On Optimizing the Sweep-Line Method

Robert Prüfer

Humboldt-Universität zu Berlin, Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany
pruefer@informatik.hu-berlin.de

Abstract. For applying the *sweep-line method*, a state space reduction technique, it is necessary to provide an automatic calculation of a *progress measure*. In [1], such a calculation has been proposed for Petri nets. The approach presented there may lead to suboptimal results, therefore the author suggested possible optimizations. In this paper, we check whether the proposed optimization goals indeed lead to an improved performance of the sweep-line method.

1 Introduction

The *sweep-line method* is a state space reduction technique that can be used to verify safety properties (e.g. the existence of deadlocks or reachability of a given state) of the modelled system. The main idea of this technique is that during state space exploration, there is some kind of "progress": States that have been processed and will never be visited again are no longer needed for further exploration. To quantify this progress, a *progress measure* is needed that assigns a *progress value* to every state. To use the sweep-line method fully automatically, the progress measure must be calculated automatically, too. Such an automatic calculation of the progress measure for applying the sweep-line method to state spaces of Petri nets has been proposed in [1]. This approach leaves degrees of freedom and, in some cases, calculates a progress measure that does not lead to an optimal state space reduction. In [1], the author suggests two optimization goals to improve the performance of the sweep-line method: minimizing the number of so-called *regress transitions* and avoiding chains of such regress transitions. In this publication, we develop methods to achieve the proposed optimization goals and apply our modified calculation of the progress measure in an experiment to check whether it leads to an improved state space reduction.

2 Definitions

We write $N = [P,T,F,W,s_0]$ for a place/transition Petri net with arc weights, where P,T and F denote places, transitions and arcs. $W : ((P \times T) \cup (T \times P)) \to \mathbb{N}$ denotes the arc weights, where $W(x,y) = 0$ iff $(x,y) \notin F$. s_0 is the initial marking of the net. For every $t \in T$, we define the vector Δt by $\Delta t(p) = W(t,p) - W(p,t)$ for all $p \in P$. For $X = P \cup T$ and every $x \in X$, we define ${}^\bullet x = \{y \in X \mid (y,x) \in F\}$ and $x^\bullet = \{y \in X \mid (x,y) \in F\}$. A place $p \in P$ is a *shared place* iff $|p^\bullet| \geq 2$. We write $s \xrightarrow{t} s'$ for an edge of the state space of N, where firing t in state s leads to the *target state* s', which is the *successor of* s.

A *directed graph* is written as $G = [V,E]$, where V is the set of vertices and $E \subseteq (V \times V)$ is the set of directed edges.

3 The Sweep-Line Method

There exist two forms of the sweep-line method: the basic form [2], which is only suitable for cycle-free state spaces, and the generalized form [3], which can be applied to all types of state spaces. We first describe the basic method and afterwards extend it to the generalized method.

As we stated above, the *progress measure* assigns a progress value $p(s)$ to every state s of the state space. For the basic sweep-line method, the progress measure is monotonous, i.e., for all states s, s' and transitions t, $s \xrightarrow{t} s'$ implies $p(s) \leq p(s')$. The state space is explored such that states with low progress values are visited first. Therefore, we can divide the set of states into three classes: (1) states that have been explored including all their successors, (2) states that have been explored, but which's successors have not been completely explored (we also call this class *front*) and (3) states that have not been explored yet. As the progress measure is monotonous, all states of class (1) can be deleted; only the states of the front must be stored.

For the generalized sweep-line method, the progress measure does not have to be monotonous, i.e., for an edge $s \xrightarrow{t} s'$ of the state space it may hold $p(s) > p(s')$. Such edges are called *regress edges*. Applying the basic method would now lead to the following problem: Let $s \xrightarrow{t} s'$ be a regress edge with s in class (2) or (3) and s' in class (1). Because all states of class (1) have been deleted, we have no information that we have explored s' yet. The exploration would be continued from this state and s' will eventually be explored again as an unknown state – the method would not terminate. That is why in the generalized sweep-line method target states of regress edges are marked as *persistent* and stored permanently in memory. In the current iteration (*sweep*) of the sweep-line method, successors of persistent states are not explored, and newly marked persistent states are initial states of the next sweep. Therefore, it is guaranteed that the method terminates and explores all states at least once.

It is obvious that a large number of persistent states increases memory consumption and can lead to a large number of sweeps.

The sweep-line method can be combined with stubborn sets, another state space reduction method; the computation method for stubborn sets described in [4] can be used for this purpose.

4 Calculation of Progress Values for Petri nets

The progress measure required for the sweep-line method needs to be generated for each system which's state space should be explored. The automatic calculation of a progress measure for Petri nets described in [1] is divided in two phases. Before state space exploration is started, an *offset function* o that maps every transition of a given Petri Net into an *offset value* $o(t) \in \mathbb{Q}$ is calculated. During state space exploration, offset values are combined to progress values: For the initial state s_0, set $p(s_0) = 0$. Then, for every edge $s \xrightarrow{t} s'$ of the state space, set $p(s') = p(s) + o(t)$. For each transition t with $o(t) < 0$, every edge $s \xrightarrow{t} s'$ of the state space is a regress edge. Therefore, such transitions are called *regress transitions* (*RTs*).

Using this approach for calculating a consistent progress measure, offset values must preserve the linear dependencies of all vectors Δt.

Definition 1 (Linear Dependency Preserving Offset Function). *Let N be a Petri net with transitions T. An offset function o is* linear dependency preserving *if for all $t \in T$, $\{t_0,\ldots,t_n\} \subseteq T \setminus \{t\}$, $\lambda_0,\ldots,\lambda_n \in \mathbb{Q}$ with $\Delta t = \lambda_0 \Delta t_0 + \cdots + \lambda_n \Delta t_n$ it holds $o(t) = \lambda_0 o(t_0) + \cdots + \lambda_n o(t_n)$.*

The method for computing offset values suggested in [1] leaves degrees of freedom and, in some cases, calculates negative offset values for more transitions than necessary. In other words, the calculation of offset values can be optimized. For the purpose of optimization, we propose a computation method for offset values that differs from the one described in [1]. Let $N = [P,T,F,W,s_0]$ be a Petri net and let $a \in \mathbb{Q}^{|P|}$ be chosen arbitrarily but fixed. Then the offset value of any transition $t \in T$ can be computed by $o(t) = a \cdot \Delta t$. As offset values computed this way are linear dependency preserving (see Def. 1), a consistent progress measure will be generated when they are used.

5 Optimized Calculation of Offset Values

The computation method for offset values described in Sect. 4 is suitable for the purpose of optimization, as for attaining any optimization goal, only one vector $a \in \mathbb{Q}^{|P|}$ must be chosen properly to calculate the offset value $o(t) = a \cdot \Delta t$ for any transition t of a Petri net. As we already mentioned, we want to attain two optimization goals: First, the number of RTs should be minimized (as one could expect that this reduces the number of persistent states and thus the number of sweeps) and second, chains of RTs should be avoided (as such chains could lead to an unneccessary re-exploration of huge parts of the state space). In our approach described in the following, at first we will compute several sets of RTs which are as small as possible, and afterwards choose a set R which promises to avoid chains of RTs the most. Afterwards, we can calculate a vector a such that $o(t) < 0$ iff $t \in R$.

5.1 Minimizing the Number of Regress Transitions

For this subsection, let $N = [P,T,F,W,s_0]$ be a Petri net. Minimizing the number of RTs means that we want to compute as few negative offset values as possible for all transitions $t \in T$. Furthermore, we decide that we do not want to assign the offset value $o(t) = 0$ to any transition $t \in T$. Therefore, with regard to the computation method described in Sect. 4, for every transition $t \in T$ we set up the inequality $a \cdot \Delta t > 0$. Thus, we obtain a system of linear inequalities which we call $\mathscr{L}$.

Obviously, if $\mathscr{L}$ is feasible, we can calculate positive offset values for all transitions $t \in T$. We may choose any vector $a \in \mathbb{Q}^{|P|}$ from the set of feasible solutions described by $\mathscr{L}$ (which can be done by a Linear Programming solver) to obtain offset values without RTs. Of course, in this case we do not have to avoid chains of RTs as there are none.

If $\mathscr{L}$ is not feasible, things become more difficult. To minimize the number of RTs, we are looking for a feasible subsystem of $\mathscr{L}$ containing as many inequalities as possible. This almost equates to solving an instance of the *Maximum Feasible Subsystem*

Problem (MAXFS) [5]; we only have to transform every inequality from $a \cdot \Delta t > 0$ to $a \cdot \Delta t \geq \varepsilon$ with a preferably small $\varepsilon \in \mathbb{Q}$ and $\varepsilon > 0$. As MAXFS is NP-hard [6], we decided to use a heuristic based on Linear Programming, namely Algorithm 1 from [7], to solve our MAXFS instances. As this heuristic only provides a single solution (like all MAXFS heuristics known to us), we modified it to obtain multiple solutions (see [8] for details). Now, for any feasible subsystem $\mathscr{L}'$ we obtained, we can choose an arbitrary vector $a \in \mathbb{Q}^{|P|}$ from the set of feasible solutions described by $\mathscr{L}'$ to compute offset values with a minimal number of RTs.

5.2 Avoiding Chains of Regress Transitions

As we are now able to generate multiple sets of RTs, in this section we want to develop a heuristic that should select the set which avoids chains of RTs the most. In contrast to the minimization of RTs which can be performed prior to state space exploration, chains of RTs are initially detected during state space exploration. As offset values are calculated prior to state space exploration, we have to think about how to optimize offset value calculation to avoid chains of RTs anyway. To attain this optimization goal, we introduce the *enabling graph*.

Definition 2 (Enabling Graph). *Let $N = [P,T,F,W,s_0]$ be a Petri net. $G = [V,E]$ is the* enabling graph *of N, where $V = T$, and $(t,t') \in E$ iff $\exists p \in P : t \in {}^\bullet p \wedge t' \in p^\bullet$.*

If there is an edge from t to t' in the enabling graph, then firing t creates at least one token on a place $p \in {}^\bullet t'$. Therefore, firing t supports enabling t'. In our heuristic for avoiding chains of RTs, we use shortest paths within the enabling graph to decide whether a set of RTs is likely to avoid chains or not. Although this approach has some weaknesses (for example, it does not take account of concurrency of transitions and of the actual marking of the net), we consider it to be appropriate for avoiding chains of RTs prior to state space exploration. For a given Petri Net $N = [P,T,F,W,s_0]$ and its enabling graph $G = [V,E]$, our heuristic for avoiding chains of RTs chooses a set R of RTs that satisfies the criteria described in the following.

1. Shortest paths from each t to each t' with $t,t' \in R$ should be as long as possible.
2. Shortest paths from each transition t enabled in s_0 to each $t' \in R$ should be as long as possible.
3. For each $t \in R$ and each $p \in {}^\bullet t$, it should hold $p \notin {}^\bullet t'$ for as many $t' \in R \setminus \{t\}$ and $p \in {}^\bullet t''$ for as many $t'' \in T \setminus R$ with $\Delta t''(p) < 0$ as possible.

Criterion 1 aims at maximizing the number of explored states between any two regress transitions. Especially, t' should not fire immediately after t has fired. Furthermore, in our implementation used for the case study in Sect. 6, we demand that the shortest path among all shortest paths between RTs should be preferably long. It is obvious that this criterion helps us to avoid chains of RTs.

Criterion 2 states that during the first sweep, as many transitions as possible should fire before a regress edge is explored. This does not avoid chains of RTs immediately, but it has the effect that a large part of the state space should be already explored in the first sweep.

Table 1. Experimental results. The entries show for how many nets LoLA_H provides better results (+*x*) and worse results (-*x*) compared to LoLA_O. *complete* refers to successful exploration of the complete state space, *#RT* to the number of RTs, *peak* to the peak number of states during state space exploration, *persistent* to the number of persistent states and *sweeps* to the number of sweeps.

	complete		*#RT*		*peak*		*persistent*		*sweeps*	
-stub	+2	-0	+24	-0	+11	-10	+11	-7	+7	-8
+stub	+0	-0	+24	-0	+16	-7	+17	-7	+12	-9

Criterion 3 takes account of shared places. We want to avoid that a RT t shares a place p with another RT, because if the marking of p is sufficiently big, then these two transitions could fire one after another, which would exactly yield a chain of RTs. On the other hand, if t shares a place with a transition t' that is not a RT and t' consumes more tokens from p than it produces on this place, the firing of t' may disable t. Therefore, such transitions should preferably be chosen as RTs.

In our heuristic, we weight these criteria and combine them to a function f that assigns a value to every set of RTs (see [8] for details). The set to which f assigns the largest value is chosen as the set that should avoid chains of RTs the most.

6 Case Study

In our case study, we checked whether the computation of progress values given in [1] really leads to a worse performance of the sweep-line method compared to the computation method and heuristics described in this paper. For this purpose, we tried to compute the state space of 32 Petri nets from a GALS project [9]. We used two versions of the model checking tool LoLA [10]: the original version (*LoLA_O*) which implements the calculation described in [1] as well as a modified version (*LoLA_H*) where we implemented our offset calculation method and heuristics. We executed the state space computation once with stubborn sets (+*stub*) and once without stubborn sets *(-stub)*.

Table 1 shows a summary of our test results. The state space was too large to be computed completely by LoLA_O for 6 nets with stubborn sets and for 10 nets without stubborn sets. In the table, these nets are not considered except for the columns *complete* and *#RT*.

From the results, we conclude that our optimization goals do not lead to an improved performance of the sweep-line method in general. Moreover, as stated in [8], none of the assumed correlations mentioned above (*#RT* and *persistent*, *#RT* and *sweeps*) is verified by the results. Using -*stub*, for 14 nets where the full state space could be computed, *#RT* was reduced, but *persistent* was only decreased for 8 of them and *sweeps* for 5 of them. For +*stub* and -*stub*, for some nets *#RT* was reduced significantly, but *peak*, *persistent* and *sweeps* even got larger, what means that the values for memory consumption and the number of iterations got worse. Changing the weights of the criteria for avoiding chains mentioned in Sect. 5.2 does not change the results in general.

As the computation of the offset values does not depend on stubborn sets, we obtain the same number of regress transitons for state space exploration with and without stubborn sets.

Although the results for the combination of the sweep-line method and stubborn sets look more promising than those obtained without stubborn sets, note that our optimization goals did not include special improvements for stubborn sets. Especially, computation of stubborn sets combined with the sweep-line method in LoLA depends on the offset values of all transitions; thus, one should not reason about these results without taking a closer look at LoLA.

7 Conclusion and Further Work

The case study in Sect. 6 revealed that the optimization goals examined in Sect. 5 do not improve the performance of the sweep-line method in general. Different optimization goals must be found to achieve this aim.

For estimating the number of persistent states, it turns out that it would be important to know how often RTs are enabled during state space exploration and how many distinct target states they lead to. Furthermore, even offset values of transitions which are no RTs influence the *peak* value. It would be appropriate to provide an offset value calculation that exploits this fact. Finally, as mentioned above, when using LoLA, the implementation of stubborn sets should be taken into consideration when trying to optimize the sweep-line method.

References

1. Schmidt, K.: Automated generation of a progress measure for the sweep-line method. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 192-204. Springer, Berlin Heidelberg (2004)
2. Christensen, S., Kristensen, L.M., Mailund, T.: A Sweep-Line Method for State Space Exploration. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 450–464. Springer Berlin Heidelberg (2001)
3. Kristensen, L.M., Mailund, T.: A Generalised Sweep-Line Method for Safety Properties. In: Eriksson, L.-H., Lindsay, P.A. (eds.) FME 2002. LNCS 2391, pp. 215–239. Springer Berlin Heidelberg (2002)
4. Schmidt, K.: Stubborn Sets for Standard Properties. In: Donatelli, S., Kleijn, H.C.M. (eds.) ICATPN'99. LNCS, vol. 1639, pp. 46–65. Springer, Berlin Heidelberg (1999)
5. Amaldi, E., Pfetsch, M.E., Trotter, L.E. Jr.: On the maximum feasible subsystem problem, IISs and IIS-hypergraphs. Math. Program. 95 no. 3, pp. 533-554 (2003)
6. Amaldi, E., Kann, V.: The Complexity and Approximability of Finding Maximum Feasible Subsystems of Linear Relations. Theor. Comput. Sci. 147 no. 1-2, pp. 181-210 (1995)
7. Chinneck, J.W.: Fast Heuristics for the Maximum Feasible Subsystem Problem. INFORMS J. Comput. 13 no. 3, pp. 210–223 (2001)
8. Prüfer, R.: Optimierung der Sweep-Line-Methode. Humboldt-Universität zu Berlin, diploma thesis (2010)
9. Stahl, C., Reisig, W., Krstic, M.: Hazard Detection in a GALS Wrapper: A Case Study. In: Desel, J., Watanabe, Y. (eds.) Proceedings of the Fifth International Conference on Application of Concurrency to System Design (ACSD05), pp. 234-243. IEEE Computer Society (2005)
10. Schmidt, K.: LoLA: A Low Level Analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 465–474. Springer, Berlin Heidelberg (2000)