*IOS Press*
This is the last draft version. The final version appeared as:
Fundamenta Informaticae 131(1): 1–25 (2014)

# Old and New Algorithms for Minimal Coverability Sets

**Antti Valmari**
*Department of Mathematics, Tampere University of Technology*
*PO Box 553, FI-33101 Tampere, FINLAND*
*antti.valmari@tut.fi*

**Henri Hansen**[*]
*Temasek Laboratories, National University of Singapore*
*henri.hansen@gmail.com*

**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 successor markings is discarded from search. In this publication, a simpler algorithm that lacks future pruning is presented and proven correct. 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. It automatically gives most of the advantage of future pruning, if the minimal coverability set is constructed in depth-first or most tokens first order, and if so-called *history merging* is applied. Some implementation aspects of minimal coverability set construction are also discussed. Some measurements are given to demonstrate the effect of construction order and other implementation aspects.

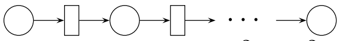
## 1. Introduction

The set of reachable markings of a finite Petri net is not necessarily finite. Karp and Miller [8], however, showed that an abstracted version of the set, called the *coverability set*, is always finite. Finkel has also shown that the algorithm of Karp and Miller can be generalized to well structured transition systems [2]. The literature also talks about a *coverability graph*, which is like the reachability graph. The vertices of a coverability graph are the elements of the coverability set, and its edges $(M, t, M')$ correspond to

[*]Formerly of Department of Software Systems, Tampere University of Technology

occurrences of transitions $M\,[t\rangle\,M''$ such that $M'$ covers $M''$. A *coverability tree* is like a coverability graph, but different instances of the same marking are not fused together. The algorithm of Karp and Miller constructs the coverability set by constructing a coverability tree, and this set is not unique. Finkel defined the *minimal coverability set*, which is unique, and presented an algorithm for constructing it [3]. Surprisingly, more than a decade later an error was found in his algorithm [4]. This inspired new interest in coverability set algorithms.

Proposals for solving the problem correctly, and more efficiently than the original, have been made [7, 10]. 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 [9].

A minimal coverability set may be huge, even if it precisely coincides with the set of reachable markings. As a simple example, consider the "linear" Petri net  with $n$ places, $n-1$ tokens in the first place, and no tokens elsewhere. It has $(2n-2)!/(n-1)!^2 \approx 2^{2n-2}/\sqrt{\pi(n-1)}$ maximal markings. The first formula is obtained by presenting the marking as the string with first $M(p_1)$ $\bullet$'s, then a $|$, then $M(p_2)$ $\bullet$'s, and so on. There are altogether $n-1$ $\bullet$'s and $n-1$ $|$'s, and their orderings map 1-to-1 to the reachable markings. So there are as many reachable markings as there are different ways of picking $n-1$ positions from $2n-2$ possibilities. The second formula is obtained with Stirling's approximation.

The algorithms of [8, 3, 10] are all based on the idea of building a tree of markings with *acceleration* or *$\omega$-addition*, that is, replacing unbounded markings of a place with an $\omega$, 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 and termination of the algorithm are jeopardized. Proving them becomes very hard, as is evidenced by the fact that the algorithm of [3] was long thought to be correct.

Pruning takes place when a sequence $M_0, t_1, \ldots, t_n, M_n$ 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 $\omega$-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 $M_0\,[t_1 \cdots t_n\rangle\,M_n$ and $M_0 < M_0'$, then there is an $M_n'$ such that $M_0'\,[t_1 \cdots t_n\rangle\,M_n'$ and $M_n < M_n'$. Inspired by this, future pruning passivates also $M_1, \ldots, M_n$ simultaneously with $M_0$. However, when $\omega$-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 $M_0'\,[t_1 \cdots t_n\rangle\,M_n$.

In this article, the goal is to construct the minimal coverability set without use of pruning. Our approach is more like constructing a graph than a tree, and our algorithm can then make use of this fact by considering a marking as having more than one history. This publication is an extension of [11], with the following major new contributions. The treatment of history merging in Section 3.5 has been significantly extended. Among other things, an advanced mode that was mentioned as an idea in [11] has now been elaborated and experimented with. A heuristic ordering of transitions is explored in Section 4.5, the idea is to first try those transitions that have the greatest potential for increasing the total number of tokens. The experimental section is also much more extensive.

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 without being overeager. The last section presents some measurements and our conclusions.

## 2. Minimal Coverablity Sets

The basic facts of coverability sets are more or less widely known, but some of the published proofs are unclear or tied to individual algorithms. Therefore, we prove them anew in this section. We prove them before introducing the notion of transition. This emphasizes that the uniqueness and finiteness of the minimal coverability set are properties of sets of vectors, and it does not matter whether the vectors are ($\omega$-extended) markings of a Petri net or something else.

The set of natural numbers (including $0$) is denoted with $\mathbb{N}$, and the set of positive integers (starting from $1$) is denoted with $\mathbb{Z}^+$. Let $P$ be any finite set. In this section, $P$ is the set of the indices of the vectors, and the only thing that matters is that it is finite. However, to anticipate its meaning in later sections, we call its elements *places*. We start with the notion of markings that may also use $\omega$ as the marking of a place. Intuitively, $\omega$ denotes unbounded.

**Definition 2.1.** An $\omega$-*marking* is a function from $P$ to $\mathbb{N} \cup \{\omega\}$. If $n \in \mathbb{N}$, we define $n < \omega$ and $\omega + n = \omega - n = \omega$. Having "$<$", the relations "$\leq$", "$>$", and "$\geq$" are defined in the usual way. Let $M$ and $M'$ be $\omega$-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 $M_1, M_2, \ldots$ of $\omega$-markings is denoted $(M_i)_{i \in \mathbb{Z}^+}$. The sequence is *growing* if and only if $M_1 \leq M_2 \leq \ldots$ and *strictly growing* if and only if $M_1 < M_2 < \ldots$.

Please note that (ordinary) markings are $\omega$-markings. An $\omega$-marking is a marking if and only if it does not contain $\omega$-symbols.

Given any infinite growing sequence $(M_i)_{i \in \mathbb{Z}^+}$ of $\omega$-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 2.2.** Let $M_1$, $M_2$, …be a growing sequence of $\omega$-markings. The *limit* of the sequence is the $\omega$-marking $M = \lim_{i \to \infty} M_i$ such that for each $p \in P$, either $M(p) = M_i(p) = M_{i+1}(p) = M_{i+2}(p) = \ldots$ for some $i$, or $M(p) = \omega$ and $M_i(p)$ grows without limit as $i$ grows.

Clearly $M_i \leq \lim_{i \to \infty} M_i$ for each $i$. The $\omega$-markings in the sequence need not be different from each other. So also the sequence $M, M, M, \ldots$ 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 2.2. It says that given an arbitrary finite value, the places marked with $\omega$ in the limit may *simultaneously* get at least that value, while the other places get their limit values. This holds even if $M_i(p) = \omega$, because then trivially $M_i(p) \geq n$.

**Lemma 2.1.** If $M = \lim_{i\to\infty} M_i$, 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 $M(p) = \omega$.

**Proof:**
For each $p \in P$, there is a smallest $i$ such that the claim holds for $p$. We denote it with $i_p$. Because the sequence is growing, the claim for $p$ remains valid if $i > i_p$. The lemma follows by letting $i$ be the maximum of $i_p$ over $p \in P$. □

For convenience, we also define a limit of a set of $\omega$-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 2.3.** Let $\mathcal{M}$ be a set of $\omega$-markings. Then $M$ is a *limit* of $\mathcal{M}$ if and only if there are $M_1 \in \mathcal{M}$, $M_2 \in \mathcal{M}, \ldots$ such that $M_1 \leq M_2 \leq \ldots$ and $M = \lim_{i\to\infty} M_i$.

The next lemma follows from a deeper result called Dickson's lemma. It is, however, easier to prove it directly than to use Dickson's lemma.

**Lemma 2.2.** Every infinite sequence of $\omega$-markings has an infinite growing subsequence.

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

Clearly $(M_i)_{i\in\mathbb{N}}$ qualifies as the $(M_{0,i})_{i\in\mathbb{N}}$. 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 $(M_{j,i})_{i\in\mathbb{N}}$ 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 $(M_{j-1,i})_{i\in\mathbb{N}}$ has an infinite subsequence where $M_{j-1,i}(p_j)$ grows. It qualifies as the $(M_{j,i})_{i\in\mathbb{N}}$.

Finally $(M_{|P|,i})_{i\in\mathbb{N}}$ qualifies as the sequence in the claim of the lemma. □

We are ready to define coverability sets. The idea is that for any given set $\mathcal{M}$ of markings, the $\omega$-markings in its coverability set $\mathcal{M}'$ cover every marking in $\mathcal{M}$ without using bigger $\omega$-markings than necessary. The latter requirement is important, because without it $\mathcal{M}' = \{(\omega, \omega, \ldots, \omega)\}$ would always be a valid coverability set. 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 $\omega$-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 $\omega$-markings. Part 2 of the definition says that this is the only justification for the introduction of $\omega$-symbols. So part 1 formalizes that all markings in $\mathcal{M}$ must be covered, and part 2 rules out unnecessary use of $\omega$-symbols in any individual $\omega$-marking. On the other hand, part 2 does not rule out unnecessary $\omega$-markings. The concept of antichain will do that. We will later show that a coverability set is minimal if and only if it is an antichain.

**Definition 2.4.** Let $\mathcal{M}$ be a set of markings and $\mathcal{M}'$ a set of $\omega$-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 $\omega$-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 $M, M, M, \ldots$. 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 2.3 can also be derived from a more general theory [5].

**Lemma 2.3.** Let $\mathcal{M}$ be a set of markings. For each $i > 0$, let $M_i$ be any limit of $\mathcal{M}$ such that $M_1 \leq M_2 \leq \ldots$. Then also $\lim_{i\to\infty} M_i$ is a limit of $\mathcal{M}$.

**Proof:**
For each $i$, let $(M_{j,i})_{j\in\mathbb{Z}^+}$ be an infinite growing sequence of elements of $\mathcal{M}$ such that $\lim_{j\to\infty} M_{j,i} = M_i$. Let $M'_1 = M_{1,1}$. When $i > 1$, let $M'_i$ be the first element of $(M_{j,i})_{j\in\mathbb{Z}^+}$ 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 2.1. 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 $(M'_i)_{i\in\mathbb{Z}^+}$ 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 $\omega$, which is also the limit of $M_i(p)$. As a consequence, $\lim_{i\to\infty} M'_i = \lim_{i\to\infty} M_i$. □

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 2.1 and Corollary 2.1 could also be derived from a more general theory [6].

**Theorem 2.1.** 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 2.4. It also satisfies part 1, because each $M \in \mathcal{M}$ is the limit of $M, M, M, \ldots$.

We prove next that for every $M \in [\mathcal{M}]$, there is an $M' \in \lceil\mathcal{M}\rceil$ 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 $M_{j+1,1} = \lim_{i\to\infty} M_{j,i}$. Clearly $M_{j+1,1} \geq M_{j,1} \geq M$. By Lemma 2.3, $M_{j+1,1} \in [\mathcal{M}]$. We repeat this reasoning with $j = 1$, $j = 2$, and so on as long as possible. Because for a given $j$, $(M_{j,i})_{i\in\mathbb{Z}^+}$ is strictly growing as $i$ grows, $M_{j+1,1}$ has more $\omega$-symbols than $M_{j,1}$. Therefore, $M_{j,1}$ has at least $j$ $\omega$-symbols. This implies that $j$ cannot grow beyond $|P|$. So a maximal element is eventually encountered.

Thanks to this, $\lceil\mathcal{M}\rceil$ satisfies part 1 of Definition 2.4. It clearly also satisfies the rest of Definition 2.4. 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 2.2, 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 2.4 yields $\mathcal{M}' \subseteq [\mathcal{M}]$.

For any $M \in \lceil\mathcal{M}\rceil$, 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 $\omega$-markings. Thus $\mathcal{M}'$ must contain an $\omega$-marking $M'$ that covers infinitely many of $M_i$. Definition 2.2 implies that $M \leq M'$. We have $M' \in \mathcal{M}' \subseteq [\mathcal{M}]$, and $M \in \lceil\mathcal{M}\rceil$ makes $M < M'$ impossible. Therefore, $M' = M$. As a conclusion, $\lceil\mathcal{M}\rceil \subseteq \mathcal{M}'$.

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

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 2.1.** 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 2.1, 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 $\lceil\mathcal{M}\rceil$. Because it is finite, it cannot have any infinite set as a proper subset. By the proof of Theorem 2.1, every finite coverability set has $\lceil\mathcal{M}\rceil$ as a subset. So $\lceil\mathcal{M}\rceil$ cannot have a proper subset that is a coverability set. $\square$

## 3. Basic Algorithm

In this section we present, discuss, and prove correct the simplest of the algorithms in this publication, and its variant that uses what we call history merging.

### 3.1. Petri Nets

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 $\omega$-markings is the same as with markings: $M[t\rangle M'$ 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 $M[t\rangle$ 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 $M\,[t_1 t_2 \cdots t_n\rangle\, M'$ in the usual way.

```
$F$ := $\{\hat{M}\}$; $A$ := $\{\hat{M}\}$; $W$ := $\{\hat{M}\} \times T$; $\hat{M}.B$ := nil
**while** $W \neq \emptyset$ **do**
    $(M, t)$ := any element of $W$; $W$ := $W \setminus \{(M, t)\}$
    **if** $\neg M[t\rangle$ **then** continue
    $M'$ := the $\omega$-marking such that $M[t\rangle M'$
    **if** $M' \in F$ **then** continue
    Add-$\omega(M, M')$        // if $\exists M''$ : $\exists$ path: $M' > M''$ $-$path$\rightarrow$ $M$ $[t\rangle$ $M'$ then ...
    **if** $\omega$ was added **then if** $M' \in F$ **then** continue
    Cover-check$(M')$      // only keep maximal — may update $A$ and $W$
    **if** $M'$ is covered **then** continue
    $F$ := $F \cup \{M'\}$; $A$ := $A \cup \{M'\}$; $W$ := $W \cup (\{M'\} \times T)$; $M'.B$ := $M$
```

Figure 1.  The basic coverability set algorithm

## 3.2. Overview

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

The set of found $\omega$-markings is divided into sets of *active* and *passive* $\omega$-markings. The set of active $\omega$-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 $\omega$-marking structure. This is an inefficient solution and probably the lowest-quality detail in our test implementation. However, designing a good implementation for $A$ would be a research topic of its own (see, e.g., [1]).

Each $\omega$-marking $M'$ has a *back pointer* $M'.B$ that points to the $\omega$-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 the program can easily and efficiently 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". (We will return to this issue in Section 4.) To model this generality, in Fig. 1, $W$ contains pairs consisting of an $\omega$-marking and a transition. In practice it suffices to store $\omega$-markings instead of pairs. In our test implementation, the workset is a queue, stack, or heap containing pointers to $\omega$-markings, and each $\omega$-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 $\omega$-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.

```
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 $now = last$
```

Figure 2. The basic version of $\omega$-addition

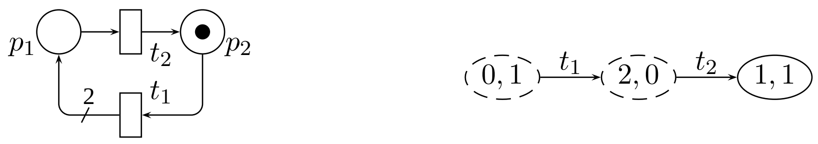


Figure 3. The advantage of repeated scanning of history

If the firing of $t$ from $M$ succeeds, the algorithm checks whether the resulting $\omega$-marking $M'$ has already been found. If found, $M'$ is rejected. Otherwise the operation Add-$\omega$ is applied to $M'$. It adds $\omega$-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-$\omega$, then the algorithm tests again whether the resulting $\omega$-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 $\omega$-marking in $A$. It also removes from $A$ those $\omega$-markings that $M'$ covers strictly. Therefore, $A$ is always the set of maximal found $\omega$-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 $\omega$-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 $\omega$-marking.

## 3.3. Add-$\omega$

Add-$\omega$ is shown in Fig. 2. It scans the history of $M$ towards the initial marking at least once. It tries to find an $\omega$-marking $M'' = now$ that is strictly covered by $M'$, in such a way that $M'$ does not yet have $\omega$ in all those places where $M''(p) < M'(p)$. Whether such a covered $\omega$-marking was found is recorded 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) $\omega$-marking, and so on. The operation terminates when it has fully tested the history without being able to add new $\omega$-symbols.

The purpose of repeated scanning of the history is to add as many $\omega$-symbols as possible to $M'$. For example (see Fig. 3), let $(0, 1)\ [t_1\rangle\ (2, 0)\ [t_2\rangle\ (1, 1)$. For each $n \in \mathbb{N}$ except 0 we have $(0, 1)\ [(t_1 t_2)^{2n-1}\rangle$ $(2n - 1, 1)\ [t_2^{n-1}\rangle\ (n, n)$, where $\sigma^i$ means $\sigma$ repeated $i$ times. So $(\omega, \omega)$ is a limit of reachable markings.

Add-$\omega$ checks whether $(1,1)$ covers $(2,0)$. It does not. Then it checks whether $(1,1)$ covers $(0,1)$. It does, so Add-$\omega$ converts $(1,1)$ to $(\omega,1)$. Then it has found the end (or beginning) of the history. If it terminated there, its result would be $(\omega,1)$. However, Add-$\omega$ starts anew at $(2,0)$ and sees that $(\omega,1)$ covers it. Therefore, it converts $(\omega,1)$ to $(\omega,\omega)$.

We will later see that inserting an $\omega$-marking to the data structures is an expensive operation. By Corollary 2.1 and Theorem 2.1, the algorithm only has to maintain *maximal* $\omega$-markings. Therefore, first inserting $(\omega,1)$ and later removing it and inserting $(\omega,\omega)$ is disadvantageous compared to just inserting $(\omega,\omega)$. Repeated scanning has a non-negligible effect to the running time of Add-$\omega$ only when $\omega$-symbols are added. The only situation where repeated scanning does extra work without giving new $\omega$-symbols in return is the last scan through the history. If there already has been a full scan through the history, then the last scan obviously less than doubles the running time of Add-$\omega$. In the opposite case, the last scan overlaps the scan that would be done without repeated scanning, so again the running time less than doubles. It is a relatively small price.

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

We write $M \mathrel{-t\overset{\omega}{\rightarrow}} M'$ 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 $M$.

We will later need the fact that a path may trigger the addition of $\omega$-symbols even if it reduces the number of tokens in some places. However, this only happens if the marking of those places is $\omega$ before the addition. In the previous example, the path $(2,0)\,[t_2\rangle\,(1,1)$ led to the addition of $\omega$ to $p_2$, although the path reduces the number of tokens in $p_1$. This was because $p_1$ already had $\omega$ as its marking. The proof that $(n,n)$ was reachable for every $n > 0$ used the path $(0,1)\ [(t_1t_2)^{2n-1}\rangle\ (2n-1,1)\ [t_2^{n-1}\rangle\ (n,n)$. That is, it first pumped so many tokens to $p_1$ that even after the repetition of $t_2$, $n$ tokens remained in $p_1$.

In general, that the marking of some places is $\omega$ implies that for any finite path, enough tokens can be loaded to the places before starting the path that they suffice for the path and still leave at least the desired number of tokens to the places. Lemma 3.2 in Section 3.6 will make this precise. This does not hold for infinite paths, however. This is a limitation of coverability graph methods in general.

Informally, once the marking of a place has become $\omega$, the place can be forgotten. The firing rule in effect does so, because the condition $\omega \geq W(p,t)$ always holds and because always $\omega - W(p,t) + W(t,p) = \omega$. Along any path, $\omega$-symbols may be added but not removed.

## 3.4. Cover-check

The purpose of Cover-check$(M')$ is to ensure that $A$ always consists of the maximal $\omega$-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 $t \in T$. 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 $\omega$-marking has already been found can be performed very efficiently with hash tables, testing whether any found $\omega$-marking covers the given $\omega$-marking seems much more

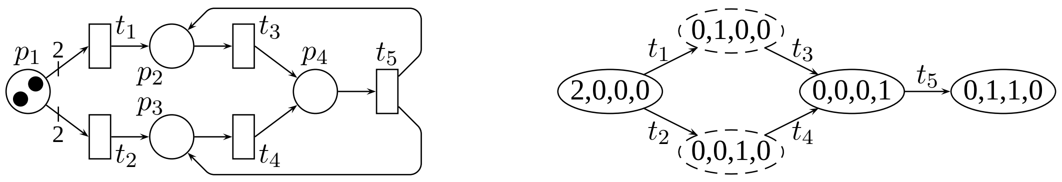


Figure 4. How history merging helps

difficult. The same applies to the removal of those found $\omega$-markings that are strictly covered by the given one. One data structure for these purposes has been used in [1]. All data structures that we are aware of eventually revert to comparing the new $\omega$-marking one by one to old $\omega$-markings, although the total amount of comparison work can be reduced significantly with heuristics.

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 $\omega$-symbols as possible to the $\omega$-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 $\omega$-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 $\omega$-marking with a hash table costs next to nothing, while the cost of the coverability check is significant. Repeated instances of the same $\omega$-marking are common with Petri nets, because if $M\,[t_1t_2\rangle\, M_{12}$ and $M\,[t_2t_1\rangle\, M_{21}$, then $M_{12} = M_{21}$. Therefore, it makes sense to implement a special mechanism for them that is much faster than the general mechanism.

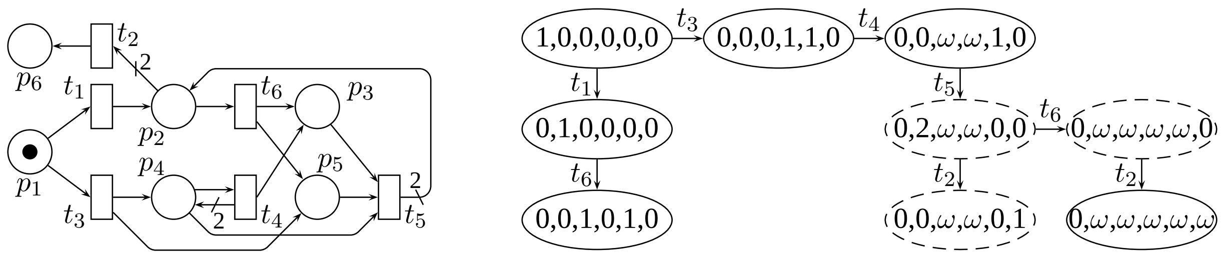
Also Add-$\omega$ is costly compared to the test whether $M' \in F$. Performing the test before Add-$\omega$ and after each addition of $\omega$-symbols inside Add-$\omega$ would speed Add-$\omega$ up. It would also globally slow down the addition of $\omega$-symbols, because a new instance of the same $\omega$-marking usually has a different history and may thus introduce $\omega$-symbols to different places.

If most calls of Add-$\omega$ lead to the addition of $\omega$-symbols, then the coverability set tends to be relatively small, because at most $|P|$ $\omega$-symbols can be added along any path. Therefore, when designing for good performance, it makes sense to concentrate on the opposite case where most calls of Add-$\omega$ do not lead to the addition of $\omega$-symbols. For this reason, we believe that it is advantageous to test $M' \in F$ before calling Add-$\omega$. On the other hand, if Add-$\omega$ has already succeeded in adding an $\omega$-symbol, then the chances of finding a new maximal $\omega$-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.

## 3.5. 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 $\omega$-markings, instead of being (a pointer to) the single predecessor $\omega$-marking via which $M$ was first found. This reflects the fact that the same $\omega$-marking may be reached in multiple ways, any of which may justify the addition of $\omega$-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 the instance of $M'$ that already is in $F$. Thus the



$p_6$
$t_2$
2
$t_1$
$p_2$
$t_6$
$p_3$
$p_4$
$p_5$
$p_1$
$t_3$
2
$t_4$
2
$t_5$
1,0,0,0,0,0
$t_3$
0,0,0,1,1,0
$t_4$
0,0,$\omega$,$\omega$,1,0
$t_1$
0,1,0,0,0,0
$t_6$
0,0,1,0,1,0
$t_5$
0,2,$\omega$,$\omega$,0,0
$t_6$
0,$\omega$,$\omega$,$\omega$,$\omega$,0
$t_2$
0,0,$\omega$,$\omega$,0,1
$t_2$
0,$\omega$,$\omega$,$\omega$,$\omega$,$\omega$

Figure 5. Advanced history merging example

predecessor set collects pointers to all $\omega$-markings from which $M'$ was reached by firing one transition and possibly executing Add-$\omega$.

Consider the example in Fig. 4. Suppose the algorithm proceeds in a breadth-first or most tokens first (Section 4.4) manner. (We want our examples to apply to depth-first or most tokens first search order, because we will later see that they are preferable.) It finds the $\omega$-markings $(0,1,0,0)$ and $(0,0,1,0)$, by firing $t_1$ and $t_2$ from $(2,0,0,0)$. It then generates $(0,0,0,1)$ by firing $t_3$ from $(0,1,0,0)$. The same $\omega$-marking $(0,0,0,1)$ is also found by firing $t_4$ from $(0,0,1,0)$, at which time the history of $(0,0,0,1)$ can be updated to contain both $(0,1,0,0)$ and $(0,0,1,0)$. The benefit comes when $\omega$-symbols are added to $(0,1,1,0)$: as both these $\omega$-markings are in its history, two omegas can be added, giving $(0,\omega,\omega,0)$.

History merging may also be applied on line 10 of Fig. 1, when $M'$ is covered by $M''$ in the following situation: for each $p \in P$ we have $M''(p) = M'(p)$ or $M''(p) = \omega$. Then $M$ can be added to the predecessor set of $M''$. Similarly, the predecessor sets of those $\omega$-markings $M''$ that $M'$ is found to strictly cover on line 9 but that have $M''(p) = M'(p)$ for those places for which $M'(p) < \omega$, can be merged into the predecessor set of $M'$. (Alternatively, the $M''$ themselves could be added to the predecessor set of $M'$. This would be simpler but would yield longer histories.) Let us look at an example.

Figure 5 shows a Petri net and its coverability graph constructed in depth-first order without history merging. In each $\omega$-marking, $t_1$ is tried first, then $t_2$, and so on. For clarity, transition occurrences that take back to their start $\omega$-marking are not shown. With history merging on line 9 switched on, the construction proceeds similarly up to $(0,0,\omega,\omega,1,0)$. Because it strictly covers $(0,0,1,0,1,0)$ and has $\omega$ in those places where it differs from $(0,0,1,0,1,0)$, the history of $(0,0,1,0,1,0)$ is added to the history of $(0,0,\omega,\omega,1,0)$. When $(0,2,\omega,\omega,0,0)$ is found, $(0,1,0,0,0,0)$ is in the history of $(0,0,\omega,\omega,1,0)$, so Add-$\omega$ converts $(0,2,\omega,\omega,0,0)$ to $(0,\omega,\omega,\omega,0,0)$. From it, $t_2$ constructs $(0,\omega,\omega,\omega,0,1)$, which is converted to $(0,\omega,\omega,\omega,0,\omega)$ by Add-$\omega$. Then $t_1$, $t_3$, and $t_5$ are disabled, and $t_2$ and $t_4$ take back to the same $\omega$-marking. Firing $t_6$ leads to $(0,\omega,\omega,\omega,1,\omega)$ which becomes $(0,\omega,\omega,\omega,\omega,\omega)$. Instead of the three dashed $\omega$-markings in Fig. 5, the algorithm stores $(0,\omega,\omega,\omega,0,0)$ and $(0,\omega,\omega,\omega,0,\omega)$. Altogether, eight $\omega$-markings are stored in $F$ instead of nine.

The correctness of this follows from Lemma 3.2, but it may be beneficial to discuss its intuition already now. The purpose of the histories is to make it possible to change the markings of places to $\omega$ by Add-$\omega$. Once the marking of a place has become $\omega$, its historical values do not affect its own future. As was discussed towards the end of Section 3.3, $\omega$-marked places can be loaded beforehand with simultaneously as many tokens as needed for any finite purpose. Therefore, $\omega$-marked places can be ignored also when considering the futures of other places. So "distorting" the history of $\omega$-marked places is not a problem from the point of view of any place. History merging does not distort the history of any place whose marking is not (yet) $\omega$, because histories are not merged unless the place has the same marking in both histories. Therefore, along any path in the histories, as long as the marking of a place is finite, it evolves correctly.

To be more concrete, consider any $M''$, transitions $t_1, \ldots, t_n$, and path such that $M[t\rangle M'$, $M' > M''$, the transition sequence along the path is $t_1 \cdots t_n$, and the path leads from $M''$ to $M$. The places $p$ such that $M'(p) = \omega$ can be loaded beforehand with many enough tokens for the rest of this discussion, by repeating the transition sequences that correspond to the paths that caused the changes of their markings from finite values to $\omega$. After that, $t_1 \cdots t_n t$ can be repeated a sufficient number of times, to pump desired numbers of tokens to those places with $M''(p) < M'(p) < \omega$. The remaining places have $M''(p) = M'(p) < \omega$, and their markings stay the same. As a result, a marking is obtained that has at least any desired number of tokens in those places with $M'(p) = \omega$ or $M''(p) < M'(p) < \omega$, and matches $M'$ for the remaining places.

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

Repeated searches require repeated resetting of "found" information, which may become a performance problem if implemented naïvely. In our test implementation, each $\omega$-marking has an attribute `search_nr` and there is a global variable `search_now`. When an $\omega$-marking is found, `search_now` is assigned to its `search_nr`. In the beginning of each search, `search_now` is incremented. If it overflows its type, it is set to 1 and the `search_nr` of every found $\omega$-marking is set to 0 by scanning the hash table that implements $F$. In a 32-bit machine, overflow occurs after each 4 294 967 295 searches. Because the base table of the hash table is much smaller than this, the cost of scanning the hash table is negligible compared to the time already spent.

The searching through the history is based on a work list, maintained by yet another pointer in the data structure for $\omega$-markings. A new search is started by updating `search_now` as was discussed above, and then making the list consist of the $\omega$-marking where the search is started. The first element of the list is tested, to see if it triggers the addition of $\omega$-symbols. If it does, $\omega$-symbols are added as appropriate and a new search is started at the current $\omega$-marking. No memory management or clean-up of the old search is needed, because the search list is maintained by pointers in the $\omega$-marking records and because of the way in which "found" information is reset. The old search is just forgotten. The execution continues where it was, independently of whether a new search was started. This implies that the new search does not test its start $\omega$-marking. This is only good, because it was just tested.

The search proceeds by removing the first $\omega$-marking from the search list and going through its predecessor set. It marks as found those predecessors that it has not yet found and adds them to the front of the search list. When the search list becomes empty, the search proceeds to the original starting $\omega$-marking. If it has been found in the current search, the search terminates. Altogether the implementation of the searching through the histories is simple and efficient.

Our test implementation represents the predecessor set as a linked list whose each record points to one predecessor and to the next record of the list. This list needs explicit records of its own, to make it possible for the same $\omega$-marking to be simultaneously in more than one predecessor set. New elements are added to the front without checking whether they already are in the list. So the same pointer to an

$\omega$-marking may be more than once in the list. (Thus really a multiset is represented.) When the searching algorithm finds the same predecessor anew, it recognizes it as having already been found and rejects it. So the additional cost caused by each duplicate is small. Unfortunately, even a small additional cost can have a big effect, if repeated often enough. We do not know whether the number of duplicates can grow so big that it would be a problem. On the other hand, the removal of duplicates would also consume time and would require more complicated code.

We say that the *finding history* of $M$ is the path $M_0 - t_1 \overset{\omega}{\rightarrow} M_1 - t_2 \overset{\omega}{\rightarrow} \ldots - t_n \overset{\omega}{\rightarrow} M_n$ such that $M = M_n$, $M_0 = \hat{M}$, and each $M_i$ other than $M_0$ was first found by firing $t_i$ in $M_{i-1}$ and executing Add-$\omega$. If history merging is switched off, then the finding history is the only history.

## 3.6. Correctness

The correctness of the algorithm and its variants presented in this publication consists of four issues, three of which correspond to the three conditions in Definition 2.4 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 $M[t\rangle M'$ and $M \leq M_1$, then there is an $M_1'$ such that $M_1[t\rangle M_1'$ and $M' \leq M_1'$.

**Lemma 3.1.** After termination, for every reachable marking $M$ of the Petri net, $A$ contains an $\omega$-marking $M'$ such that $M \leq M'$.

**Proof:**
Each time when an $\omega$-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 $\neg M[t\rangle$, or there is an $M'$ such that $M[t\rangle M'$. In the latter case, the set $F$ either contains an $\omega$-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 $(M, t) \in W$, $\neg M[t\rangle$, or for the $M'$ such that $M[t\rangle M'$ 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 $M_0\,[t_1\rangle\, M_1\,[t_2\rangle \cdots [t_n\rangle\, M_n$ 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 $\neg M_{i-1}'[t_i\rangle$, because $M_{i-1}[t_i\rangle M_i$ and $M_{i-1} \leq M_{i-1}'$. Let $M_i'''$ be such that $M_{i-1}'[t_i\rangle M_i'''$. By I2 there is an $M_i'' \in F$ and by I1 an $M_i' \in A$ such that $M_i \leq M_i''' \leq M_i'' \leq M_i'$. $\Box$

**Lemma 3.2.** 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. For each $M$, let $\Omega(M) = \{p \in P \mid M(p) = \omega\}$.

We show first that if $M[t\rangle M'$ and $M$ is a limit of $R$, then also $M'$ is a limit of $R$. We have $\Omega(M) = \Omega(M')$. Let $d$ be the minimum of $W(t,p) - W(p,t)$ over $p \in \Omega(M)$. By Lemma 2.1, 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 $p \in \Omega(M)$ and $M_i(p) = M(p)$ for every $p \in P \setminus \Omega(M)$. We have $M_i[t\rangle$. Let $M_i'$ be the $\omega$-marking such that $M_i[t\rangle M_i'$. We have $M_i'(p) = M_i(p) - W(p,t) + W(t,p) \geq M_i(p) + d \geq n$ for every $p \in \Omega(M)$ and $M_i'(p) = M(p) - W(p,t) + W(t,p) = M'(p)$ for every $p \in P \setminus \Omega(M)$. 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-$\omega$ to it, then $M''$ is a limit of $R$. Consider any $\omega$-marking $M = now$ that triggers addition of $\omega$-symbols to $M'$. Let $t_1, \ldots,$ $t_k$ 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 three possibilities (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 2.1, 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 $M_i'$ 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 $M'' = \lim_{i \to \infty} M_i'$. Furthermore, $M_i\left[(t_1 \cdots t_k)^n\right\rangle M_i'$, 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 $\omega$-markings yields a limit of $R$, if its input $\omega$-markings are limits of $R$. Originally there is only the initial marking $\hat{M}$. It is obviously reachable and the limit of $\hat{M}, \hat{M}, \hat{M}, \ldots$. So all $\omega$-markings found by the algorithm are limits of $R$. $\square$

**Lemma 3.3.** 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 3.4.** The algorithm terminates.

**Proof:**
Termination of loops other than the main loops of Fig. 1 and Add-$\omega$ are obvious. Add-$\omega$ stops adding $\omega$-symbols to $M'$ at the latest when $M'(p) = \omega$ for every $p \in P$, so each call of Add-$\omega$ terminates.

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

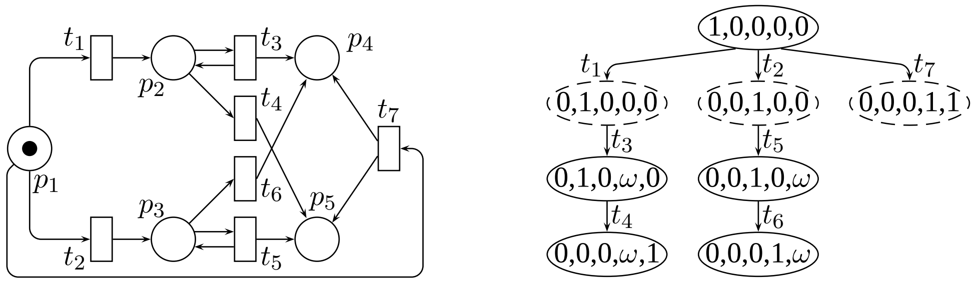


Figure 6. The target of an edge of a minimal coverability graph is not necessarily unique

We conclude that failure of termination requires the existence of an infinite sequence $\hat{M} = M_0 -t_1\overset{\omega}{\rightarrow} M_1 -t_2\overset{\omega}{\rightarrow} M_2 -t_3\overset{\omega}{\rightarrow} \cdots$ such that each $M_i$ is first found by firing $t_i$ from $M_{i-1}$, and then possibly adding $\omega$-symbols with Add-$\omega$. The $M_i$ are distinct because of the tests $M' \in F$. By Lemma 2.2, $M_0, M_1, M_2, \ldots$ has an infinite strictly growing subsequence $M'_0 < M'_1 < M'_2 < \cdots$. Thanks to Add-$\omega$, each $M'_{i+1}$ has at least one $\omega$-symbol more than $M'_i$. However, there are only $|P|$ places, so we run out of places where to add new $\omega$-symbols at $M'_{|P|+1}$, if not earlier. This is a contradiction. So failure of termination is impossible. $\Box$

Together Lemmas 3.1 to 3.4 yield the following theorem.

**Theorem 3.1.** The algorithm in Fig. 1 terminates, and then $A$ contains the minimal coverability set of the Petri net.

## 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 elapsed since the pair was inserted to $W$. (Time is, for example, the number of iterations of the main loop of Fig. 1.) The age of $(M, t)$ is determined by $M$, because the pairs $(M, t)$ for every $t \in T$ are inserted simultaneously to $W$.

### 4.1. A Remark on Minimal Coverability Graphs

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 an order that enables running the verification algorithm on-the-fly. With coverability sets, however, due to the high cost of unnecessary construction of non-maximal $\omega$-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.

As a matter of fact, the notion of *minimal* coverability *graph* is not entirely unproblematic. Clearly the minimal coverability set $\lceil\mathcal{M}\rceil$ should be its set of vertices. Furthermore, if $M \in \lceil\mathcal{M}\rceil$ and $M[t\rangle M'$, then an edge $M -t\rightarrow M''$ should be included in the graph, where $M'' \in \lceil\mathcal{M}\rceil$ and $M' \leq M''$, and there should be no other edges than those created by this rule. Unfortunately, in such a case $M''$ is not

```
$F$ := $\{\hat{M}\}$; $A$ := $\{\hat{M}\}$; $Q$ := $\{\hat{M}\}$; $\hat{M}.B$ := nil
while $Q \neq \emptyset$ do
    $M$ := 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
        $F$ := $F \cup \{M'\}$; $A$ := $A \cup \{M'\}$; add $M'$ to the end of $Q$; $M'.B$ := $M$
```

Figure 7. Breadth-first discipline

necessarily unique. In Fig. 6, the minimal coverability set consists of the $\omega$-markings in solid nodes. The target of $(1,0,0,0,0)\,[t_7\rangle$ could be $(0,0,0,\omega,1)$ or $(0,0,0,1,\omega)$. (Again, self-loops of the coverability graph are omitted for clarity.)

## 4.2. Breadth-first

Breadth-first discipline is obtained by always picking one of the oldest pairs from $W$. One possible implementation is described in Fig. 7, 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 $Q$ could actually be in the same linked list. Such a 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 $\omega$-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 $M$.`next_A` denote the pointer of $M$ that points to the next $\omega$-marking in the list. The test $M \notin A$ can be done in constant time: $M$.`next_A` is made to point to $M$ after $M$ is removed from $A$. It would be correct to skip the test, but then the algorithm would unnecessarily fire transitions from $\omega$-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, intuitively it seems likely that breadth-first typically adds $\omega$-symbols to $\omega$-markings later, that is, after exploring more $\omega$-markings, and thus would have longer running time than other common disciplines. In our measurements (see Section 6.1), breadth-first was never clearly the fastest but was often clearly the slowest. Therefore, we do not recommend breadth-first. Like many of the arguments in this publication, this is a heuristic and not a theorem.

In fact, 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 $(\omega,\ldots,\omega)$

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.

A heuristic that aims at improving all search disciplines in this kind of situations will be discussed in Section 4.5.

## 4.3. Depth-first

Depth-first discipline is obtained by always picking a youngest pair from $W$. It can be implemented by storing the $\omega$-markings of the pairs in a stack; returning $(M, M.\texttt{next\_tr})$ 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 $\omega$-symbols to $\omega$-markings early on, and we provide a theoretical reason for this in Lemma 4.1. This gives reason to believe that depth-first might have a good running time in general. In our measurements it was sometimes 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 investigates $M{-}t\overset{\omega}{\rightarrow}M'$ and either puts $M'$ into $F$ or detects that it is there already. A *descendant* of an $\omega$-marking is the $\omega$-marking itself, its successor, or a descendant of its successor. We say an $\omega$-marking is *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.

**Lemma 4.1.** If the construction order is depth-first and $M$ has more $\omega$-symbols than the $\omega$-marking via which it was first found, then the algorithm will not backtrack from $M$ before it has investigated all descendants of $M$.

**Proof:**
Depth-first search has the property that the set of grey $\omega$-markings and the transitions via which they were first found, constitute a path from the initial marking to the current $\omega$-marking. We call any contiguous sub-path of this path a *grey path*.

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

If $M_2$ is a descendant of $M_1$, then $M_2$ has $\omega$-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 $\omega$-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. $\square$

Another consequence of this lemma is that with depth-first, there is no point in implementing history merging on line 10 of Fig. 1. To see this, let $M'$ and $M''$ be such that $M'$ has just been found, $M'' \in A$, and for every $p \in P$ either $M''(p) = M'(p)$ or $M''(p) = \omega$. If $M''(p) = M'(p)$ for every $p \in P$, then history merging takes place on line 6 or 8. Otherwise there is a $p$ such that $M''(p) = \omega > M'(p)$. Let

$M_1$ be the first in the finding history of $M''$ such that $M_1(p) = \omega$. Because $M_1(p) = \omega > M'(p)$, $M'$ is not a descendant of $M_1$. Therefore, when $M'$ is found, the algorithm has backtracked from $M_1$. By the lemma, then the algorithm has investigated all descendants of $M_1$, and thus of $M''$. So the algorithm will no more look at the predecessor set of $M''$, and adding something to it would have no effect.

## 4.4. Most Tokens First

The desire to add $\omega$-symbols as early as possible naturally leads to the heuristic of always trying next the $\omega$-marking that has the most tokens. The $\omega$-marking with the maximal number of $\omega$-symbols is preferred, and if it is not unique, then the total number of tokens in the places whose markings are not $\omega$ is used as the secondary criterion. Like before, only $\omega$-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 $w$ $\omega$-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 $\omega$-markings. It lost to depth-first in some cases, and often there was no clear difference. It may be remarkable that it lost in the example with the biggest coverability set. However, our set of measurements is far too small for firm conclusions. This is noteworthy because even in this small set of measurements, the variability in performance due to minor details of implementation was sometimes very dramatic.

Also with most tokens first, there is no point in implementing history merging on line 10 of Fig. 1. The reason is similar to with depth-first, except that the argument follows directly from the most tokens first discipline instead of Lemma 4.1.

## 4.5. The Ordering of Transitions

The orderings discussed above affect which $\omega$-marking is taken from the workset. For each $\omega$-marking, transitions could be studied in any order. In this section we discuss the effect of the ordering of transitions.

Towards the end of Section 4.2 we presented a generic example that artificially favors breadth-first search. Also, it turned out in our earlier measurements [11] that the number of constructed $\omega$-markings sometimes depends dramatically on whether the transitions are investigated in the order that they were presented in the input file or in the opposite order (see Section 6.1). These observations led us to ask the question: what happens when we order the transitions according to $\Delta(t) = \sum_{p \in P} W(t,p) - W(p,t)$ before the construction of the minimal coverability set? In each $\omega$-marking, the transition with the biggest $\Delta(t)$ is tried first. The idea is to try first the transitions that make the total number of tokens grow most. This heuristic is imprecise when dealing with $\omega$-markings that already have at least one $\omega$-symbol, because $\Delta(t)$ also considers $\omega$-marked places although their numbers of tokens will no longer change. This could be avoided by re-ordering transitions after each addition of $\omega$-symbols, but this would have been much more complicated.

In the example in Section 4.2, $\Delta(t) = |P|$ and $\Delta(t') = |\hat{M}| - 1$, where $|\hat{M}|$ denotes the total number of tokens in the initial marking. Typically (but not by necessity) many places are initially empty and very few have more than one token, yielding $|\hat{M}| - 1 < |P|$. This forces $t$ to be tried before $t'$ and leads to fast termination with all search disciplines.

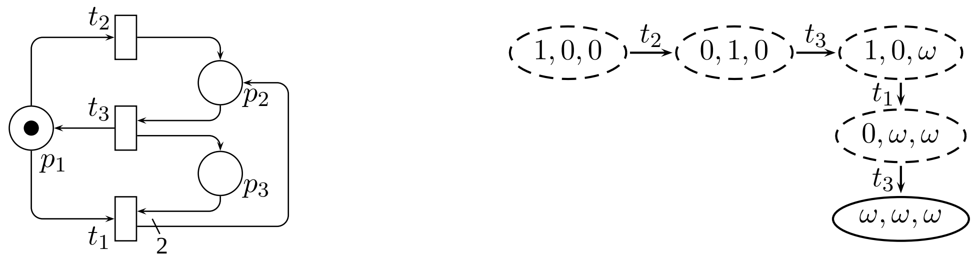


Figure 8. A pumping $\omega$ example

We expected this order of transitions to be typically better and the reverse of this order typically worse than the order of the transitions in the input file and its reverse. However, the measurements in Section 6.1 did not meet this expectation. The imprecision mentioned above may be one reason. It is also possible that in an $\omega$-marking, to enable a transition that leads to addition of $\omega$-symbols, another transition whose $\Delta(t)$ is small has to be fired first.

## 5. To Prune or Not to Prune

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

### 5.1. Pumping Cycle Passivation

Consider $M_0$ in the history of $M_n$ such that $M_0$ triggered the addition of at least one $\omega$-symbol to $M_n$ along the path $M_0 \; {-t_1}\!\overset{\omega}{\rightarrow} \; M_1 \; {-t_2}\!\overset{\omega}{\rightarrow} \; \dots \; {-t_n}\!\overset{\omega}{\rightarrow} \; M_n$. Then $M_0 < M_n$ and $M_n \, [t_1 \cdots t_n\rangle$. When $0 \leq i \leq n$, let $M'_i$ be the $\omega$-marking such that $M_n \, [t_1 \cdots t_i\rangle \, M'_i$. 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 $M_0, \dots, M_{n-1}$, when $\omega$-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 [3] 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.

Secondly, the removal may slow down the addition of $\omega$-symbols. Consider the example in Fig. 8, assuming depth-first discipline. The algorithm in Fig. 1 fires $(1,0,0) \; [t_2\rangle \; (0,1,0) \; [t_3\rangle \; (1,0,1)$ and converts $(1,0,1)$ to $(1,0,\omega)$. Then it fires $(1,0,\omega) \; [t_1\rangle \; (0,2,\omega)$ and converts $(0,2,\omega)$ to $(0,\omega,\omega)$, because it covers $(0,1,0)$. Finally $(0,\omega,\omega) \; [t_3\rangle \; (1,\omega,\omega) > (0,\omega,\omega)$, yielding $(\omega,\omega,\omega)$. 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 $(1,0,\omega) \; [t_1\rangle \; (0,2,\omega) \; [t_3\rangle \; (1,1,\omega)$ are fired, yielding $(1,\omega,\omega)$. Again, all other $\omega$-markings are removed. Firing $t_1$ and $t_2$ from $(1,\omega,\omega)$ does not yield new maximal $\omega$-markings, but $t_3$ yields $(\omega,\omega,\omega)$. So seven transition firings were needed.

## 5.2. Future Pruning

Pruning of futures refers to the passivation or removal of some or all found $\omega$-markings whose histories contain an $\omega$-marking that was strictly covered by a newly found $\omega$-marking. Pumping cycle passivation can be considered as a special case of future pruning. The algorithms in [3] and [10] 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 [4] reveals a flaw in the future pruning of the algorithm in [3]. In the counter-example, an $\omega$-marking $M_1$ first triggers pumping cycle removal. Then another $\omega$-marking $M_2$ with a different history is found, and its successor $\omega$-markings are covered by $M_1$. Therefore, $M_2$ remains active but does not lead to any new $\omega$-markings. An $\omega$-marking in the (removed) pumping cycle is covered by $M_2$, but the algorithm fails to notice this, since the cycle's $\omega$-markings have been removed. Finally, a third $\omega$-marking $M_3$ is found that covers strictly some $\omega$-marking in the history of $M_1$, and $M_1$ is removed. The firing of transitions from $M_3$ leads to an $\omega$-marking that is covered by $M_2$, and exploration stops short of finding an $\omega$-marking that covers $M_1$.

In [10], the algorithm never removes, only passivates $\omega$-markings. The presence of these passive $\omega$-markings in the histories of active $\omega$-markings means that pruning happens differently from [3]. When a pumping cycle is found, the intermediate $\omega$-markings are passivated, but they remain in the history of the new $\omega$-marking. Whenever a new $\omega$-marking $M$ strictly covers some $\omega$-marking not in its own history, the whole branch starting from that $\omega$-marking is passivated, even if the covered $\omega$-marking is passive. This avoids the behaviour described above. When $M_1$ in the counter-example triggers pumping cycle passivation, the intermediate $\omega$-markings remain in a tree. On the way to $M_2$ the algorithm encounters another $\omega$-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 $\omega$-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. There are also arguments of more general nature that suggest that future pruning is not necessarily worth the effort. They are the topic of the next subsection.

## 5.3. Is It Worth the Effort?

Our first observation is that there is no conclusive "yes" or "no" answer that would be valid in all situations. This is because the running time may depend heavily on finding a certain $\omega$-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. (These Petri nets are so densely connected that represented as pictures, they would be hard to read.) 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 $\omega$-marking, transitions are tried in the numeric order. Like in [10] and unlike in Section 3, we assume that only active $\omega$-markings are taken into account in Add-$\omega$.

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

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

With the first Petri net, both algorithms fire first $(1,0,0) -t_2\overset{\omega}{\rightarrow} (0,2,0) -t_4\overset{\omega}{\rightarrow} (0,2,\omega) -t_3\overset{\omega}{\rightarrow}$ $(1,0,\omega)$ 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 $(1,0,\omega) -t_1\overset{\omega}{\rightarrow} (\omega,\omega,\omega)$, passivates all other $\omega$-markings, fires $(\omega,\omega,\omega) -t_i\overset{\omega}{\rightarrow} (\omega,\omega,\omega)$ 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 $(0,2,\omega) -t_4\overset{\omega}{\rightarrow} (0,2,\omega)$ and $(0,2,\omega) -t_5\overset{\omega}{\rightarrow}$ $(\omega,2,\omega) -t_1\overset{\omega}{\rightarrow} (\omega,\omega,\omega)$, fires each $t_i$, and terminates. So the pruning algorithm is faster. (Thanks to $(1,0,0)$, the Add-$\omega$ in Section 3 would have yielded $(0,2,\omega) -t_5\overset{\omega}{\rightarrow} (\omega,\omega,\omega)$.)

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

Our second observation is that the pruning algorithm may activate the same $\omega$-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 $(0,2,\omega,0,0) -t_1t_6t_6t_6\overset{\omega}{\rightarrow} (0,0,\omega,0,3)$. Then it fires $(0,2,\omega,0,0) -t_3\overset{\omega}{\rightarrow} (1,0,\omega,0,0)$, notices that $(1,0,0,0,0)$ is covered, and passivates all $\omega$-markings other than $(1,0,\omega,0,0)$. Next it fires $t_2$, activating $(0,2,\omega,0,0)$ again. Then it fires $(0,2,\omega,0,0) -t_1t_6t_6t_6\overset{\omega}{\rightarrow} (0,0,\omega,0,3)$ for a second time.

The goal of pruning is to avoid unnecessarily investigating $\omega$-markings that will later be strictly covered by other $\omega$-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 5.1.** Let the construction order be depth-first and history merging be applied on lines 6 and 8 of Fig. 1. Assume that $M_0 -t_1 \cdots t_n\overset{\omega}{\rightarrow} M_n$ and $M_0 < M'_0$. Assume that all transitions along the path $M_0 -t_1 \cdots t_n\overset{\omega}{\rightarrow} M_n$ were found before $M'_0$. After finding $M'_0$, the algorithm will not fire transitions from $M_n$, unless $M'_0 \; [t_1 \cdots t_n\rangle \; M_n$.

**Proof:**
Let $\Omega(M)$ be the set of places such that $M(p) = \omega$. Let $M_1, \ldots, M_{n-1}$ 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 $\omega$-marking, that is, $M'_0$. We denote its transitions and $\omega$-markings with $t_{n+1}, \ldots, t_m$ and $M_{n+1}, \ldots, M_m$, where $M_m = M'_0$. We have $M_0 -t_1 \cdots t_n\overset{\omega}{\rightarrow} M_n$ $-t_{n+1} \cdots t_m\overset{\omega}{\rightarrow} M'_0$, and $M_0 < M'_0$. Thanks to Add-$\omega$, for each $p \in P \setminus \Omega(M'_0)$, we have $M'_0(p) = M_0(p)$. In particular, $M'_0$ has more $\omega$-symbols than $M_0$.

Along any path, the marking of any place may change from finite to $\omega$ but not vice versa. Let $M''$ be the first $\omega$-marking found by the algorithm from which $M'_0$ is reachable and who has $\Omega(M'') = \Omega(M'_0)$. By Lemma 4.1, the algorithm will not backtrack from $M''$ before it has visited $M'_0$. Because $M'_0$ has just been found, $M''$ is grey. Let $\mathcal{M}$ be $\{M''\}$ union the set of $\omega$-markings that have been found after $M''$. There has been only one firing of a transition that leads from outside $\mathcal{M}$ to $\mathcal{M}$, and that is the transition via which $M''$ was found. This is because the algorithm has not backtracked from $\mathcal{M}$ after finding $M''$. Because the path $M_0 -t_1 \cdots t_m\overset{\omega}{\rightarrow} M'_0$ has such a transition, it must be the same transition, and therefore $M'' = M_h$ for some $0 < h \leq m$.

Let $M''_n$ be the $\omega$-marking such that $M'_0\ [t_1 \cdots t_n\rangle\ M''_n$. 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 $\omega$-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 $\Omega(M_n) = \Omega(M'_0)$. Also $\Omega(M''_n) = \Omega(M'_0)$, because $M''_n$ was defined using "$[\cdots\rangle$" instead of "$- \cdots \overset{\omega}{\rightarrow}$". For $p \in P \setminus \Omega(M'_0)$ we have $M_0(p) = M'_0(p)$, so also $M_n(p) = M''_n(p)$. We conclude that $M''_n = M_n$, implying $M'_0\ [t_1 \cdots t_n\rangle\ M_n$. $\square$

We prove a similar theorem for most tokens first search.

**Theorem 5.2.** Let the construction order be most tokens first and history merging be applied on lines 6 and 8 of Fig. 1. Assume that $M_0\ -t_1 \cdots t_n \overset{\omega}{\rightarrow}\ M_n$ and $M_0 < M'_0$. Assume that all transitions along the path $M_0\ -t_1 \cdots t_n \overset{\omega}{\rightarrow}\ M_n$ were found before $M'_0$. After finding $M'_0$, the algorithm will not fire transitions from $M_n$, unless $M'_0\ [t_1 \cdots t_n\rangle\ M_n$.

**Proof:**
By investigating an $\omega$-marking we mean trying to fire one or more transitions in it. With most tokens first search, the investigation of an $\omega$-marking may be interrupted by the investigation of other $\omega$-markings and continued later.

Let $\Omega(M)$ be the set of places such that $M(p) = \omega$. Let $M_1, \ldots, M_{n-1}$ be defined in the obvious way. Let $M \prec M'$ denote that $M$ has fewer $\omega$-marked places than $M'$, or the same number of $\omega$-marked places but altogether fewer tokens in the remaining places than $M'$. By the most tokens first discipline, if $M \prec M'$ and $M'$ has been found, then transitions are not investigated at $M$ before all transitions have been investigated at $M'$ or $M'$ has been removed from $A$. Furthermore, $M < M'$ implies $M \prec M'$. If $M \not\prec M'$, then $M' \preceq M$. Also note that along any path, $\omega$-symbols may be introduced but cannot disappear. Let $A(M)$ denote any $M''$ that currently represents $M$, that is, $M \leq M''$ and $M'' \in A$ (see I1 in the proof of Lemma 3.1).

If $M_n \preceq M_i$ for each $0 \leq i \leq n$, then $\Omega(M_n) = \Omega(M_0)$. Furthermore, $M_n \preceq M_0 \prec M'_0 \preceq A(M'_0)$, so $A(M'_0) {-t_1\overset{\omega}{\rightarrow}} M'_1$ is fired before firing transitions from $M_n$. Because $M_0 < M'_0$ and $\omega$-symbols were not added during $M_0\ -t_1 \overset{\omega}{\rightarrow}\ M_1$, we have $M_n \preceq M_1 < M'_1$. The reasoning continues until we get $M_n < M'_n$. 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 $M_i, \ldots, M_n$ has been found before $M_{i-1} {-t_i \overset{\omega}{\rightarrow}} M_i$ is fired, then all transitions from such an $\omega$-marking and eventually all transitions from $M_n$ are fired before $M_{i-1} {-t_i \overset{\omega}{\rightarrow}} M_i$. 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 visited after firing $M_{i-1} {-t_i \overset{\omega}{\rightarrow}} M_i$ but before $M'_0$ is visited 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 $M_{i-1} {-t_i \overset{\omega}{\rightarrow}} M_i$, and (1) from then on every $\omega$-marking $M$ visited has $M_n \preceq M$ until $M'_0$ is found.

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

Table 1. Some measurements

| model | $\lvert A\rvert$ | most tokens first | | depth-first | | breadth-first | | [10] |
|---|---|---|---|---|---|---|---|---|
| fms | 24 | $^{q}53$ | $^{i}63$ | $^{q}52$ | $^{i}110$ | $^{-i*}139$ | $^{i*}509$ | 809 |
| kanban | 1 | 12 | 12 | 12 | 12 | 12 | 12 | 114 |
| mesh2x2 | 256 | $^{-q}451$ | $^{q}479$ | $^{-q}355$ | $^{i}774$ | $^{-i*}2977$ | $^{q}20307$ | 6241 |
| mesh3x2 | 6400 | $^{-q}11480$ | $^{i}11495$ | $^{-q}6879$ | $^{-i}10394$ | T/O | T/O | |
| multipoll | 220 | $^{q}231$ | $^{i}245$ | $^{-q}233$ | $^{q}245$ | $^{q}334$ | $^{i}507$ | 2004 |
| pncsacover | 80 | $^{i}215$ | $^{-i}246$ | $^{q}277$ | $^{-i*}327$ | $^{q*}5205$ | $^{-q}9490$ | 1604 |
| AP13a | 1245 | $^{i}1295$ | $^{q}1296$ | $^{q}1263$ | $^{-q}1318$ | $^{i}1296$ | $^{q}1323$ | |
| OS12a | 2735 | $^{q}4701$ | $^{-q}17853$ | $^{q}5460$ | $^{i}$T/O | T/O | T/O | |
| OS11a | 2 | $^{q}448$ | $^{-i}$T/O | $^{q}678$ | $^{-i}$T/O | T/O | T/O | |

By $M_n \preceq M_i$, we have $\Omega(M_n) = \Omega(M_i)$. Therefore, $\Omega(M_n) \subseteq \Omega(M'_0)$.

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

Otherwise $\Omega(M'_0) = \Omega(M_n)$. This implies $M'_0\ [t_1 \cdots t_n\rangle\ M_n$, because $M_0(p) = M'_0(p)$ if $M'_0(p) < \omega$. □

## 6. Experiments and Conclusion

In this section we first discuss some experiments we carried out, and then provide a few concluding remarks.

### 6.1. Experiments

We implemented the algorithm in C++, using the three search orders and the transition ordering discussed in Section 4. Our implementation also supports history merging as discussed in Section 3.5, including history merging on lines 9 and 10 of Fig. 1.

Table 1 shows results on some test runs we made with the program. The first six cases are from the test set used in [7], and the rest are miscellaneous semi-random models.

The second column shows the size of the minimal coverability set of the model. The other numbers are the total numbers of constructed distinct $\omega$-markings, that is, $|F|$. Each search order was implemented with and without history merging. We tried firing transitions in four different orders: first, the order in which they were given in the input file (indicated by $^{i}$), and second, sorted according to the $\Delta(t)$-heuristics described in Section 4.5 (indicated by $^{q}$). The reversed versions ($^{-}$) of both these orderings were also tried. The table reports the minimum and the maximum $|F|$ for each construction order, and reports with which transition order the result was gained. We omit mentioning transition order when all possibilities produced the same result.

History merging can never increase the size of $|F|$, but it can sometimes reduce it. This seems to happen especially in conjunction with breadth-first search, but appears to be uncommon otherwise. There

Table 2. Measurements of the effect of $F$-check on execution time

| model | most tokens f. | | | depth-first | | |
|---|---|---|---|---|---|---|
| multipoll | $^{q}$233 | 5ms | 13ms | $^{q}$230 | 7ms | 20ms |
| mesh2x2 | $^{-q}$451 | 12ms | 40ms | $^{-q}$355 | 18ms | 68ms |
| AP13a | $^{i}$1295 | 296ms | 311ms | $^{q}$1263 | 242ms | 246ms |
| OS12a | $^{q}$4701 | 1872ms | 2888ms | $^{q}$5460 | 2084ms | 3924ms |
| mesh3x2 | $^{-q}$11480 | 2763ms | 11668ms | $^{-q}$6879 | 2744ms | 28555ms |

were some differences between depth-first and most tokens-first in the model pnsacover, but even in that model, the minimal solutions for these two construction orders were the same with and without history merging. The numbers marked with an asterisk (*) indicate cases where history merging resulted in a smaller $|F|$.

A one minute timeout (indicated by T/O) was set for the experiments. The running time was quite fast, in many cases in the order of a few milliseconds, in most cases less than one millisecond. When the one minute timeout was encountered, the program had usually explored between 50 000 and 100 000 $\omega$-markings, depending on the model. We used the one minute cut off because some of the large models, such as mesh3x2, failed to terminate with breadth-first search even when left running for several hours. On the other hand, most of the experiments with other search orders were able to finish within one minute. These times should not be compared to those in [7, 10], because we used a different hardware and programming language.

As the table shows, the low-level, more or less arbitrary difference in how transitions are ordered, sometimes had a dramatic impact. The OS11a model above is an extreme example. When transitions were sorted according to the $\Delta(t)$-heuristic, depth-first and most tokens-first searches resulted in under a thousand $\omega$-markings. The initial ordering with depth-first resulted in 36444 $\omega$-markings, and the reversed orderings failed to terminate within one minute. These results act as a dire warning that numbers like the ones in the table are much less reliable than they may appear.

In Section 3.4, the argument was made that checking first whether an $\omega$-marking is in $F$ before proceeding with Add-$\omega$ or Cover-check would speed up the program. We tested the gravity of this argument with our program. Because breadth-first construction order did not look promising and because history merging made very little difference in depth-first and most tokens- first construction orders, we measured the effect only with depth-first and most tokens-first orders, without history merging. We selected the transition orders where the program had the smallest $|F|$, and measured the time. We took multiple measurements and recorded fastest time to eliminate the effect of load and other factors that cause random fluctuations in execution time. In no case did skipping the $F$-check improve performance, as is to be expected. It seems also that model-specific factors determine how much the check improves performance. The measurements of shorter times should be considered somewhat unreliable. From what we can tell, the $F$-check can provide significant improvement, especially with the large cases.

## 6.2. Conclusions

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 $\omega$-markings, like pruning algorithms do. We also pointed out that it may be advantageous to add as many $\omega$-symbols as early as possible, and presented techniques towards such a goal. We argued that detecting already encountered $\omega$-markings using a fast data structure, such as a hash table, to prevent unnecessary coverability checks speeds up the algorithm. Our experiments support this argument.

## Acknowledgements

We would like to thank Prof. Mikko Tiusanen and the reviewers for their help, and the authors of [10] for providing help for the experiments.

## References

[1] G. Delzanno, J.-F. Raskin, and K. Van Begin. Covering Sharing Trees: A Compact Data Structure for Parameterized Verification. *Software Tools for Technology Transfer* 5(2-3), 268–297 (2004)

[2] A. Finkel. A Generalization of the Procedure of Karp and Miller to Well Structured Transition Systems. In *ICALP 1987*, volume 267 of *LNCS*, pages 499–508, Springer, 1987.

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

[4] A. Finkel, G. Geeraerts, J.-F. Raskin, and L. Van Begin. A counter-example to the minimal coverability tree algorithm. Technical Report 535, Universite Libre de Bruxelles, 2005.

[5] A. Finkel and J. Goubault-Larrecq. Forward Analysis for WSTS, Part I: Completions. In *STACS 2009*, pages 433–444, 2009.

[6] A. Finkel and J. Goubault-Larrecq. Forward Analysis for WSTS, Part II: Complete WSTS. In *ICALP 2009*, volume 5556 of *LNCS*, pages 188–499, Springer, 2009.

[7] G. Geeraerts, J.-F. Raskin, and L. Van Begin. On the efficient computation of the minimal coverability set of Petri nets. *International Journal of Foundations of Computer Science*, 21(2):135–165, 2010.

[8] R.M. Karp and R.E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

[9] B. König and V. Koziura. Incremental construction of coverability graphs. *Information Processing Letters*, 103(5):203–209, 2007.

[10] P.-A. Reynier and F. Servais. Minimal coverability set for Petri nets: Karp and Miller algorithm with pruning. In L. Petrucci and L. M. Kristensen, editors, *PETRI NETS 2011*, volume 6709 of *LNCS*, pages 69–88. Springer, 2011.

[11] A. Valmari and H. Hansen. Old and New Algorithms for Minimal Coverability Sets. In S. Haddad and L. Pomello, editors, *PETRI NETS 2012*, volume 7347 of *LNCS*, pages 12–21. Springer, 2012.