A Tool for the Coverability Problems in Petri Nets

Alain Finkel[1], Serge Haddad[12], and Igor Khmeniltsky[12]

[1] LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
{finkel,haddad,khmelnitsky}@lsv.fr
[2] Inria, France

Goals of CovMin. Petri nets are an infinite-state model for specification and verification of concurrent systems. An important generic problem of this model is coverability, which asks the following question: Given a Petri net $\mathcal{N}$ and two markings $\mathbf{m}_0, \mathbf{m}_t \in \mathbb{N}^p$, does there exist a firing sequence σ such that $\mathbf{m}_0 \xrightarrow{\sigma} \mathbf{m}' \geq \mathbf{m}_t$. Another related problem, is the coverability set problem which asks: Given a Petri net $\mathcal{N}$ and an initial marking $\mathbf{m}_0$, produce a finite set $Clover(\mathcal{N}, \mathbf{m}_0)$ of markings such that a marking $\mathbf{m}$ is coverable iff there exists an ω-marking $\mathbf{m}' \in Clover(\mathcal{N}, \mathbf{m}_0)$ such that $\mathbf{m}' \geq \mathbf{m}$. The computational complexity of these problems is very high: The coverabilty problem is EXPSPACE-complete while the size of the coverabilty set may be non primitive recursive. The tool CovMin was designed to efficiently solve these problems.

Functionalities of CovMin. The coverability set problem was first solved by the Karp-Miller's algorithm [5]. However the space requirement of this algorithm may be huge. To address this issue, Alain Finkel designed a new algorithm (denoted AF) [2] Unfortunately, it was shown that this algorithm was incomplete [3]. Thus there were a few attempts ([7],[6],[4]) to fix this algorithm by keeping more information during the execution. However the size of this extra memory may be non primitive recursive. In a paper submitted to an international conference, we design an algorithm that fixes the AF using at most 2-EXPSPACE of extra memory by reification of accelerations. Furthermore CovMin solves the coverability problem via the coverability set.

Technical details. CovMin was implemented in Python 3.7 using the Numpy and the Z3-solver libraries, and it is around 2000 lines of code. As input CovMin imports Petri net in .spec format from Mist. CovMin and the benchmarks discussed below can be found here: https://github.com/IgorKhm/MinCov.

Benchmarks. All the benchmarks were performed on a single computer equipped with Intel i5-8250U CPU with 4 cores, 16 GB of memory and Ubuntu Linux 18.03.

• **Minimal coverability set:** We compare CovMin with the tool MP [6], the tool VP [7] and the original algorithm AF. Both MP and VP tools were sent to us by the courtesy of the authors.

We ran two types of benchmarks: (1) 123 standard benchmarks from the literature Table 1, (which were taken from [1]); (2) 100 randomly generated Petri nets Table 1. The execution time of the tools was limited to 900 seconds.

Each of these tables is a summary of all the instances of the benchmarks. The first column shows the number of instances that the tool timed out on. The next two columns only consider the instances that didn't time out on any of the tools. The second column consists of the total time. The third one consists of the peak number of nodes where for CovMin we also took into account the number of accelerations.

In the benchmarks from the literature we observed that the instances that timed out from CovMin are included in those of AF and MP. However there were instances that timed out on VP but did not time out on CovMin and vice versa. W.r.t. memory requirements AF and CovMin have the least number of nodes. CovMin is the second fastest tool, and compared to VP it is 2.6 times slower. A possible explanation would be that VP is implemented in C++. To get a fairer comparison, we counted for VP and CovMin the number of comparisons between two ω-markings and observed that VP has approximatively 1.5 less comparisons then CovMin.

Table 1: Benchmarks for the clover construction.

Benchmarks from the literature

	T/O	#Nodes	Time
`CovMin`	16/123	48218	391
`AF`	19/123	45660	431
`VP`	15/123	75225	163
`MP`	24/123	478681	2304

Random benchmarks

	T/O	#Nodes	Time
`CovMin`	14/100	61164	225
`AF`	16/100	63275	302
`VP`	15/100	208134	16
`MP`	21/100	755129	2818

• **Coverability:** We compare `CovMin` to the tool `qCover` [1] on the set of benchmarks from the literature Table 2. We split the results into safe instances (coverable) and unsafe ones (not coverable). In both categories we counted the number of instances that the tools successfully verified and the total time it took to run the instances that did not time out.

Table 2: Benchmarks for the coverability problem.

	Time Unsafe	T/O Unsafe	Time safe	T/O safe
`CovMin`	854	1/60	3623	53/115
`qCover`	3067	26/60	1965	11/115
`CovMin` ‖ `qCover`	41	2/60	3593	11/115

We observed that the two tools are complementary, i.e. `qCover` is faster at proving that an instance is safe and `CovMin` is faster at proving that an instance is unsafe. Therefore, by splitting the processing time between them we get better results. The third row of Table 2 represents this parallel execution, where the time for each instance is computed as follows: $\text{Time} = 2\min(\text{Time}(\texttt{CovMin}), \text{Time}(\texttt{qCover}))$. We got that combining both tools is 7 times faster than `qCover` and 10 times faster than `CovMin` (where we add 900 seconds for each instance that timed out). This confirms the above statement. We could still get better results by dynamically deciding which ratio of CPU to be shared between the tools depending on some predicted status of the instance.

References

[1] Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. Approaching the coverability problem continuously. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 480–496, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[2] A. Finkel. The minimal coverability graph for petri nets. In *Advances in Petri Nets 1993*, 1993.

[3] Alain Finkel, Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. A counter-example the the minimal coverability tree algorithm. Technical Report 535, Université Libre de Bruxelles, Belgium, 2005.

[4] G. Geeraerts, A. Heußner, M. Praveen, and J.-F. Raskin. ω-Petri nets: algorithms and complexity. *Fundamenta Informaticae*, 137(1):29–60, 2015.

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

[6] P.-A. Reynier and F. Servais. Minimal coverability set for Petri nets: Karp and Miller algorithm with pruning. *Fundamenta Informaticae*, 122(1–2):1–30, 2013.

[7] A. Valmari and H. Hansen. Old and new algorithms for minimal coverability sets. *Fundamenta Informaticae*, 131(1):1–25, 2014.