Deadlock Analysis of Petri Nets: Minimization of Memory Amount

Andrei Karatkevich, Marian Adamski

Dept. of Computer Eng. and Electronics, Technical University of Zielona Gora
ul. Podgorna 50, 65-246 Zielona Gora, Poland
E-mail: <A.Karatkevich, M.Adamski>@iie.pz.zgora.pl

Abstract. The task of finding all the deadlocks in a Petri net is considered. Modified approach of state space construction is suggested. We propose a way to minimize the amount of memory needed for such analysis by removing from the reachability graph some nodes, which are not necessary for further analysis. Such minimization is important because of the state explosion problem that arises during such analysis. The approach may be applicable to other problems of Petri net analysis.

I. Introduction

Petri nets [1,2] are a popular model of concurrent digital systems used in computer-aided design. Various methods of analysis of Petri nets, are known, but the most universal approach, theoretically applicable to all kinds of bounded Petri nets, is the constructing of state space. But because of the state explosion problem this approach in its classical variant cannot be applied to big nets. That’s why the methods of dynamic analysis are important, which allow avoiding constructing of complete state spaces [3,10,11]. Usually they construct reduced state space, which is however enough for checking the important properties.

When constructing a state space, all the considered states are usually kept in memory till the end of analysis, although this is not always necessary. Removing from memory some of the considered states allows reducing the amount of memory needed for analysis.

II. Necessary Definitions

We omit the known definitions because of lacking space; all the necessary definitions can be found in [1,2,4] (Petri nets), [5,6] (stubborn set method) and [8] (graphs). Note that we use the next definition of deadlock: a deadlock is a marking (reachable from the initial one) such that no transition is enabled in it [4].

III. Finding the Deadlocks of a Net

Let’s consider the next task: a bounded Petri net with its initial marking is given, and it is necessary to find all its deadlocks or to prove that there are none. Constructing the complete reachability graph or of a reduced reachability graph by stubborn set method [6] can solve that task. But both approaches require keeping in memory all the visited markings. Why a marking that is not dead should be kept in memory, when all the markings directly reachable from it are investigated? They cannot be dropped, because otherwise the algorithm could loop forever. But is it necessary to keep all of them? Let’s consider the next algorithm.

Algorithm 1

Let $\Sigma = (P,T,F)$ be a Petri net, let $G(V,E)$ be its reachability graph to be constructed. Initially $V = \{v_0\}$, $v_0$ corresponds to $M_0$; $E = \emptyset$. $Fr = V$. 
1. While \( Fr \neq \emptyset \ (x \in Fr) \), repeat the following. If there are enabled transitions in \( M \) corresponding to \( x \),

1.1. For all the transitions that are enabled in \( M \) (variant: for all the enabled transitions belonging to one of stubborn sets\(^1\)) get the new markings; create for them corresponding nodes; add them to \( V \) and \( Fr \).

1.2. Add to \( E \) arcs leading from \( v_0 \) to all the nodes corresponding to markings obtained at the previous step.

1.3. If \( x \neq v_0 \), then \( V := V \setminus \{x\} \), remove from \( E \) all the arcs incident to \( x \).

1.4. \( Fr := Fr \setminus \{x\} \).

If there are no enabled transitions, then \( Fr := Fr \setminus \{x\} \), and \( M \) is a deadlock.

2. The end.

**Affirmation 1.** Algorithm 1 will find all the deadlocks and will stop, if no different reachable markings are mutually reachable from each other. Else the algorithm may loop forever.

(Proofs are omitted because of limited space.)

So the Algorithm 1 has to be modified to prevent looping. That means that *some* information on the visited states has to be kept in memory. And as far as looping is the problem, it would be enough to keep in the graph at least one node for each cycle. So the main idea is the following: select transitions in each cycle of net graph and keep in memory all the markings obtained by firing those transitions, and only them. The modified algorithm is described below.

**Algorithm 2**

Let \( \Sigma = (P,T,F) \) be a Petri net, let \( G(V,E) \) be its reachability graph to be constructed. Initially \( V = \{v_0\} \), \( v_0 \) corresponds to \( M_0 \); \( E = \emptyset \). \( Fr = V \); \( D = \emptyset \).

1. Find in the graph of the net such a set \( Q \) of transitions that each cycle in that graph comes through at least one transition in that set.

2. While \( Fr \neq \emptyset \ (x \in Fr) \), repeat the following. If there are enabled transitions in \( M \) corresponding to \( x \),

2.1. For all the transitions that are enabled in \( M \) (variant: for all the enabled transitions belonging to one of stubborn sets) get the new markings by their firing. Create corresponding nodes to them, if \( V \) doesn’t contain such nodes; add them to \( V \) and \( Fr \); add to \( E \) all the arcs from \( x \) to those nodes. If a newly created marking is obtained by firing of a transition \( Qt \in Q \), then add the corresponding node to \( D \).

2.2. If \( x \not\in D \), then:

2.2.1. For all the pairs of arcs \( (a, x) \) and \( (x, b) \): \( E := (E \cup \{(a,b)\}) \setminus \{(a,x),(x,b)\} \).

2.2.2. \( V := V \setminus \{x\} \).

2.3. \( Fr := Fr \setminus \{x\} \).

If there are no enabled transitions, then \( Fr := Fr \setminus \{x\} \), and \( M \) is a deadlock.

3. The end.

**Affirmation 2.** Algorithm 2 will find all the deadlocks (if there are any) and will stop for any bounded Petri net.

---

\(^1\)This approach guarantees finding all the deadlocks; see [6].
IV. "Decyclization" of Petri nets

The first item of Algorithm 2 may be a non-trivial task, especially if we attempt to minimize the number of transitions selected in it. It can be considered as a task of decyclization of an oriented graph. Some methods of solving the task of decyclization for oriented graphs are known [7], but they are not effective for large graphs. So we had to develop an approximate algorithm.

**Algorithm 3 Decyclization of oriented graph**

Let \( G=(V, E) \) be the given graph. \( G'=(V', E') \) such that \( V'=V, E'={} \). \( S={} \).

1. While \( \exists v \in V : (id(v) = 0 \lor od(v) - 0) \)
   1.1. \( E := E \setminus \{ e \in E : v \in e \} \).
2. Calculate for every arc \( e \in E \) its weight:
   \[ w(e) = id(\text{init}(e)) - od(\text{init}(e)) + od(\text{ter}(e)) - id(\text{ter}(e)). \]
3. While \( E \neq {} \)
   3.1. \( e := \min(\{w(e) : e \in (E \setminus S)\}); \)
   3.2. \( S := S \cup \{e\}; \)
   3.3. \( E' := E \cup \{e\}; \)
   3.4. If \( G' \) has a cycle, then \( E' := E' \setminus \{e\} \).
4. \( (E \setminus E') \) is a set of arcs, removing of which is enough to destroy all circuits in \( G \). The end.

Detailed explanation of this algorithm and its background can be found in [9]. For our purpose it is enough to apply it to the Petri net graph, and then to create the set of transitions needed in Algorithm 2 item 1:

\[ Q = \{ t \in T \mid ((t,x) \in F) \land ((t,x) \in (E \setminus E')) \lor ((t,x) \in F) \land ((t,x) \in (E \setminus E')) \}. \]

V. Complexity Evaluation

Let’s try to evaluate the economy of the amount of memory that can be obtained by the algorithm. For that purpose we have to compare the size of a reachability graph created by known methods and the size of reachability graph without intermediate nodes, created by Algorithm 2. Let \( G \) be a reachability graph, \( G' \) - graph for the same net created by Algorithm 2. \( V' \subseteq V \), by construction. It’s needed to evaluate relation between \( |V'| \) and \( |V| \).

In the worst case every transition of the Petri net belongs to at least one cycle of the net graph. Usually the cycles are intersecting, and that’s why Algorithm 3 is needed. In such case, \( |Q| < n \), where \( n \) is the number of cycles in the net graph. But suppose that the cycles do not intersect \( (Q = n) \). Let \( k \) be the average number of transition firing during the reachability graph constructing, \( l \) - the average number of incoming arcs to a node in \( G \), \( m \) – the average number of transitions in a cycle of the net graph. Then

\[ |V'| = |V| \cdot \frac{knl}{m|E|} + q + 1, \]

where \( q \) is the number of deadlocks. The number of terminal nodes is usually much less than number of intermediate nodes, so we can write

\[ \frac{|V|}{|V'|} = \frac{nl|E|}{knl}. \]

But according to our supposing, \( |E| = kmn \). Hence

\[ \frac{|V|}{|V'|} = \frac{m^2}{l}. \]

To show some numbers, let’s suppose that the net has additional transitions with such a configuration that it is impossible to obtain a considerable gain by using the stubborn set method. Then we can roughly suppose that \( l = n \), and \( |V'| = m^n \). If \( m=n=8 \), \( |V'| = 8^8 = 1.7 \cdot 10^7 \). If keeping in memory each node of the graph with the corresponding marking requires 16 bytes, the whole amount
of memory needed will be about 270 MB, which may be too much. By using the suggested method, the graph will be built with the number of nodes $|V|^7$, and the amount of memory will be only 34 MB.

Now let’s evaluate time complexity. A marking already visited may be considered again if the corresponding node was deleted from the reachability graph. Again, let’s consider the worst case. In such case a cycle in reachability graph has one corresponding cycle in Petri net graph, and it can be shown that their lengths are the same (the length of a cycle in Petri net graph is understood as the number of transitions in it). The cycle in a reachability graph can be entered at most at each node it is coming through, or $m$ times. And then all the markings corresponding to the nodes that belong to the cycle can be considered again, until the algorithm reaches the node marked as “don’t delete”. So each marking can be analysed in average $m^2/2$ times. So, we gain in memory but lose in time.

VI. Conclusion

The suggested approach allows reducing the amount of memory used for analysis of Petri nets, if the analysis is performed by constructing a reachability graph. This reduction is possible at the cost of analysis time, so it makes sense to use if the amount of memory is a more critical parameter than the time. The approach is applicable for the methods constructing reduced reachability graphs; in the paper the stubborn set method is considered, but the approach can be used for other methods of that kind, for example maximal concurrent simulation [10]. In the paper the deadlock analysis is considered, but the idea can be used for analysis of other behavioural properties of Petri nets. Using of the approach for more general analysis will be the topic of further work.

References