Constructing Minimal Coverability Sets

Artturi Piipponen and Antti Valmari

Department of Mathematics
Tampere University of Technology
P.O. Box 553, FI–33101 Tampere, Finland
{artturi.piipponen,antti.valmari}@tut.fi

Abstract. This publication addresses two bottlenecks in the construction of minimal coverability sets of Petri nets: the detection of situations where the marking of a place can be converted to ω, and the manipulation of the set A of maximal ω-markings that have been found so far. For the former, a technique is presented that consumes very little time in addition to what maintaining A consumes. It is based on Tarjan's algorithm for detecting maximal strongly connected components of a directed graph. For the latter, a data structure is introduced that resembles BDDs and Covering Sharing Trees, but has additional heuristics designed for the present use. Results from initial experiments are shown. They demonstrate significant savings in running time and varying savings in memory consumption compared to an earlier state-of-the-art technique.

Keywords: coverability set, Tarjan's algorithm, antichain data structure.

1 Introduction and Notation

The background of this work would be very difficult to introduce without first making certain notions precise. Therefore, this section is an interleaving of definitions and the discussion of the background.

A very well-known form of Petri nets is *place/transition net* $(P, T, W, \hat{M})$. It consists of a set P of *places*, set T of *transitions* (such that $P \cap T = \emptyset$), function $W : (P \times T) \cup (T \times P) \mapsto \mathbb{N}$ of *weights* and the *initial marking* $\hat{M}$. In this publication P and T are finite. A *marking* M is a vector of $|P|$ natural numbers. A transition t is *enabled at* M, denoted with $M\,[t\rangle$, if and only if $M(p) \geq W(p, t)$ for every $p \in P$. Then t may *occur* yielding the marking M' such that $M'(p) = M(p) - W(p, t) + W(t, p)$ for every $p \in P$. This is denoted with $M\,[t\rangle\, M'$. It is also said that t is *fired* at M yielding M'. The notation is extended to sequences of transitions in the natural way. A marking M' is *reachable from* M if and only if there is $\sigma \in T^*$ such that $M\,[\sigma\rangle\, M'$.

The set of *reachable markings* (that is, markings that are reachable from the initial marking) of a finite place/transition net is not necessarily finite. However, there always is a finite *coverability set* of certain kind of extended markings that can be used for some of the same purposes as the set of reachable markings is often used [6]. We call them *ω-markings.* An ω-marking is a vector of $|P|$

P.A. Abdulla and I. Potapov (Eds.): RP 2013, LNCS 8169, pp. 183–195, 2013.

elements of the set $\mathbb{N} \cup \{\omega\}$, where ω intuitively denotes "unbounded". The enabledness and occurrence rules of transitions are extended to ω-markings with the conventions that for every $i \in \mathbb{N}$, $\omega \geq i$ and $\omega + i = \omega - i = \omega$.

We say that M' *covers* M if and only if $M(p) \leq M'(p)$ for every $p \in P$. We say that M is *a limit* of a set $\mathcal{M}$ of ω-markings if and only if $\mathcal{M}$ contains $M_0 \leq M_1 \leq \ldots$ such that for every $p \in P$, either $M(p) = \omega$ and $M_i(p)$ grows without limit as i grows, or there is i such that $M(p) = M_i(p) = M_{i+1}(p) = \ldots$.

A coverability set is any set $\mathcal{M}$ that satisfies the following conditions:

1. Every reachable marking M is covered by some $M' \in \mathcal{M}$.
2. Every $M \in \mathcal{M}$ is a limit of reachable markings.

A coverability set is not necessarily finite. Indeed, the set of reachable markings is a coverability set. Fortunately, there is a unique *minimal coverability set* that is always finite [4]. It has no other coverability set as a subset, and no ω-marking in it is covered by another ω-marking in it. It consists of the maximal elements of the set of the limits of the set of reachable markings.

The construction of coverability sets resembles the construction of the set of reachable markings but has additional features. A central idea is that if $M_0 [\sigma\rangle M [t\rangle M' > M_0$, then the sequence σt can occur repeatedly without limit, making the markings of those p grow without limit that have $M'(p) > M_0(p)$, while the remaining p have $M'(p) = M_0(p)$. The limit of the resulting markings is M'', where $M''(p) = \omega$ if $M'(p) > M_0(p)$ and $M''(p) = M_0(p)$ otherwise.

Roughly speaking, instead of storing M' and remembering that $M [t\rangle M'$, most if not all algorithms store M'' and remember that $M \overset{\omega}{-t\rightarrow} M''$, where $M \overset{\omega}{-t\rightarrow} M''$ denotes that M'' was obtained by firing t at M and then possibly adding ω-symbols to the result. However, this is not precisely true for four reasons.

First, the algorithms need not remember that $M \overset{\omega}{-t\rightarrow} M''$. It suffices to remember that there is t such that $M \overset{\omega}{-t\rightarrow} M''$. Second, instead of $M_0 [\sigma\rangle M$ the algorithms use $M_0 \overset{\omega}{-\sigma\rightarrow} M$, because they only have access to the latter.

Third, after firing t at M, an algorithm may use more than one M_0 and σ that have $M_0 \overset{\omega}{-\sigma\rightarrow} M$ to add ω-symbols. The *pumping operation* assigns ω to those $M'(p)$ that have $M_0(p) < M'(p) < \omega$. It may be triggered when the algorithm detects that the *pumping condition* holds with M_0. It holds with M_0 when there is $\sigma \in T^*$ such that $M' > M_0 \overset{\omega}{-\sigma\rightarrow} M$, where M' has been obtained by firing t at M and then doing zero or more pumping operations. The algorithms in [9] and this publication never fail to do the pumping operation when the pumping condition holds, but this is not necessarily true of all algorithms.

Fourth, after firing $M [t\rangle M'$ and doing zero or more pumping operations, an algorithm may reject the resulting M', if it is covered by some already stored ω-marking M''. The intuition is that whatever M' could contribute to the minimal coverability set, is also contributed by M''. So M' need not be investigated.

Whether $M \overset{\omega}{-t\rightarrow} M'$ holds depends on not just M, t, and M', but also on what the algorithm has done before trying t at M. So the precise meaning of the notation $M \overset{\omega}{-t\rightarrow} M'$ depends on the particular algorithm. The meaning used in this publication will be made precise in Section 2.

Some ideas for speeding up the construction of minimal coverability sets have been suggested [4,5,7]. However, [9] gave both theoretical, heuristic, and experimental evidence that a straightforward approach is very competitive, when ω-markings are constructed in depth-first or so-called most tokens first order and pumping conditions are always detected when possible. Nevertheless, as was pointed out in [9], performance measurements must be taken with more than one grain of salt. The running time of an algorithm may depend dramatically on the order in which the transitions are listed in the input, and sorting the transitions according to a natural heuristic does not eliminate this effect.

The algorithm in [9] maintains the set A of maximal ω-markings that have been constructed so far, and most others maintain something similar (but not necessarily precisely the same). ("A" stands for "antichain".) At a low level, the most time-consuming operations in [9] — and probably also in most, if not all, other minimal coverability set construction algorithms — are the manipulation of A and the detection of pumping conditions. In this publication we present a significant improvement to both.

The overall structure of our new algorithm is presented in Section 2. The detection of the pumping condition involves finding out that $M_0 -\sigma\overset{\omega}{\rightarrow} M$. Section 3 describes how *Tarjan's algorithm* for detecting maximal strongly connected components of a directed graph [8,1] can be harnessed to convert this otherwise expensive test to only consume constant time. A data structure that improves the efficiency of maintaining A is introduced in Section 4. It uses ideas from BDDs [2] and covering sharing trees [3], and has heuristics designed for coverability sets. An additional optimisation is discussed in Section 5. Section 6 presents some performance measurements without and with using the new ideas.

2 Overall Algorithm

Figure 1 shows the new minimal coverability set construction algorithm of this publication in its basic form. Variants of it will be discussed in Section 6.

Lines 1, 3–6, 13–16, 18, 20–24, and 27 implement most of the coverability set construction algorithm of [9]. Let us discuss them in this section. The remaining lines may be ignored until they are discussed in later sections.

The set A contains the maximal ω-markings that have been found so far. Upon termination it contains the result of the algorithm. Its implementation will be discussed in Section 4. In addition to what is explicit in Fig. 1, the call on line 22 may remove elements from A in favour of a new element M' that strictly covers them. We will discuss this in detail later.

The set F is a hash table. ω-Markings are added to it at the same time as to A, but they are never removed from it. So always $A \subseteq F$. The attributes of an ω-marking such as $M.\mathsf{tr}$ (discussed soon) are stored in F and not in A. That is, F contains records, each of which contains an ω-marking and some additional information. The reason is that, as we will see later, some information on an ω-marking may remain necessary even after it has been removed from A. Like in [9], F is also used to implement an optimisation that will be discussed together with

```
1   F := {M̂}; A := {M̂}; W.push(M̂); M̂.tr := first transition
2   S.push(M̂); M̂.ready := false; n_f := 1; M̂.index := 1; M̂.lowlink := 1
3   while W ≠ ∅ do
4       M := W.top; t := M.tr; if t ≠ nil then M.tr := next transition endif
5       if t = nil then
6           W.pop
7           activate transitions as discussed in Section 5
8           if M.lowlink = M.index then
9               while S.top ≠ W.top do S.top.ready := true; S.pop endwhile
10          else if W ≠ ∅ then
11              W.top.lowlink := min{W.top.lowlink, M.lowlink}
12          endif
13          go to line 3
14      endif
15      if ¬M [t⟩ then go to line 3 endif
16      M′ := the ω-marking such that M [t⟩ M′
17      if M′ ≤ M then passivate t; go to line 3 endif
18      if M′ ∈ F then
19          if ¬M′.ready then M.lowlink := min{M.lowlink, M′.lowlink} endif
20          go to line 3
21      endif
22      Cover-check(M′, A)      // only keep maximal — may update A and M′
23      if M′ is covered then go to line 3 endif
24      F := F ∪ {M′}; A := A ∪ {M′}; W.push(M′); M′.tr := first transition
25      S.push(M′); M′.ready := false
26      n_f := n_f + 1; M′.index := n_f; M′.lowlink := n_f
27  endwhile
```

Fig. 1. A coverability set algorithm that uses Tarjan's algorithm and some heuristics

lines 18 and 20. Hash tables are very efficient, so F does not cause significant extra cost.

For efficiency, instead of the common recursive implementation, depth-first search is implemented with the aid of a stack which is called W (for work-set). The elements of W are pointers to records in F. Each ω-marking M has an attribute tr that points to the next transition that should be tried at M.

The algorithm starts on lines 1 and 2 by putting the initial marking to all data structures. Roughly speaking, lines 3 to 27 try each transition t at each encountered ω-marking M in depth-first order. (This is not strictly true, because heuristics that are discussed later may prematurely terminate the processing of M and may cause the skipping of some transitions at M.) If M has untried transitions, line 4 picks the next, otherwise lines 6–13 that implement backtracking are executed. Lines 7–12 will be discussed later.

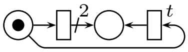
If the picked transition t is disabled at the current ω-marking M, then it is rejected on line 15. Otherwise t is fired at M on line 16. Lines 17 and 19 will be discussed later. If M' has already been encountered, it is rejected on lines 18 and 20. This quick rejection of M' is useful, because reaching the same ω-marking again is expected to be very common, because $M\,[t_1 t_2\rangle\, M_{12}$ and $M\,[t_2 t_1\rangle\, M_{21}$ imply that $M_{12} = M_{21}$. Without lines 18 and 20, M' would be rejected on line 23, but after consuming more time. Line 18 is also needed because of line 19.

The call Cover-check(M', A) first checks whether M' is covered by any ω-marking in A. If yes, then M' is rejected on line 23.

In the opposite case, Cover-check checks whether the pumping condition holds with any $M_0 \in A$. (In [9], the pumping condition was detected for $M_0 \in F$. Theorem 4 will tell why it suffices to use A instead.) If yes, it changes $M'(p)$ to ω for the appropriate places p. Cover-check also removes from A those ω-markings that the updated M' covers strictly. When M is removed, $M.\mathsf{tr}$ is set to nil, so that even if the algorithm backtracks to M in the future, no more transitions will be fired at it. The addition of ω-symbols makes M' grow in the "$\leq$" ordering and may thus make the pumping condition hold with some other M_0. Cover-check continues until there is no $M_0 \in A$ with which the pumping condition holds but the pumping operation has not yet been done. The details of Cover-check will be discussed in later sections.

If M' was not covered, its updated version is added to the data structures on lines 24–26. This implements the entering to M' in the depth-first search.

It is the time to make the notation $M -t\overset{\omega}{\rightarrow} M'$ precise. It denotes that t was fired at M on line 16 resulting in some M'' such that $M'' \not\leq M$, and either $M'' \in F$ held on line 18 (in which case $M' = M''$), or M'' was transformed to M' on line 22 and then added to F on line 24.

Thus $M -t\overset{\omega}{\rightarrow} M'$ is either always false or becomes true during the execution of the algorithm. Even if $M \in F$ and $M\,[t\rangle$, it may be that there never is any M' such that $M -t\overset{\omega}{\rightarrow} M'$. This is the case if M is removed from A and $M.\mathsf{tr}$ is set to nil before t is tried at M, or if the result of trying t at M is rejected on line 17 or 23. With the following Petri net, the latter happens although $M \in A$ when the algorithm has terminated: 

.

The correctness of this approach has been proven in detail in [9]. Intuitively, every ω-marking that is put into F is a limit of reachable markings, because for each p, $M -t\overset{\omega}{\rightarrow} M'$ either mimics $M\,[t\rangle\, M'$, copies ω from $M(p)$ to $M'(p)$, or sets $M'(p)$ to ω as justified by some pumping condition. Pumping operations make progress towards termination. The algorithm does not terminate prematurely, because each time when something is rejected or passivated, something else is kept or remains active that makes at least the same contribution to the final A.

The following details are essential for this publication.

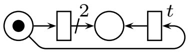
Lemma 1. *For every ω-markings M and M', $p \in P$, $t \in T$, and $\sigma \in T^*$,*

1. *If $M -\sigma\overset{\omega}{\rightarrow} M'$ and $M(p) = \omega$, then $M'(p) = \omega$.*
2. *Assume that $M -t\overset{\omega}{\rightarrow} M'$ and $M(p) < \omega = M'(p)$. After constructing the edge $M -t\overset{\omega}{\rightarrow} M'$, the algorithm does not backtrack from M' before it has investigated all M'' that have $\sigma \in T^*$ such that $M' -\sigma\overset{\omega}{\rightarrow} M''$.*
3. *For every $M_0 \in F$, there is $M_0' \in A$ such that $M_0 \leq M_0'$.*
4. *Every M constructed by the algorithm is a limit of $M_1, M_2, \ldots$ such that there are $\sigma_1, \sigma_2, \ldots$ such that $\hat{M}\,[\sigma_i\rangle\, M_i$ for $i \geq 1$.*

In depth-first search, an ω-marking M is *white* if it has not been found (that is, $M \notin F$); *grey* if it has been found but not backtracked from (that is, $M \in W$); and *black* if it has been backtracked from. If M is black and $M -t\overset{\omega}{\rightarrow} M'$, then M' is grey or black. The grey ω-markings M_i^{g} (where $M_0^{\mathsf{g}} = \hat{M}$) and the $M_{i-1}^{\mathsf{g}} -t_i\overset{\omega}{\rightarrow} M_i^{\mathsf{g}}$ via which they were first found constitute a path from the initial to the current ω-marking.

Lemma 1(1) follows trivially from the transition firing rule. It implies that if M has been found and has ω in some place where the current ω-marking M_{c} does not have ω, then there is no path from M to M_{c}. As a consequence, M and all its descendants are black. They remain black from then on, because a black ω-marking no longer changes colour. This implies Lemma 1(2).

Inspired by the above, we say that M is *ripe* if and only if $M \in F$ and either the algorithm has terminated, or after finding M there has been an instant of time such that for some place p, $M_{\mathsf{c}}(p) < \omega = M(p)$, where M_{c} was the current ω-marking at that time. All descendants of all ripe ω-markings are black.

The following lemma says that the "future" of each ripe ω-marking has been fully covered. This result does not immediately follow from the fact that the descendants of each ripe ω-marking are black, because any such descendant may have been rejected in favour of another ω-marking that strictly covers it (cf. ). For the same reason, the lemma does not promise that its M'_n is obtained via the sequence $M'_0 -t_1 \cdots t_n\overset{\omega}{\rightarrow} M'_n$.

Lemma 2. *If M'_0 is ripe and $M'_0 \geq M_0\,[t_1 \cdots t_n\rangle\, M_n$, then there is M'_n such that M'_n is ripe and $M'_n \geq M_n$. A similar claim holds for $M_0 -t_1 \cdots t_n\overset{\omega}{\rightarrow} M_n$.*

Proof. To prove the first claim, consider the moment when M'_0 became ripe. We use induction on $1 \leq i \leq n$. By Lemma 1(3), there was $M''_{i-1} \in A$ such that $M'_{i-1} \leq M''_{i-1}$. The algorithm had tried t_i at M''_{i-1}. If the result was kept, it qualifies as M'_i, otherwise it was rejected because it was covered by an ω-marking that qualifies as M'_i. By Lemma 1(1), M'_i has ω-symbols in at least the same places as M'_{i-1}. So M'_i is ripe and $M'_i \geq M_i$.

The above proof referred to a certain moment in time to ensure that $M''_{i-1} \in A$. Later $M''_{i-1} \in A$ may cease to hold, but what was proven remains valid. We point out for the sequel that if $M'_{i-1}(p) < \omega$ then $M'_i(p) - M'_{i-1}(p) \geq M_i(p) - M_{i-1}(p)$, because the firing of t_i has the same effect to the ω-marking of p in both cases, and the possible additional operations may not reduce $M'_i(p)$.

With $M_{i-1} -t_i\overset{\omega}{\rightarrow} M_i$, there may be $p_1, \ldots, p_k$ such that $M_{i-1}(p_j) < \omega = M_i(p_j)$. Given M'_{i-1}, we apply induction on $1 \leq j \leq k$ to obtain an M'_i that has the required properties. Let $M_{i,j}$ be the ω-marking just after the algorithm made $M_i(p_j) = \omega$. So $M_{i-1}\,[t_i\rangle\, M_{i,0}$ and $M_{i,k} = M_i$. The first claim yields $M'_{i,0}$. Let σ_j be the sequence that justified converting $M_{i,j-1}$ to $M_{i,j}$. There are $\dot{M}_{i,j}$ and $\ddot{M}_{i,j}$ such that $M_{i,j-1}\,[\sigma_j\rangle\, \dot{M}_{i,j}\,[\sigma_j\rangle\, \ddot{M}_{i,j}$. The first claim can be applied to this sequence, yielding $\dot{M}'_{i,j}$ and $\ddot{M}'_{i,j}$. If $\dot{M}'_{i,j}(p) < \omega$, then $\ddot{M}'_{i,j}(p) - \dot{M}'_{i,j}(p) \geq \ddot{M}_{i,j}(p) - \dot{M}_{i,j}(p) \geq 0$. Therefore, $\dot{M}'_{i,j} \leq \ddot{M}'_{i,j}$. Because of the use of A in the proof of the first claim, $\dot{M}'_{i,j} \not< \ddot{M}'_{i,j}$. So $\ddot{M}'_{i,j} = \dot{M}'_{i,j}$, implying $\ddot{M}'_{i,j}(p) = \omega$ if

$\ddot{M}_{i,j}(p) > \dot{M}_{i,j}(p)$, that is, if $M_{i,j}(p) = \omega > M_{i,j-1}(p)$. With the remaining p, $\ddot{M}'_{i,j}(p) \geq M'_{i,j-1}(p) \geq M_{i,j-1}(p) = M_{i,j}(p)$. These yield $M_{i,j} \leq \ddot{M}'_{i,j}$. So $\ddot{M}'_{i,j}$ qualifies as the $M'_{i,j}$. Choosing $M'_i = M'_{i,k}$ completes the proof of step i. □

3 Constant-Time Reachability Testing

A *maximal strongly connected component* or *strong component* of a directed graph (V, E) is a maximal set of vertices $V' \subseteq V$ such that for any two vertices u and v in V', there is a path from u to v. The strong components constitute a partition of V. Tarjan's algorithm [8,1] detects strong components in time $O(|V| + |E|)$. It is based on depth-first search of the graph. It is slower than depth-first search only by a small constant factor.

In our case, V consists of all ω-markings that are encountered during the construction of the minimal coverability set, that is, those that are (eventually) stored in F. The edges are defined by $(M, M') \in E$ if and only if there is $t \in T$ such that $M -t\overset{\omega}{\rightarrow} M'$. This notion, and thus also V and E, depends on the order in which transitions are picked on lines 1, 4, and 24 in Fig. 1. Fortunately, this does not confuse Tarjan's algorithm, because an edge is introduced either when the algorithm is ready to investigate it or not at all.

In Fig. 1, Tarjan's algorithm is represented via lines 2, 8–12, 19, and 25–26. In addition to W, it uses another stack, which we call S. Also its elements are pointers to records in F.

Tarjan's algorithm also uses two attributes on each ω-marking called index and $\mathsf{lowlink}$. The index is a running number that the ω-marking gets when it is encountered for the first time. It never changes afterwards. The lowlink is the smallest index of any ω-marking that is known to belong to the same strong component as the current ω-marking. When backtracking and when encountering an ω-marking that has already been visited and is in the same strong component with the current ω-marking, the lowlink value is backward-propagated and the smallest value is kept. The lowlink value is not backward-propagated from ω-markings that belong to already completed strong components.

Each ω-marking is pushed to S when it is found and popped from S when its strong component is ready, and it never returns to S. Presence in S is tested quickly via an attribute ready that is updated when S is manipulated.

The following is the central invariant property of Tarjan's algorithm:

Lemma 3. *Let $M_0 \in F$. There is a path from M_0 to the M of Fig. 1 if and only if $M_0 \in S$. If $M_0 \notin S$, then every ω-marking to which there is a path from M_0 is neither in S nor in W.*

Cover-check(M', A) has to find each M_0 such that $M_0 \in A$ and $M_0 < M'$, because they have to be removed from A. When it has found such an M_0, it checks whether $M_0.\mathsf{ready} = \mathsf{false}$, that is, whether $M_0 \in S$. This is a constant-time test that reveals whether there is a path from M_0 to M'. In this way Cover-check detects each valid pumping condition where $M_0 \in A$ with a constant amount of additional effort per removed element of A.

If ω-symbols are added to M', then the checking is started again from the beginning, because the updated M' may cover strictly elements of A that the original M' did not cover strictly. Also they have to be removed and checked against the pumping condition. When Cover-check terminates, there are no unused instances of the pumping condition where $M_0 \in A$, and A no longer contains ω-markings that are strictly covered by M'.

This method only detects the cases where $M_0 \in A$, while [9] uses $M_0 \in F$. Fortunately, the following theorem tells that it does not make a difference.

Theorem 4. *The algorithm in Fig. 1 constructs the same ω-markings as it would if F were used instead of A in the pumping conditions.*

Proof. In this case the pumping condition is $M' > M_0 -\sigma\xrightarrow{\omega} M\,[t\rangle\, M_\#$, where $\sigma \in T^*$ and M' has been made from $M_\#$ by replacing the contents of zero or more places by ω. By Lemma 3, from each ω-marking in S and from no ω-marking in $F \setminus S$ there is a path to M. The pumping condition triggers the updating of M' to M'' such that for every $p \in P$, either $M_0(p) = M'(p) = M''(p)$ or $M_0(p) < M'(p) \leq M''(p) = \omega$.

We prove the claim by induction. We show that in every pumping operation, A causes (at least) the same updates as F, the induction assumption being that also in the previous times A caused the same updates as F. At least the same updates implies precisely the same updates, because $A \subseteq F$.

Let the pumping condition hold such that $M_0 \in F$. If $M_0 \in A$, then the induction step holds trivially. From now on $M_0 \notin A$.

Lemma 1(3) yields $M_0' \in A$ such that $M_0 < M_0'$. It was found after M_0, because otherwise M_0 would have been rejected on line 23. Because $M_0 -\sigma\xrightarrow{\omega} M$ now, M_0 is now in S. So M_0 was in S when M_0' was found. At that moment there was a path from M_0 to what was then M, that is, there is ρ such that $M_0 -\rho\xrightarrow{\omega} M_0'$. So the pumping condition held with $M_0 \in F$. By the induction assumption, ω-symbols were added to M_0'. Therefore, for every $p \in P$, either $M_0'(p) = M_0(p)$ or $M_0'(p) = \omega$.

Return to the moment when ω-symbols are added to M'. If $M_0' \in S$, there is a path from M_0' to M. If $M_0'(p) = \omega$, then also $M'(p) = \omega$ by Lemma 1(1). We already saw that if $M_0'(p) \neq \omega$, then $M_0'(p) = M_0(p)$. So the pumping condition holds with M_0' and causes precisely the same result as M_0 causes.

The case remains where $M_0' \notin S$. If there is M'' such that $M_0' -\sigma\xrightarrow{\omega} M''$, then $M'' \neq M$. Furthermore, $M'' \geq M$, because $M_0 < M_0'$, and $M_0 -\rho\xrightarrow{\omega} M_0'$ implies that ω-symbols are added to (or are already in) at least the same places along $M_0' -\sigma\xrightarrow{\omega} M''$ as along $M_0 -\sigma\xrightarrow{\omega} M$. So $M'' > M$. But that is a contradiction with the fact that M is the current ω-marking.

So there are σ_i, t_i, σ_i', and M_i' such that $\sigma_i t_i \sigma_i' = \sigma$, $M_0' -\sigma_i\xrightarrow{\omega} M_i'$, but t_i was not tried at M_i' or the result of trying it was rejected. We discuss the case that t_i was not tried. The other case is similar.

Failure to try t_i implies that there was M_i'' such that $M_i' < M_i''$. If M_i is the ω-marking such that $M_0 -\sigma_i\xrightarrow{\omega} M_i$, then $M_i \leq M_i'$. Thus M_i'' was found after t_i was fired at M_i but before M_i' was backtracked from. Because M_i is in S now, it was in S when M_i'' was found. So there was a path from M_i to M_i'', triggering the pumping condition. There is at least one p such that $M_i(p) \leq M_i'(p) < M_i''(p)$. Therefore, M_i'' has more ω-symbols than M_i'. So M_i'' is ripe. Lemma 2 says that M is covered by some ripe M_n'', which is a contradiction with the fact that M is the current ω-marking. □

4 A Data Structure for Maximal ω-Markings

This section presents a data structure for maintaining A. It has been inspired by Binary Decision Diagrams [2] and Covering Sharing Trees [3]. However, ω-markings are only added one at a time. So we are not presenting a symbolic approach. The purpose of using a BDD-like data structure is to facilitate fast detection of situations where an ω-marking covers another. The details of the data structure have been designed accordingly. We will soon see that they make certain heuristics fast.

We call the M' on line 22 of Fig. 1 the *new* ω-marking, while those stored in A are *old*. Cover-check first uses the data structure to detect if M' is covered by any old ω-marking. If yes, then nothing more needs to be done. In the opposite case, Cover-check then searches for old ω-markings that are covered by M'. By the first search, they are strictly covered. This search cannot be terminated when one is found, because Cover-check has to remove all strictly covered ω-markings from A and use them in the pumping test. Therefore, finding the first one quickly is less important than finding quickly the first old ω-marking that covers M'. As a consequence, the data structure has been primarily optimised to detect if any old ω-marking covers the new one, and secondarily for detecting covering in the opposite order.

Let $\underline{M}(p) = M(p)$ if $M(p) < \omega$ and $\underline{M}(p) = 0$ otherwise. Let $\overline{M}(p) = 1$ if $M(p) = \omega$ and $\overline{M}(p) = 0$ otherwise. In this section we assume without loss of generality that $P = \{1, 2, \ldots, |P|\}$.

The data structure consists of $|P| + 1$ layers. The topmost layer is an array of pointers that is indexed with the total number of ω-symbols in an ω-marking, that is, $\sum_{p=1}^{|P|} \overline{M}(p)$. This number can only be in the range from 0 to $|P|$, so a small array suffices. An array is more efficient than the linked lists used at lower layers. The pointer at index w leads to a representation of the set of ω-markings in A that have w ω-symbols each.

Layer $|P|$ consists of $|P| + 1$ linked lists, one for each total number of ω-symbols. Each node v in the linked list number w contains a value $v.m$, a pointer to the next node in the list, and a pointer to a representation of those ω-markings in A that have $\sum_{p=1}^{|P|} \overline{M}(p) = w$ and $\sum_{p=1}^{|P|} \underline{M}(p) = v.m$. The list is ordered in decreasing order of the m values, so that the ω-markings that have the best chance of covering M' come first.

Let $1 \leq \ell < |P|$. Each node v on layer ℓ contains two values $v.w$ and $v.m$, a link to the next node on the same layer, and a link to a node on layer $\ell - 1$. Of course, this last link is nil if $\ell = 1$. The node represents those ω-markings in A that have $\sum_{p=1}^{\ell} \overline{M}(p) = v.w$, $\sum_{p=1}^{\ell} \underline{M}(p) = v.m$, and the places greater than

ℓ have the unique ω-markings determined by the path that leads to v, as will be discussed below. If more than one path leads to v, then v represents more than one subset of A. They are identical with respect to the contents of the places from 1 to ℓ, but differ on at least one place above ℓ. The lists on these layers are ordered primarily in increasing order of the w values and secondarily in increasing order of the m values.

Like in BDDs, nodes with identical values and next-layer pointers are fused. To be more precise, when a node is being created, it is first checked whether a node with the desired contents already exists, and if yes, it is used instead. A specific hash table makes it fast to find existing nodes based on their contents.

Because every ω-marking that is in A is also in F, it has an explicit representation there. As a consequence, unlike with typical applications of BDDs, the storing of dramatically big numbers of ω-markings is not possible. As was mentioned above, the goal is not to do symbolic construction of ω-markings. Even so, the fusing of identical nodes pays off. Otherwise, for each ω-marking, A would use a whole node on layer 1 and additional partially shared nodes on other layers, while F represents the ω-marking as a dense vector of bytes. So A would use much more memory for representing each ω-marking than F uses.

Consider the checking whether $M' \leq M$, where M' is the new and M is any old ω-marking. After entering a node v at level $\ell - 1$, $M(\ell)$ is computed as $u.w - v.w$ and $u.m - v.m$, where u is the node at level ℓ from which level $\ell - 1$ was entered. If $M'(\ell) > M(\ell)$, then the traversal backtracks to u. This is because, thanks to the ordering of the lists, both v and the subsequent nodes in the current list at level $\ell - 1$ correspond to too small a marking in $M(\ell)$.

On the other hand, M may be rejected also if $\sum_{p=1}^{\ell-1} \overline{M}'(p) > \sum_{p=1}^{\ell-1} \overline{M}(p)$. To quickly detect this condition, an array wsum is pre-computed such that $\mathsf{wsum}[\ell] = \sum_{p=1}^{\ell} \overline{M}'(p)$:

$$\mathsf{wsum}[1] := \overline{M}'(1)$$
$$\textbf{for } p := 2 \textbf{ to } |P| \textbf{ do } \mathsf{wsum}[p] := \mathsf{wsum}[p-1] + \overline{M}'(p) \textbf{ endfor}$$

This pre-computation introduces negligible overhead. The condition becomes $\mathsf{wsum}[\ell - 1] > v.w$, which is a constant time test. If this condition is detected, layer $\ell - 2$ is not entered from the current node, but the scanning of the list on layer $\ell - 1$ is continued.

The current node v (but not the current list) is rejected also if $\mathsf{wsum}[\ell - 1] = v.w$ and $\mathsf{msum}[\ell - 1] > v.m$, where $\mathsf{msum}[\ell] = \sum_{p=1}^{\ell} \underline{M}'(p)$. There also is a third pre-computed array mmax with $\mathsf{mmax}[\ell]$ being the maximum of $\underline{M}(1)$, $\underline{M}(2)$, $\ldots$, $\underline{M}(\ell)$. It is used to reject v when

$$\mathsf{wsum}[\ell-1] < v.w \;\text{ and }\; (v.w - \mathsf{wsum}[\ell-1]) \cdot \mathsf{mmax}[\ell-1] + v.m < \mathsf{msum}[\ell-1]\ .$$

The idea is that considering the places from 1 to $\ell - 1$, each extra ω-symbol in M covers at most $\mathsf{mmax}[\ell - 1]$ ordinary tokens in M'. There is thus a fast heuristic for each of the cases $\mathsf{wsum}[\ell-1] < v.w$, $\mathsf{wsum}[\ell-1] = v.w$, and $\mathsf{wsum}[\ell-1] > v.w$. These heuristics are the reason for storing $\sum_{p=1}^{\ell} \overline{M}(p)$ and $\sum_{p=1}^{\ell} \underline{M}(p)$ into the node instead of $M(\ell)$.

Consider the situation where none of the above heuristics rejects v. Then layer $\ell - 2$ is entered from v. If it turns out that M' is covered, then the search need not be continued. In the opposite case, it is marked into v that layer $\ell - 2$ was tried in vain. If v is encountered again during the processing of the same M', this mark is detected and v is not processed further. To avoid the need of frequently resetting these marks, the mark is a running number that is incremented each time when the processing of a new M' is started. The marks are reset only when this running number is about to overflow the range of available numbers. This trick is from [9].

Similar heuristics are used for checking whether the new ω-marking strictly covers any ω-marking in A. The biggest difference is that now the search cannot be stopped when such a situation is found, as has been explained above. The condition "strictly" need not be checked, because if $M' \in A$, then M' is rejected by the first search or already on line 20. The third heuristic mentioned above is not used, because information corresponding to $\mathsf{mmax}[\ell]$ cannot be obtained cheaply for ω-markings in A. It would not necessarily be unique, and it would require an extra field in the record for the nodes.

5 Transition Removal Optimisation

By Lemma 1(1), if $M \xrightarrow{t}{}^{\omega} M'$ and there is p such that $M(p) < M'(p) = \omega$, then the ω-marking of p will remain ω until the algorithm backtracks to M. If the firing of a transition does not increase the ω-marking of any place, that is, if $M \xrightarrow{t}{}^{\omega} M'$ and $M' \leq M$, then t is *useless*. Lines 22 and 23 would reject M', had it not already been done on line 17. A transition that is not originally useless in this sense becomes useless, if ω-symbols are added to each p such that $W(p,t) < W(t,p)$.

Lines 7 and 17 implement an additional optimisation based on these facts. The "first transition" and "next transition" operations in Fig. 1 pick the transitions from a doubly linked list which we call the *active list*. When t that is in the active list has become useless, that is detected on Line 17. Then t is linked out from the active list and inserted to a singly linked list that starts at $\mathsf{passive}[c]$, where c is the number of locations in W where ω-symbols have been added, and $\mathsf{passive}$ is an array of size $|P|+1$. There also is a similarly indexed array toW such that $\mathsf{toW}[c]$ points to the most recent location in W where ω-symbols have been added. The forward and backward links of t still point to the earlier successor and predecessor of t in the active list. This operation takes constant time.

From then on, t is skipped at no additional cost until the algorithm backtracks to M. This moment is recognized from the current top of W getting below $\mathsf{toW}[c]$. Then all transitions from $\mathsf{passive}[c]$ are removed from there and linked back to their original places in the active list, and c is decremented. Because each passive list is manipulated only at the front, it releases the transitions in opposite order to in which they were inserted to it. This implies that the original ordering of the active list is restored when transitions are linked back to it, and the "next transition" operation is not confused. Also the linking back is constant time per transition.

Table 1. Initial measurements with some versions of the new algorithm

	mesh2x2			mesh3x2			AP13a			smallT5x2			largeT2x15x2		
$\|A\|$ $\|F\|$	256		316	6400		7677	1245		1281	31752		31752	32768		32768
$\|S\|$			316			7677			65			50			32768
≈ [9]	4	5	23	696	739	718	60	66	467	3151	3182	1736	7121	7142	3328
no node fusion	2	2	70	46	48	1389	49	56	4393	67	72	4058	231	237	20736
basic new	3	4	24	52	68	732	57	62	850	80	89	1738	248	273	4224
no tr. rem.	4	4	24	64	76	732	56	60	850	81	91	1738	261	270	4224
partial F	3	4	22	55	61	672	54	71	401	152	162	4	250	258	3968

If the check on line 17 were removed, the algorithm would still reject M', but in the worst case that might happen much later in Cover-check. This heuristic is very cheap and may save time by rejecting M' early. Unfortunately, in our initial experiments (Section 6) it did not perform well.

6 Experiments and Conclusions

In this section we present some of the first experiments that we have made with an implementation of our new algorithm. To make comparison of running times reasonable, we compare the new implementation to a slightly improved version of the implementation in [9] (better hash function, etc.). Both have been written in the same programming language (C++) and were executed on the same computer.

The Petri net mesh2x2 is the heaviest example from [5,7]. It has been included to point out that the examples from [5,7] are not challenging enough for testing the new implementation. Mesh3x2 is a bigger version of it. AP13a has been modified from users.cecs.anu.edu.au/~thiebaux/benchmarks/petri/ by Henri Hansen, to present a somewhat bigger challenge. SmallT5x2 was designed for this publication, to have a small S and offer many possibilities for fusing nodes in the representation of A. LargeT2x15x2 had precisely the opposite design goal.

The results are in Table 1. The second row shows the final sizes of the sets A and F, assuming that ω-markings are never removed from F. The third row shows the maximal sizes of S. The sizes of W are not shown, because always $|W| \leq |S|$.

The next five rows show results for various implementations. "≈ [9]" was explained above. "Basic new" is the algorithm described in this publication. "No node fusion" is otherwise the same as "Basic new", but nodes in the data structure for A that have the same values and next-layer pointers are not fused. "No tr. rem." is otherwise the same as "Basic new", but the optimization discussed in Section 5 is not in use. "Partial F" is otherwise the same as "Basic new", but when an ω-marking is removed from S it is also removed from F. If such an ω-marking is constructed anew, it is covered by some ω-marking in A, and thus will be rejected on line 23 at the latest.

For each implementation and Petri net, the first two numbers report the shortest and longest running times for five identical measurements in milliseconds.

The third number is the amount of memory consumed, measured in kilobytes (1024 bytes), assuming that memory is reserved for A, S, and the base table of F only as needed. For W, the same amount of memory was reserved as for S. These numbers are theoretical in the sense that the implementation did not use dynamically growing arrays in reality. We plan to fix this defect in future measurements.

To protect against programming errors, we checked for each Petri net that every version returned the same A.

All new versions are significantly faster than the one in [9] excluding AP13A where all versions are roughly equally fast, and mesh2x2 that is too small for a meaningful comparison. Although precise comparison is not possible, the results on mesh2x2 make it obvious that the new implementation outperforms the one in [7]. The memory consumptions of the four new versions relate to each other as one would expect. Compared to [9] whose A was a doubly linked list of the same records that F used, the new versions consume much more memory except when the fusion of nodes and the removal of ω-markings from F have a big effect. While [9] uses two additional pointers per ω-marking to represent A, the new versions have the complicated structure described in Section 4. Furthermore, S was absent from [9].

Acknowledgements. We thank the anonymous reviewers for their effort.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)
3. Delzanno, G., Raskin, J.-F., Van Begin, L.: Covering Sharing Trees: A Compact Data Structure for Parameterized Verification. Software Tools for Technology Transfer 5(2-3), 268–297 (2004)
4. Finkel, A.: The Minimal Coverability Graph for Petri Nets. In: Rozenberg, G. (ed.) APN 1993. LNCS, vol. 674, pp. 210–243. Springer, Heidelberg (1993)
5. Geeraerts, G., Raskin, J.-F., Van Begin, L.: On the Efficient Computation of the Minimal Coverability Set of Petri Nets. International Journal of Foundations of Computer Science 21(2), 135–165 (2010)
6. Karp, R.M., Miller, R.E.: Parallel Program Schemata. Journal of Computer and System Sciences 3(2), 147–195 (1969)
7. Reynier, P.-A., Servais, F.: Minimal Coverability Set for Petri Nets: Karp and Miller Algorithm with Pruning. In: Kristensen, L.M., Petrucci, L. (eds.) PETRI NETS 2011. LNCS, vol. 6709, pp. 69–88. Springer, Heidelberg (2011)
8. Tarjan, R.E.: Depth-First Search and Linear Graph Algorithms. SIAM J. Computing 1(2), 146–160 (1972)
9. Valmari, A., Hansen, H.: Old and New Algorithms for Minimal Coverability Sets. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 208–227. Springer, Heidelberg (2012) (Extended version has been accepted to Fundamenta Informaticae)