How to Distribute LTL Model-Checking Using Decomposition of Negative Claim Automaton[⋆]

Jiří Barnat

Faculty of Informatics, Masaryk University Brno,
Botanická 68a, 602 00 Brno, Czech Republic
`barnat@fi.muni.cz`

Abstract. We propose a distributed algorithm for model-checking LTL formulas that works on a network of workstations and effectively uses the decomposition of the formula automaton to strongly connected components to achieve more efficient distribution of the verification problem. In particular, we explore the possibility of performing a distributed nested depth-first search algorithm.

1 Introduction

The verification of complex concurrent systems requires techniques to avoid the state-explosion problem. Several methods to overcome this barrier have been proposed and successfully implemented in automatic verification tools. One of the relatively successful recently used method is a distribution of required computational resources among several LAN interconnected computers.

The main issue in distributing *explicit state* model-checking algorithms is the way how to *partition* the state space among the individual computers (*network nodes*) and how to modify sequential algorithms to work correctly and effectively in the distributed environment.

Some of the known state space partition techniques exploit certain characteristics of the system, and hence work well for systems possessing these characteristics, but fail to work well for systems which do not have them. One of the very successful general technique to cope with situations where we are unable to compute some characteristic in advance is the randomization. Probabilistic techniques to partition the state space have been used e.g. in [6, 8, 1] with surprisingly good results.

Recently, there have been proposed two approaches that use additional information about the state space to make the verification process more efficient. In [4] the authors have exploited the particular structure of the verified property to get shorter counter-examples while in [2] the authors have used similar method to distribute the verification algorithm for alternation-free mu-calculus.

In this paper we report on one particular way how to obtain some information about the structure of the state space from the verified property and how to use it for the distribution of explicit state LTL model-checking.

[⋆] This work has been partially supported by the Grant Agency of Czech Republic grant No. 201/00/1023 and by Ministry of Education grant FRVŠ No. 598/G4/2002

2 Distributed Nested DFS algorithm

Our aim is to explore a technique that allows effective partition of the state space and thus allows a distribution of the on-the-fly automata-based model-checking of LTL properties. The automata-based approach transforms the verification problem to the Büchi automaton emptiness problem. This can be further transferred to a problem of detecting a reachable cycle with an accepting state in the underlying graph of the automaton. The cycle detection is done by the nested depth-first search algorithm. As the depth-first search is a P-complete problem, promising efficient parallel DFS-based algorithm that would work on general systems is unlikely to exist [7]. The practical effectiveness of such an algorithm depends very much on the appropriate partitioning of the graph among the participating network nodes. Ideally, if the partition respects the decomposition into strongly connected components in such a way that no cycle crosses network nodes boundaries then on each network node the search for accepting cycles can proceed independently and in parallel with computations on other network nodes. The question is whether any partition satisfying the above mentioned criterion can be efficiently found and whether the partition is well-balanced among the network (each node "owns" approximately the same number of states).

In the case of automata-based LTL model-checking so called *product* automaton is built. The product automaton is a result of a synchronization of the small *negative claim automaton* with a huge *system automaton* (with all states considered as accepting). The system automaton models the behavior of the given system and the negative claim automaton describes the behavior which contradicts the verified property. If the language of the product automaton is not empty then the system has some illegal behavior which means that verified property is not satisfied.

Definition 1. *Let* $A = (\Sigma, S_A, q_A, \delta_A, F_A)$ *and* $B = (\Sigma, S_B, q_B, \delta_B, F_B)$ *be Büchi automata. The* synchronous product $A \otimes B$ *of* A *and* B *is the automaton* $(\Sigma, S, q, \delta, F)$*, where* $S = S_A \times S_B, q = (q_A, q_B), F = F_A \times F_B$ *and* $(u', v') \in \delta((u, v), a)$ *if and only if* $u' \in \delta_A(u, a)$ *and* $v' \in \delta_B(v, a)$*. The automata* A *and* B *are called* projections.

Our aim is to distribute the standard (sequential) nested DFS algorithm on a network of workstations that communicate via message-passing. The algorithm was chosen not only because of its efficiency, but also because a distribution of this algorithm seems to be a natural extension of commonly used verification tools such as SPIN [5].

A safe solution is to decompose the graph into strongly connected components first and then to partition the graph according to this decomposition. In addition, the decomposition can make the nested (second) search even more efficient by searching only those paths that can really form a cycle in the graph (i.e., the paths that belong to one maximal strongly connected component). However, decomposing the system in advance would actually solve the verification problem.

In contrast to the product automaton it is possible to effectively decompose the negative claim automaton into all its maximal strongly connected components in advance since the size of a negative claim automaton is not overwhelming. Moreover, we can carry over this decomposition to the product automaton and split the product automaton into *sets of maximal strongly connected components* according to the following lemma.

Lemma 1. *For each maximal strongly connected component in B there is a corresponding set of maximal strongly connected components in $A \otimes B$.*

The partition function, which is responsible for the distribution of the state space among the network nodes, checks to which set of maximal strongly connected component the state belongs to. The state is placed on the same network node as the other states of the same set.

Lemma 2. *Let A, B be Büchi automata. If $\mathcal{C}$ is a maximal strongly connected component in $A \otimes B$ then projection $\pi_1(\mathcal{C})$ forms a (not necessarily maximal) strongly connected component of A and projection $\pi_2(\mathcal{C})$ forms a (not necessarily maximal) strongly connected component of B.*

It follows immediately that no cycle in the underlying graph can cross the set boundaries. Hence, the searches revealing the cycles can be performed within the different sets of $A \otimes B$ independently.

Lemma 3. *Let A, B be Büchi automata. If $\mathcal{D}$ in $A \otimes B$ is an accepting cycle then the set $\pi_1(\mathcal{D})$ forms an accepting cycle in A and the set $\pi_2(\mathcal{D})$ forms an accepting cycle in B.*

Moreover, according to Lemma 3 it is meaningful to characterize the types of the sets of maximal strongly connected components of a given graph. The classification of the sets of maximal strongly connected components can be derived from the following classification of maximal strongly connected components:

Type F: (*Fully Accepting*) Any cycle within the component contains at least one accepting state. (There is no non-accepting cycle within the component.)
Type P: (*Partially Accepting*) There is at least one accepting cycle and one non-accepting cycle within the component.
Type N: (*Non-Accepting*) There is no accepting cycle within the component.

In the presented technique the types of the sets of the components are not used to partition of the state space, but enable further optimization of the depth-first search algorithm (as presented in [4]). On the other hand, the information about the types is easily obtained and thus may be fruitfully used in other approaches to the problem of state space partition.

The pseudo-code of the final Distributed Nested DFS algorithm is given in Figure 1. Each node maintains its own local queue of states to be explored and is responsible for its own part of the state space. When a network node computes a new state it checks whether the state belongs to its subset of the state space. If the state is local then the network node continues locally, otherwise a message containing the state is sent to the state's owner.

3 Implementation

We have implemented an experimental version of the Distributed Nested DFS algorithm using the SPIN verifier version 3.4.10 and performed a series of preliminary tests. All the experiments were performed without partial order reductions.

Our algorithm uses a manager process running on the “master” network node to initiate and terminate the distributed computation. The manager process decomposes the negative claim automaton and defines the partition of the state space. In the current implementation we assign the part corresponding to one set of maximal strongly connected components to one network node.

In the distributed computation all the network nodes involved perform the same algorithm for the assigned part of the state space. After the manager process has detected termination it stops the entire computation. The termination detection is done using a virtual ring-based distributed algorithm. The distributed algorithm terminates when all local computations are finished and all communication channels are empty.

The experiments we made indicate that there are some realistic formulas that can help the decomposition. For example the following formula express some kind of fairness in serving floors by an elevator cabin:

$$G((r1) \implies ((\neg p1)U((p1)U((\neg p1)U((p1)U((p1) \wedge o))))))$$

The negative claim of the formula has 8 strongly connected components. And experiment has shown that the maximal memory utilization per node is about 25% of the memory requirenments during the standard sequential computation.

4 Conclusion

We propose a distributed algorithm for LTL model checking that runs on a cluster of PCs. The main novelty of our approach is that we use the decomposition of the negative claim automaton into maximal strongly connected components to distribute the verification problem over the cluster. In addition to the fact that we are able to decompose the task so that several instances of the verification procedure can be performed in parallel, we are also able to perform an improved version of the nested DFS algorithm. Our new approach to the distribution of the algorithm can be used in the framework of multi-thread programming as well. We stress that our technique is compatible with other state space saving techniques and that it is independent of the others partition techniques, hence, it can be fruitfully combined with them.

There are other possible strategies for partitioning the state space. One of them places states belonging to a component of type N randomly on network nodes. This is possible because the only relevant information for these states is their reachability and can be analysed e.g. using the algorithm of Lerda and Sisto[6]. The other parts of the graph which are of type P and F can be distributed using algorithm presented in [1] or [3] or using the simplest baton algorithm.

```
proc $Node(i)$
  if $Part(initstate) = i$
    then $queue[i] := \{initstate\}$
     else $queue[i] := \emptyset$
  fi
  while $Not\ End$ do
        if $queue[i] \neq \emptyset$
          then $state := Head(queue[i])$
               $queue[i] := Tail(queue[i])$
               $in\_stack[i] := \{state\}$
               $DFS(i, state)$
        fi
  od
end


proc $DFS(i, state)$
  if $(state, 0) \notin visited[i]$
    then $visited[i] := visited[i] \cup \{(state, 0)\}$;
          foreach $s \in Succ(state)$ do
            if $Part(s) \neq i$
              then $queue[Part(s)] := (queue[Part(s)], s)$
               else $in\_stack[i] := in\_stack[i] \cup \{s\}$
                    $DFS(i, s)$
                    $in\_stack[i] := in\_stack[i] \setminus \{s\}$
            fi od
          if $Accepting(state) \wedge Part(state)$ is of type $P$
            then $NestedDFS(i, state)$
          fi
     else if $state \in in\_stack[i] \wedge Part(state)$ is of type $F$
            then $Report(\text{"Cycle Found"})$
          fi fi
end


proc $NestedDFS(i, state)$
  if $(state, 1) \notin visited$
    then $visited := visited \cup (state, 1)$
          foreach $s \in Succ(state)$ do
            if $Part(s) = Part(state)$
              then if $state \in in\_stack[i]$
                          then $Report(\text{"Cycle Found"})$
                           else $NestedDFS(i, s)$
                        fi fi
          od fi
end
```

Fig. 1. Distributed Nested DFS Algorithm

In the future we intend to implement and experiment other strategies for distribution of the verification problem that use additional information from the verified property. Also, we would like to continue our search for similar improvements achieved by the structure gained from the modeled system.

The other question is whether it is possible to find a specialized algorithm for Fully and Partially accepting subclasses of the problem. We would like also to explore the possibility and cost of turning all type P components of negative claim automaton into type F components which can lead to simpler algorithm.

Acknowledgment. I would like to thank my supervisor Luboš Brim for his highly appreciated support. I also acknowledge the many comments and remarks made by Ivana Černá that were used to polish the final version of the paper.

References

1. J. Barnat, L. Brim, and J. Stříbrná. Distributed ltl model-checking in SPIN. In Matthew B. Dwyer, editor, *8th International SPIN Workshop*, volume 2057 of *LNCS*, pages 200–216. Springer, 2001.
2. B. Bollig, M. Leucker, and M Weber. Parallel model checking for the alternation free mu-calculus. In T. Margaria and W. Yi, editors, *Proc. TACAS 2001*, volume 2031 of *LNCS*, pages 543–558. Springer, 2001.
3. L. Brim, I. Černá, P. Krčál, and R. Pelánek. Distributed LTL Model Checking Based on Negative Cycle Detection. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *FST TCS 2001*, volume 2245 of *LNCS*, pages 96–107. Springer, 2001.
4. S. Edelkamp, A. Lluch-Lafuente, and S. Leue. Directed model-checking in HSF-SPIN. In Matthew B. Dwyer, editor, *8th International SPIN Workshop*, number 2057 in LNCS, pages 57–79. Springer, 2001.
5. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
6. F. Lerda and R. Sisto. Distributed-Memory Model Checking with SPIN. In D. Dams, R. Gerth, S. Leue, and M. Massink, editors, *Proc. 6th SPIN Workshop on Model Checking of Software (SPIN99)*, volume 1680 of *LNCS*. Springer, 1999.
7. J.H. Reif. Depth-first search is inherrently sequential. *Information Processing Letters*, 20(5):229–234, 1985.
8. U.Stern and D. L. Dill. Parallelizing the murφ verifier. In O. Grumberg, editor, *Proceedings of Computer Aided Verification (CAV '97)*, volume 1254 of *LNCS*, pages 256–267. Springer, 1997.