Two Dimensional Bounded Model Checking for Unbounded Client-Server Systems

Tephilla Prince

August 11, 2022

Two Dimensional Bounded Model Checking for Unbounded Client-Server Systems

Tephilla Prince

Indian Institute of Technology Dharwad, India
tephilla.prince.18@iitdh.ac.in

Abstract

Bounded model checking (BMC) is an efficient formal verification technique which allows for desired properties of a software system to be checked on bounded runs of an abstract model of the system. The properties are frequently described in some temporal logic and the system is modeled as a state transition system. In this work we propose a novel counting logic, $\mathcal{L}_C$, to describe the temporal properties of client-server systems with an unbounded number of clients. We also propose two dimensional bounded model checking ($2D$-BMC) strategy that uses two distinguishable parameters, one for execution steps and another for the number of tokens in the net representing a client-server system, and these two evolve separately, which is different from the standard BMC techniques in the Petri Nets formalism. This $2D$-BMC strategy is implemented in a tool called DCModelChecker which leverages the $2D$-BMC technique with a state-of-the-art satisfiability modulo theories (SMT) solver Z3. The system is given as a Petri Net and properties specified using $\mathcal{L}_C$ are encoded into formulas that are checked by the solver. Our tool can also work on industrial benchmarks from the Model Checking Contest (MCC). We report on these experiments to illustrate the applicability of the $2D$-BMC strategy.

Keywords: Bounded Model Checking, SAT solvers, Petri Nets, Counting Logics, Temporal Logics

Introduction: Model checking [1] is a formal verification technique that allows for desired behavioral properties of a given system to be verified based on a suitable model of the system through systematic inspection of all states of the model. The major challenge of model checking is that the state space of systems might be infinite. In BMC [3, 4, 2] this challenge is overcome by assuming a predetermined bound on the runs, and all possible paths are explored. BMC has been successfully used in the industry by restricting the model checking problem to a bounded problem and verifying properties on bounded runs of the system. This technique uses a single parameter for execution steps, to unfold the system behaviour. Primarily, BMC is used as a bug finding approach in large systems. In this work, we examine BMC of client-server systems with an unbounded number of clients which we represent as Petri Nets. Our key contribution is the extension to standard BMC technique, the two dimensional bounded model checking ($2D$-BMC) strategy [6], where the system is unfolded along two distinguishable parameters, one for execution steps of the net called λ and another for the number of tokens in the net called κ and these two evolve separately.

A Case Study: The Autonomous Parking System (APS) is the running example to demonstrate the $2D$-BMC strategy. We can model the interactions between the parking system and the vehicle requesting parking space by state-transition systems as shown in Fig. 1 and Fig. 2. Consequently, this can be composed into a Petri Net as in Fig. 3. In order to express temporal and counting properties, we require a logic that is more expressible than Linear Temporal Logic (LTL), the Model Checking Contest (MCC) [5] uses LTL with counting. For our case study, we use the counting temporal logic $\mathcal{L}_C$ which is LTL with counting for client-server systems. Consider a system M represented as a Petri Net in Fig. 3 for which we want to verify the following $\mathcal{L}_C$ property ϕ which states that there is always at least one token in either place p_1 or p_7 or p_8. Based on the structure of the net we expect ϕ to hold in Fig. 3. $\phi = G((\#x > 0)p_1(x)|(\#x > 0)p_7(x)|(\#x > 0)p_8(x))$.

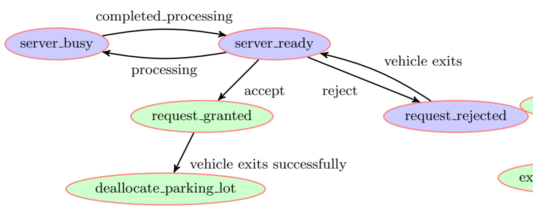

Figure 1: State diagram of APS(server)

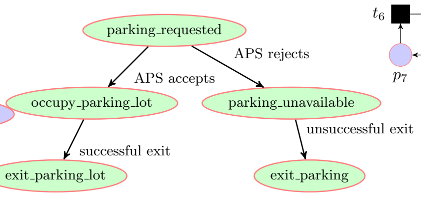

Figure 2: State diagram of vehicle (client)

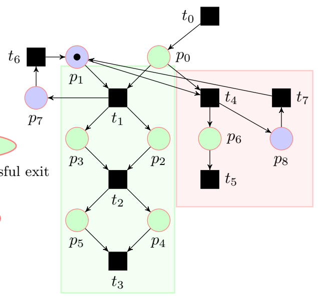

Figure 3: Petri Net for APS

Assume the input bound 100. As shown in Fig. 4 the parameters λ and κ denote the execution steps and the number of tokens in the net respectively.

Our tool negates the property $\psi = \neg\phi$ and encodes the system M against the property ψ for the parameters $k = 0, \lambda = 0, \kappa = 0$, given by the formula $[\mathcal{M}, \psi]_{(0,0)}$. Next, it checks whether this formula is satisfiable by feeding it to the solver. If the above formula is satisfiable at $k = 0$, the property ϕ is violated and we can stop our search since we have obtained a **counterexample**. However, if it is not satisfiable we increment k and consider the next **micro-step**. For a given k each combination of λ and κ such that $k = \lambda + \kappa$ constitutes a micro-step. For $k = 1$, there are two micro steps: $\lambda = 0, \kappa = 1$ where the formula is encoded as $[\mathcal{M}, \psi]_{(0,1)}$ and $\lambda = 1, \kappa = 0$ where the formula is encoded as $[\mathcal{M}, \psi]_{(1,0)}$. The order in which the micro-steps are considered is shown in Fig. 4. In practice, for the property ϕ, we search until the given input bound. As expected, for the given model and bound 100 (i.e, when parameter k reaches 100) it is observed that a counterexample is not found, hence we terminate. This means that the property holds for all traces of the model up to the length 100.

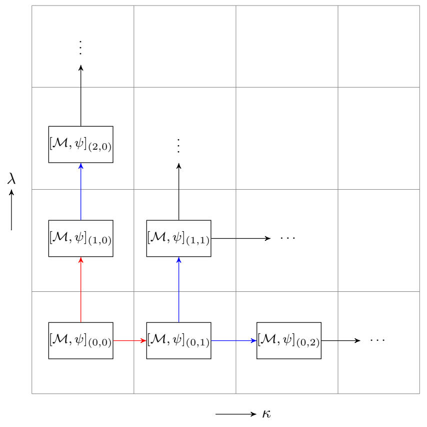
$\vdots$
$[\mathcal{M},\psi]_{(2,0)}$
$\vdots$
λ
$[\mathcal{M},\psi]_{(1,0)}$
$[\mathcal{M},\psi]_{(1,1)}$
$\cdots$
$[\mathcal{M},\psi]_{(0,0)}$
$[\mathcal{M},\psi]_{(0,1)}$
$[\mathcal{M},\psi]_{(0,2)}$
$\cdots$
κ

Figure 4: Unfolding of the encoded formula $[\mathcal{M}, \psi]_{(\lambda,\kappa)}$ with respect to λ (execution steps) and κ (number of tokens)

Table 5 shows results of testing using a model from MCC with the number of sat and unsat properties and execution time (in seconds) for DCModelChecker as well as the state-of-the-art model checking tool ITS-Tools [7]. Our tool was executed with the bound 1 whereas, ITS-Tools was executed with a timeout of 600 seconds. For LTL Fireability and LTL Cardinality, DCModelChecker takes comparatively less execution time.

Model Name	Property Category	DCModelChecker			ITS-Tools		
		sat	unsat	time(s)	sat	unsat	time(s)
Dekker-PT-010	LTL Cardinality	3	13	11.219	4	12	15.7
	LTL Fireability	2	14	10.353	2	14	18.372
	Reachability Cardinality	0	16	10.497	5	11	3.45
	Reachability Fireability	0	16	11.628	4	12	6.061

Figure 5: Results of comparative testing

References

[1] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

[2] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Adv. Comput.*, pages 117–148, 2003.

[3] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *TACAS*, LNCS, pages 193–207. Springer, 1999.

[4] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods Syst. Des.*, pages 7–34, 2001.

[5] F. Kordon, P. Bouvier, H. Garavel, L. M. Hillah, F. Hulin-Hubard, N. Amat., E. Amparore, B. Berthomieu, S. Biswal, D. Donatelli, F. Galla, , S. Dal Zilio, P. G. Jensen, C. He, D. Le Botlan, S. Li, , J. Srba, . Thierry-Mieg, A. Walner, and K. Wolf. Complete Results for the 2020 Edition of the Model Checking Contest. http://mcc.lip6.fr/2021/results.php, June 2021.

[6] Ramchandra Phawade, Tephilla Prince, and S. Sheerazuddin. Bounded model checking for unbounded client-server systems, url:https://iitdh.ac.in/∼prb/2dbmc_tr_2.pdf.pdf. A technical report, 2022.

[7] Yann Thierry-Mieg. Symbolic model-checking using its-tools. In *TACAS*, LNCS, pages 231–237. Springer, 2015.