WDS'07 Proceedings of Contributed Papers, Part I, 52–57, 2007. ISBN 978-80-7378-023-4 © MATFYZPRESS

Heuristics for the State Space Division

T. Poch

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic.

Abstract. All model checkers suffer from the state space explosion. The distribution of the task among several machines is a technique which adresses this problem. The key issue of a distributed checker are the rules for assigning the states to the participating machines. The task of the checker is to visit all states of the system, which usually arise from the composition of smaller systems. This paper presents ideas how to use information obtained from the state spaces of the particular systems to predict suitable division of the composite state space among the participating machines.

Introduction

Growth of the computability power in the last years enabled practical use of model checking of software systems. The developer provides a model of the software and a set of properties that are believed to hold. Then the state space of the model is examined to prove that all properties really hold in all states the system can reach. This proof, or the counter example which is provided if some property does not hold, helps the developer to deliver the software sooner and cheaper.

Unfortunately, the size of the state space is typically exponential to the size of the model description. This fact is referred as 'state space explosion'. The state space explosion is a burning problem that limits usage of the model checking to relatively small tasks.

One of the approaches that significantly decrease state space of the task are Behavior Protocols [1]. This is achieved by using component paradigm for splitting the model into independent parts. A Behavior Protocol is a regular expression that describes the behavior of a software component. The component implementation details are hidden during the checking of the whole application—what is reduced to the checking whether behavior protocols of used components are compliant. However, even the checking of Behavior Protocols compliance faces the exponential growth of number of states. Here, the distribution can help.

As a Behavior Protocol is similar to a regular expression, it can be expressed by a finite state machine—FSM. We are interested in such state spaces that originate from a set of behavior protocols. Such state spaces—composite state spaces (CSS)—resemble the cartesian product of the original protocols' FSMs—primitive state spaces (PSS). In the following text, the states (resp. edges) of the Composite State Space are refered to as composite states (resp. edges). If there is no communication between components, the CSS is even equal to the cartesian product of PSSs. Having a system consisting of n components, a state is uniquely identified by an n-component vector. Each vector component identifies a state of one protocol. The entire state space can be thus enclosed in an n-dimensional cuboid.

One of the key issues of a distributed checker is the set of rules for assigning the states to the participating machines. We can formulate this issue as assigning parts of the n-dimensional cuboid to the available machines. Other approaches are described in [2–5]. There are two requirements for the division.

Requirement 1 *Number of composite states assigned to each machine must be similar.*

Requirement 2 *Number of composite edges crossing a machine boundary must be minimal.*

The task of splitting the cuboid can be split into two independent problems. First, split the cuboid into areas, and second, assign the areas to distinct machines. In the following text, the

first problem is discussed. Our goal is to split the cuboid into a given number of areas such that the number of composite states belonging to each area is similar and the number of composite edges crossing the area boundary is minimal. The second problem is elaborated in [6].

Related Work

All model checkers suffers from the state space explosion and the task distribution can help. The distributed version of the explicit Murφ verifier was presented in [3]. In this work, random assignment of particular states to available machines is used. Although the communication criteria is utterly omitted good results are presented. It is caused mainly by the fact that computation of neighbour states in Murφ is very demanding. In this work, it is proved that the random assignment provides uniform distribution of workload if the number of states is much higher than the number of machines.

In [2], a distributed version of the SPIN model checker is described. Its main goal is to employ all memory of the available computers, not necesarily to have the result sooner. As the enumeration of state neighbours is quite fast in SPIN, the completely random assignment does not give good results. The authors propose another assignment. The SPIN state identifier is a vector. Each component identifies a state of one process. The assignment function uses only one component of the vector. The machine boundary is crossed only when the state of one particular process is changed. But there is typically many processes in the system and each transition means a change of only one or two of them. This way, lower communication requirements are achieved.

Division heuristics

The division of the state space must be done before the traversal begins. Unfortunately, in order to divide the state space optimally with respect to the requirements, we must know the shape of the state space precisely. As this information is not available before the traversal is finished we must rely on heuristics.

Let A be an architecture. Its behavior is described by composition of Primitive State Spaces $PSS_1, \ldots, PSS_n$. Let $Cube_A$ is the n-dimensional cuboid containing all composite states representing the composition of PSSs. The size of the cuboid in the dimension i corresponds to the number of primitive states in PSS_i. One can imagine the primitive states sorted along the cuboid edges using the DFS ordering. As the composite state corresponds to a combination of primitive states, we can easily map the composite states to the space within the cuboid.

Let us suppose that we want to split $Cube_A$ along the axis k into p parts. The PSS corresponding to the axis k has $States_{PSS_k}$ states. It means that we are looking for splitting points $s_0, s_1, \ldots, s_p$ such that $\forall i \in \{0, \ldots, p-1\} : s_i < s_{i+1}$, $s_0 = 0$ and $s_p = States_{PSS_k}$.

Heuristics aiming at Requirement 1

Heuristic 1 *Divide the cuboid into areas of the same size.*

This heuristic is a naive one. It is based on the fact that the volume of an area is very coarse upper bound of the number of reachable states within the area. This splitting uses a little information from the PSS—just the number of states—and respects only the requirement for the similar number of reachable states assigned to the machines. It utterly omits the requirement of minimal number of edges crossing machine boundaries. On the other side, the best splitting from the point of view of minimal number of crossing edges is no splitting. We need to find a balance between these two requirements.

If we want to numerically rate how much a particular division along one axis comply this heuristic, we can use variance of splitting point distances σ^2.

$$\sigma^2 = \frac{\sum_{i=0}^{p-1}(s_{i+1} - s_i - \frac{States_{PSS_k}}{p})^2}{p} \quad (1)$$

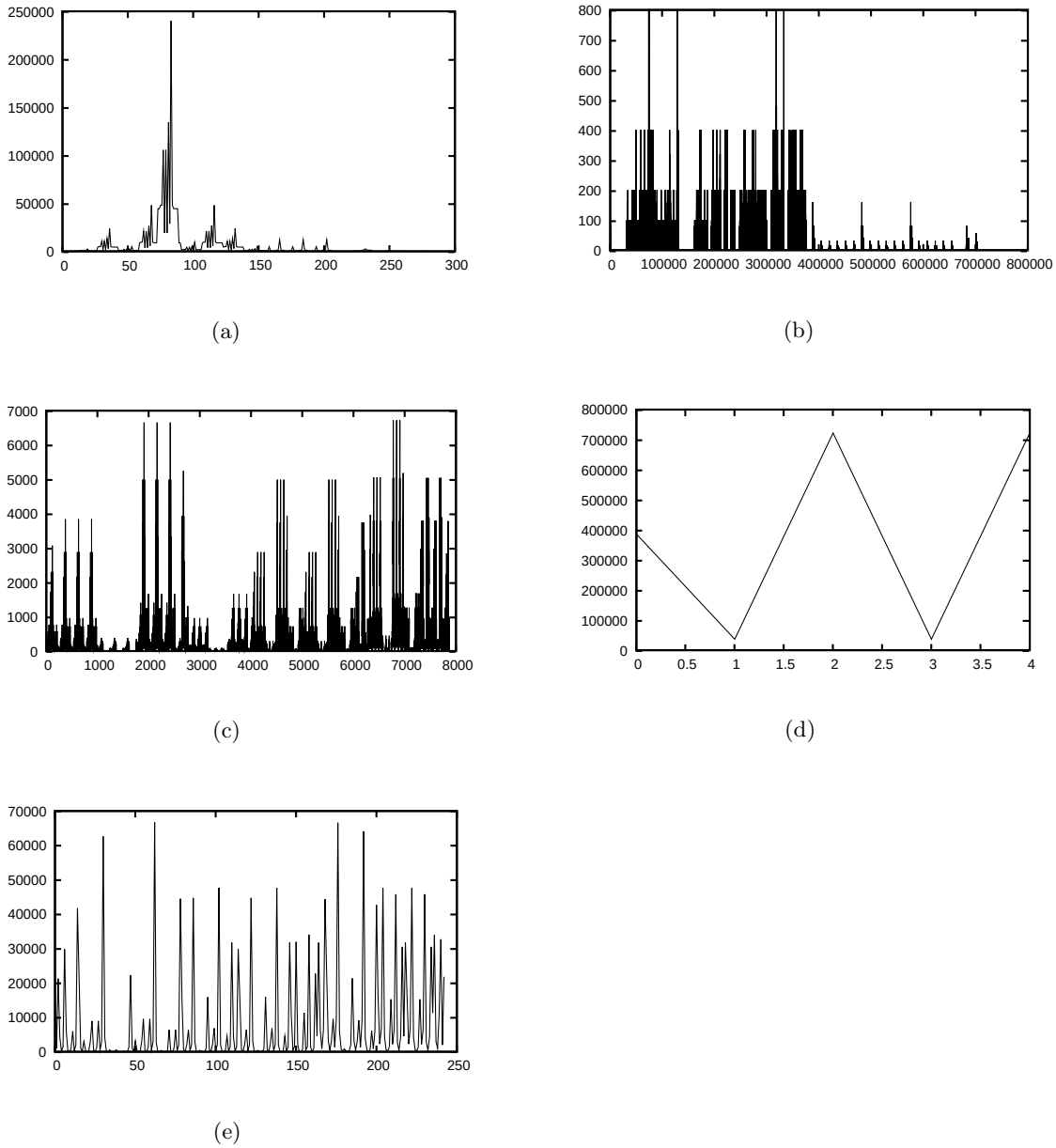


Figure 1. These graphs capture how many times a particular state of the PSS is involved in a composite state. Having the particular splitting, which divides the x-axis into a number of parts, similar pictures can be easily used to evaluate the splitting. If the splitting is ideal from the point of view of the Requirement 1, the integral over the different areas is equal.

To provide a better division, we must use more information from the PSS than the number of states.

The next two heuristics in this section rely on the fact that the states of the primitive state spaces are not equally important. As each composite state correspond to the combination of states from each primitive state space, some primitive states participate in many composite states, some primitive states participate in a few and some in none. This fact can be expressed by assigning the weight to each state. Let us denote $w_i^{PSS_k}$ the weight of the i-th state from the

primitive state space PSS_k. Then, if we want to split the cuboid with respect to this weight, we want the sum of all states' weights from each area to be similar. If we denote $Weight_w^{PSS_k}$ as the sum of weights of all states from the PSS_k, we want the following variance to be minimal:

$$\sigma^2{}_w = \frac{\sum_{i=0}^{p-1}(\sum_{j=s_i}^{s_{i+1}-1}(w_j^{PSS_k}) - \frac{Weight_w^{PSS_k}}{p})^2}{p} \tag{2}$$

This is variance with respect to the weight w.

The following heuristics suggest different weight functions:

Heuristic 2 *Weight a state by the number of input edges.*

Heuristic 3 *Weight a state by the number or cycles it participates in.*

Heuristics aiming at Requirement 2

Let us denote $CE_k(l)$ as the number of composite edges crossing the boundary between two areas obtained by splitting $Cube_A$ at the number l on the axis k. $PE_k(l)$ is the number of all edges E in PSS_k such that $E =< Id_a, Id_b >$ where Id_a and Id_b are state identifiers of directly connected states and $Id_a \leq l < Id_b$:

$$\forall k, l : CE_k(l) \leq PE_k(l) * \prod_{i=0, i\neq k}^{n} States_{PSS_i} \tag{3}$$

In the following text, the right part of the equation is referred to as *crossing edges upper bound* $CE_k^{upBound}(l)$:

$$CE_k^{upBound}(l) = PE_k(l) * \prod_{i=0, i\neq k}^{n} States_{PSS_i} \tag{4}$$

Heuristic 4 *Divide the cube in such a way that the crossing edges upper bound is minimal.*

We want to choose k and l such that $CE_{P_k}^{upBound}(l)$ is minimal. However, we do not get minimal $CE_{P_k}(l)$ necessarily this way. The algorithm for computation of $PE_k(l)$ can be found in [6].

This way, we can split a cuboid with respect to the number of edges crossing machine boundaries. However, the volume assigned to particular machines can differ. The ballance between Heuristic 1, which aims the Requirement 1, and Heuristic 4, which aims the Requirement 2, can be found by minimalization of the following expression:

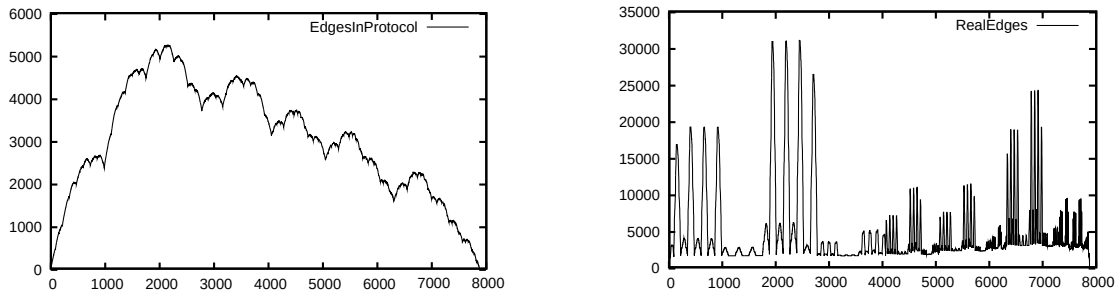
$$Unsuitability = C_{edges} * \sum_{i=1}^{p-1} CE_k^{upBound}(s_i) + \sigma^2 \tag{5}$$

Where C_{edges} is a constant expressing how long takes transmitting of one edge in comparison to the time needed to check one state. We are looking for $s_0, s_1, \ldots, s_p$ such that $Unsuitability$ is minimal.

It is important to mention that all of this is just a heuristic. By choosing a division that has rather small unsuitability, we avoid 'obviously bad' divisions. Moreover, this heuristic depends on the estimation $CE_k^{upBound}(l)$ of the number of edges crossing areas. The experiments shows that this upper bound is too coarse.

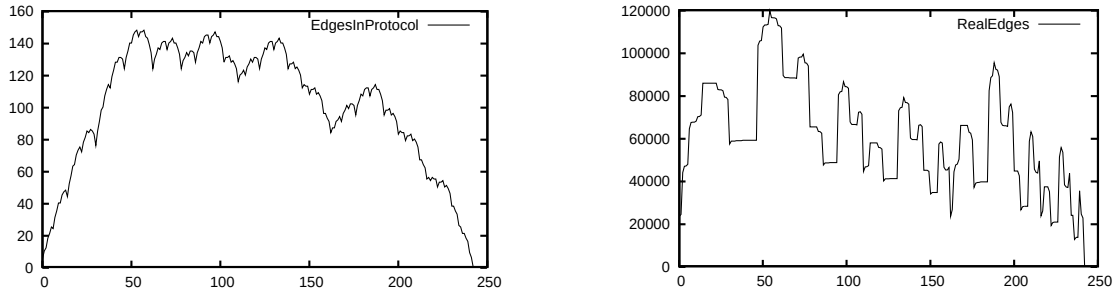
Figures 2 and 3 show the number of edges coming from one part of the the PSS to another. The left image is the graph of $PE_k(l)$ while the right image always shows $CE_k(l)$.

The following heuristics take an inspiration from the graph theory. Their precise definition and evaluation is a subject of future investigation.



(a) (b)

Figure 2. Experiment without apparent relation between $PE_P(l)$ and $CE_P(l)$



(a) (b)

Figure 3. Experiment with apparent relation between $PE_P(l)$ and $CE_P(l)$

Heuristic 5 *Create an areas corresponding to the strongly connected components of the primitive state spaces.*

Heuristic 6 *Find the articulations in the primitive state spaces and split the cuboid at these points.*

Future Work

This paper is on ongoing work, so it does not present compact results. So far, methods for comparing different heuristics were elaborated. The methods were applied to two different heuristics. There is lot of work to evaluate the other heuristics that were only touched in this paper. Also, although the concrete choice depends on the particular formalism, the general ideas might be applied to other formalisms.

Acknowledgments. This work was partially supported by the Ministry of Education of the Czech Republic (grant MSM0021620838).

References

[1] F. Plasil and S. Visnovsky. Behavior Protocols for Software Components. IEEE Transactions on Software Engineering, 28(11), 2002.

[2] F. Lerda and R. Sisto. Distributed-memory model checking with SPIN. In Proc. of the 5th International SPIN Workshop, volume 1680 of LNCS. Springer-Verlag, 1999.

[3] U. Stern and D. L. Dill. Parallelizing the Mur' Verifier. In Computer Aided Verification, pages 256-278, 1997.

[4] S. Ben-David, T. Heyman, O. Grumberg, and A. Schuster. Scalable distributed on- the-fly symbolic model checking. In Formal Methods in Computer-Aided Design, pages 390-404, 2000.
[5] T. Heyman, D. Geist, O. Grumberg, and A. Schuster. Achieving scalability in parallel reachability analysis of very large circuits. In Proc. of the 12th International Conference on Computer Aided Verification. Springer-Verlag, 2000.
[6] T. Poch. Distributed Behavoir Protocol Checker, Master Thesis, Charles University in Prague, 2006.