0)$. We consider the most tokens first order, and at the same ω -marking, transitions are tried in the numeric order. (We have designed a similar example for depth-first order.) Like in [6] and unlike in Section 3, we assume that only active ω -markings are taken into account in Add- ω .

	t_1	t_2	t_3	t_4	t_5			t_1	t_2	t_3	t_4	t_5
p_1	-1, 2	-1, 0	-0, 1	-0, 0	-0, 1	p_1		-1, 2	-1, 0	-0, 1	-0, 1	-0, 0
p_2	-0, 1	-0, 2	-2, 0	-1, 1	-0, 0	p_2		-0, 0	-0, 2	-2, 0	-1, 2	-1, 1
p_3	-1, 0	-0, 0	-1, 0	-0, 1	-1, 1	p_3		-1, 0	-0, 0	-1, 0	-1, 0	-0, 1

With the first Petri net, both algorithms fire first $(1, 0, 0) [t_2]^\omega (0, 2, 0) [t_4]^\omega (0, 2, \omega) [t_3]^\omega (1, 0, \omega)$ and passivate at least $(0, 2, 0)$ and $(1, 0, 0)$. The pruning algorithm also passivates $(0, 2, \omega)$ simultaneously with $(1, 0, 0)$. Then it fires $(1, 0, \omega) [t_1]^\omega (\omega, \omega, \omega)$, passivates all other ω -markings, fires $(\omega, \omega, \omega) [t_i]^\omega (\omega, \omega, \omega)$ for $1 \leq i \leq 5$, and terminates. The non-pruning algorithm continues with $(0, 2, \omega)$, because it has more tokens than $(1, 0, \omega)$. It fires $(0, 2, \omega) [t_4]^\omega (0, 2, \omega)$ and $(0, 2, \omega) [t_5]^\omega (\omega, 2, \omega) [t_1]^\omega (\omega, \omega, \omega)$, fires each t_i , and terminates. So the pruning algorithm is faster. (Thanks to $(1, 0, 0)$, the Add- ω in Section 3 would have yielded $(0, 2, \omega) [t_5]^\omega (\omega, \omega, \omega)$.)

With the second Petri net, both algorithms fire first $(1, 0, 0) [t_2]^\omega (0, 2, 0) [t_5]^\omega (0, 2, \omega) [t_3]^\omega (1, 0, \omega)$. The non-pruning algorithm continues $(0, 2, \omega) [t_4]^\omega (\omega, \omega, \omega)$, while the pruning algorithm fires $(1, 0, \omega) [t_1]^\omega (\omega, 0, \omega) [t_1]^\omega (\omega, 0, \omega) [t_2]^\omega (\omega, \omega, \omega)$. So with this Petri net, the non-pruning algorithm is faster.

Our second observation is that the pruning algorithm may activate the same ω -marking more than once, leading to repeated work. To illustrate this, let the t_1 of the second Petri net be replaced by a transition that takes two tokens from each of p_2 and p_3 , and puts three tokens to a new place p_4 . There is also a transition t_6 that moves a token from p_4 to p_5 . After constructing $(0, 2, \omega, 0, 0)$, the algorithm fires $(0, 2, \omega, 0, 0) [t_1 t_6 t_6 t_6]^\omega (0, 0, \omega, 0, 3)$. Then it fires $(0, 2, \omega, 0, 0) [t_3]^\omega (1, 0, \omega, 0, 0)$, notices that $(1, 0, 0, 0, 0)$ is covered, and passivates all ω -markings

other than $(1, 0, \omega, 0, 0)$. Next it fires $[t_2]^\omega$, activating $(0, 2, \omega, 0, 0)$ again. Then it fires $(0, 2, \omega, 0, 0) [t_1 t_6 t_6 t_6]^\omega (0, 0, \omega, 0, 3)$ for a second time.

The goal of pruning is to avoid unnecessarily investigating ω -markings that will later be strictly covered by other ω -markings. Fortunately, the following theorem says that if the construction order is depth-first and history merging is applied, this happens automatically, without any explicit future pruning.

Theorem 3. *Let the construction order be depth-first and history merging be applied. Assume that $M_0 [t_1 \cdots t_n]^\omega M_n$ and $M_0 < M'_0$. Assume that all transitions along the path $M_0 [t_1 \cdots t_n]^\omega M_n$ were found before M'_0 . After finding M'_0 , the algorithm will not fire transitions from M_n , unless $M'_0 [t_1 \cdots t_n] M_n$.*

Proof. Let M_1, \dots, M_{n-1} be defined in the obvious way. Consider the moment when M'_0 has just been found. If M_n is black, then it has no more transitions to fire. From now on we assume that M_n is grey.

There is a grey path from M_n to the newest ω -marking, that is, M'_0 . We denote its transitions and ω -markings with t_{n+1}, \dots, t_m and M_{n+1}, \dots, M_m , where $M_m = M'_0$. We have $M_0 [t_1 \cdots t_n]^\omega M_n [t_{n+1} \cdots t_m]^\omega M'_0$, and $M_0 < M'_0$. Thanks to Add- ω , for each $p \in P$, $M'_0(p) = M(p)$ or $M'_0(p) = \omega$. In particular, M'_0 has more ω -symbols than M_0 .

Along any path, the marking of any place may change from finite to ω but not vice versa. Let M'' be the first ω -marking along the grey path that has ω -symbols in precisely the same places as M'_0 . By Lemma 8, the algorithm will not backtrack from M'' before it has investigated all descendants of M'' . Therefore, currently the only investigated transition from any non-descendant of M'' to any descendant of M'' is the transition via which M'' was first found. Because also the path $M_0 [t_1 \cdots t_m]^\omega M'_0$ has such a transition, it must be the same transition. So $M'' = M_h$ for some $0 < h \leq m$.

Let M''_n be the ω -marking such that $M'_0 [t_1 \cdots t_n] M''_n$. The algorithm will not backtrack from M_h before it has found an M''_n that covers M''_n . If $n < h$, then M''_n is found before backtracking to M_n . Furthermore, $M_n < M''_n$, because M'_0 has ω -symbols in the same places as M_n and in at least one more place. So the algorithm passivates M_n by direct coverage before backtracking to it.

If $n \geq h$, then M_n has ω -symbols in precisely the same places as M'_0 . Also M''_n has ω -symbols in precisely the same places, because it was defined using “[\dots]” instead of “[\dots] $^\omega$ ”. For the remaining places, $M_0(p) = M'_0(p)$, so also $M_n(p) = M''_n(p)$. We conclude that $M''_n = M_n$, implying $M'_0 [t_1 \cdots t_n] M_n$. \square

We prove a similar theorem for most tokens first search.

Theorem 4. *Let the construction order be most tokens first and history merging be applied. Assume that $M_0 [t_1 \cdots t_n]^\omega M_n$ and $M_0 < M'_0$. Assume that all transitions along the path $M_0 [t_1 \cdots t_n]^\omega M_n$ were found before M'_0 . After finding M'_0 , the algorithm will not fire transitions from M_n , unless $M'_0 [t_1 \cdots t_n] M_n$.*

Proof. Let M_1, \dots, M_{n-1} be defined in the obvious way. Let $M \prec M'$ denote that M has fewer ω -places than M' , or the same number of ω -places but altogether fewer tokens in the remaining places than M' . Then $M < M'$ implies

$M \prec M'$. Also note that along any path, ω -symbols may be introduced but cannot disappear. Let $A(M)$ denote any M'' such that $M \preceq M''$ and $M'' \in A$.

If $M_n \preceq M_i$ for each $0 \leq i \leq n$, then M_n has ω -symbols in precisely the same places as M_0 . Furthermore, $M_n \preceq M_0 \prec M'_0 \preceq A(M'_0)$, so $A(M'_0)[t_1]^\omega M'_1$ is fired before firing transitions from M_n . Because $M_0 \prec M'_0$ and ω -symbols were not added during $M_0 [t_1]^\omega M_1$, we have $M_n \preceq M_1 \prec M'_1$. The reasoning continues until we get $M_n \prec M'_n$. Then M_n is passivated by coverage.

In the opposite case, let i be maximal such that $M_{i-1} \prec M_n$. So $M_n \preceq M_j$ for $i \leq j \leq n$. If any of M_i, \dots, M_n has been found before $M_{i-1}[t_i]^\omega M_i$ is fired, then all transitions from such an ω -marking and eventually all transitions from M_n are fired before $M_{i-1}[t_i]^\omega M_i$. In that case, the algorithm never fires any transitions from M_n after finding M'_0 , simply because it already has fired them all. The same happens if any M that is investigated after firing $M_{i-1}[t_i]^\omega M_i$ but before M'_0 is found has $M \prec M_n$.

The case remains where $M_{i-1} \prec M_n \preceq M_j$ for $i \leq j \leq n$, none of the M_j is found before firing $M_{i-1}[t_i]^\omega M_i$, and (1) from then on every ω -marking M investigated had $M_n \preceq M$ until M'_0 is found.

Because M'_0 is not found before completing the path, the finding history of M'_0 has some $M'_{h-1} [t'_h]^\omega M'_h$ (where $h \leq 0$) such that M'_{h-1} (but not M'_h) has been found when $M_{i-1} [t_i]^\omega M_i$ is fired. This implies $M'_{h-1} \preceq M_{i-1} \prec M_n$. By (1), $M'_{h-1} [t'_h]^\omega M'_h$ cannot be fired after $M_{i-1} [t_i]^\omega M_i$ until M'_0 is found. The remaining possibility is that $M'_{h-1} [t'_h]^\omega M'_h$ is the same transition as $M_{i-1} [t_i]^\omega M_i$. This implies that $M_0 [t_1 \cdots t_i]^\omega M_i [t'_{h+1} \cdots t'_0]^\omega M'_0$. Add- ω guarantees that for each $p \in P$, either $M_0(p) = M'_0(p)$ or $M'_0(p) = \omega$.

By $M_n \preceq M_i$, M_n and M_i have ω -symbols in precisely the same places. Therefore, M'_0 has ω -symbols in at least the same places as M_n .

If M'_0 has more ω -symbols than M_n , then the same holds for all M'_i along the path $M'_0 [t_1 \cdots t_n]^\omega M'_n$, and we have $M_n \prec M'_i \preceq A(M'_i)$. Therefore, all the transitions corresponding to $A(M'_0) [t_1 \cdots t_n]^\omega A(M'_n)$ are fired before firing any transition from M_n , and M_n is passivated before being investigated further.

Otherwise, M'_0 and M_n have ω -symbols in precisely the same places. This implies $M'_0 [t_1 \cdots t_n]^\omega M_n$, because $M_0(p) = M'_0(p)$ if $M'_0(p) < \omega$. □

6 Conclusion

We have given a simple algorithm for calculating minimal coverability sets. Furthermore, we have given arguments that lead us to believe that published more complicated algorithms are in general no more efficient.

Using examples, we have demonstrated that by tailoring the incoming Petri net in a suitable way, almost any algorithm can be made terminate much quicker for that particular Petri net than its competitors. Therefore, there probably cannot be any theorem that one algorithm is *systematically* better, or even as good as, another. However, we proved two theorems saying that certain versions of the simple non-pruning algorithm automatically have the benefits that pruning tries to achieve. At the same time, the simple algorithm does not run the risk of repeating work on identical ω -markings, like pruning algorithms do. We also

Table 1. Some measurements with test data from [3]

model	$ A $	most tokens f.		depth-first		breadth-first		[6]
fms	24	63	53	110	56	421	139	809
kanban	1	12	12	12	12	12	12	114
mesh2x2	256	479	465	774	455	10733	2977	6241
mesh3x2	6400	11495	11485	8573	10394			
multipoll	220	245	234	244	244	507	507	2004
pncsacover	80	215	246	284	325	7122	5804	1604

pointed out that it may be advantageous to add as many ω -symbols as early as possible, and presented techniques towards such a goal.

Table 1 shows results of the six biggest test runs that we have made with the test set from [3]. The second column shows the size of the minimal coverability set. The other numbers are the total numbers of constructed distinct ω -markings, that is, $|F|$. The running time was always below 0.1 s except with mesh3x2 (30 s, 29 s, 8 s, 9 s) and, in the case of breadth-first search also mesh2x2 (0.7 s, 0.1 s) and pncsacover (0.3 s, 0.3 s). These times should not be compared to those in [3,6], because we used a different computer and programming language (C++).

We ran each experiment with transitions tried in the order that they were given in the input and in the opposite order. As the table shows, this low-level difference had sometimes a dramatic impact on the result. This acts as a warning that numbers like the ones in the table are much less reliable than we would like.

History merging was applied on lines 6 and 8 of Fig. 1. Switching it off had very little effect on $|F|$ except with breadth-first search.

We leave further analysis and measurements as potential future work.

Acknowledgements. We would like to thank Prof. Mikko Tiisanen and the reviewers for their help, and the authors of [6] for providing help for the experiments.

References

1. Finkel, A.: The Minimal Coverability Graph for Petri Nets. In: Rozenberg, G. (ed.) APN 1993. LNCS, vol. 674, pp. 210–243. Springer, Heidelberg (1993)
2. Finkel, A., Geeraerts, G., Raskin, J.-F., Van Begin, L.: A counter-example to the minimal coverability tree algorithm. Technical Report 535, Universite Libre de Bruxelles (2005)
3. Geeraerts, G., Raskin, J.-F., Van Begin, L.: On the efficient computation of the minimal coverability set of Petri nets. *International Journal of Foundations of Computer Science* 21(2), 135–165 (2010)
4. Karp, R.M., Miller, R.E.: Parallel program schemata. *Journal of Computer and System Sciences* 3(2), 147–195 (1969)
5. König, B., Koziura, V.: Incremental construction of coverability graphs. *Information Processing Letters* 103(5), 203–209 (2007)
6. Reynier, P.-A., Servais, F.: Minimal Coverability Set for Petri Nets: Karp and Miller Algorithm with Pruning. In: Kristensen, L.M., Petrucci, L. (eds.) *PETRI NETS 2011*. LNCS, vol. 6709, pp. 69–88. Springer, Heidelberg (2011)