MODEL CHECKERS –TOOLS AND LANGUAGES FOR SYSTEM DESIGN- A SURVEY

Shubha Raj K B and Suryaprasad J

Department of Computer Science and Engineering,
PESIT-Bangalore South Campus, Bengaluru, Karnataka, India
{shubharajkb, surya}@pes.edu

ABSTRACT

For over four decades now, variants of Model Checkers are being used as an approach for formal verification of systems consisting of software, hardware or combination of both. Though various model checking tools are available like NuSMV, UPPAAL, PRISM, PAT,FDR, it is difficult to comprehend their usage for systems in different domains like telecommunication, automobile, health and entertainment. However, industry experts and researchers have showcased the use of formal verifications techniques in various domains including Networking, Security and Semiconductor design. With current generation systems becoming more complex, there is an urgent need to better understand and use appropriate methodology, language and tool for definite domain. In this paper, we have made an effort to present Model checking in detail with relevance to available tools and languages to specific domain. For novices in the field, this paper would provide knowledge of model checkers languages and tools that would be suitable for various purposes in diverse systems.

KEYWORDS

Formal Methods, Formal Verification, Model Checkers, System Modelling

1. INTRODUCTION

The cutting edge technologies in some of the critical systems like Cyber Physical Systems, Mobile Cloud Computing, Wireless Sensor Network and Mobile Crowd Sensing systems have high degree of complexity. The complexities of these systems are due to the challenges in conformance of software, hardware, network, telecommunication and mobile industry. Large software systems comprise of several million lines of source code. Additional complexities could include structural, environmental (Reactive, Ubiquitous, context-aware), application domain and communication complexity. Hence, it is difficult to understand the requirements, architecture, design, implementation and testing of these systems unless a precise engineering notion is used. One of the systematized tactics that is prevalent to achieve reliable and correct system is Formal Methods (FM) [1][2][3].

Formal methods are techniques used to model complex systems as mathematical entities [18]. They are required for specifying and verifying the system and to ensure that the system is developed "correctly". Formal methods may not be suitable for all types of applications like; problems over simple domains are usually less complex and do not warrant formal methods. Formal methods are vital whenever the cost of failure is high in business critical systems, safety critical systems and machine critical systems [9].

David C. Wyld et al. (Eds) : ICAITA, CDKP, CMC, SOFT, SAI - 2016
pp. 39– 51, 2016. © CS & IT-CSCP 2016 DOI : 10.5121/csit.2016.61304

The development of safety-critical systems requires the use of formal methods for specifying and analyzing critical components and their properties. Formal methods are used at all stages of System development that is at Specification [2][6], Architecture [11][22][13], Design[24][8], Coding[15][16][17] and Testing[18][9].

Typically Formal methods are employed to

- Verify the mathematic model at any time of system development
- Find errors at early stage
- Reduce system cost, development time and effort to build the system
- Improve quality of the system like security, reliability and performance and so on

An application which primarily uses the traditional structured development techniques may use formal methods only for the purpose of documenting data dictionaries. The objectives of these applications will have different impacts on the development process and consequently will influence different choices of model checking techniques. In Formal Methods, mainly we have two main steps. One is Formal Specification and the second one is Formal Verification. Currently there are many techniques that are employed to verify the correctness of system being built like inspection, audit, testing, review, simulation, walkthrough and formal verification. Among these, the Formal Verification techniques offer a formal proof grounded on mathematical model of the system. We need suitable methodologies, tools and languages which can assist in early detection of defects in a structured manner. One such option would be to explore formal verification techniques. There are two types of formal verification; Automatic (Model Checkers) and semi-automatic (Theorem Provers). Here, we focus primarily on Model checkering Languages and Tools with a discussion on current challenges.

There are mainly three major steps in model checking process, namely:

1. System Specification
2. System Modeling
3. System Verification

The rest of the paper is structured as follows: In Section 2, we present one of the inputs of the model checking tool that is Formal specification methods. Section 3 outlines the Formal modelling methods. Section 4 summarizes the formal verification methods. Section 5 briefly discusses the types of model checking methods and their corresponding tools. Section 6 describes model checkers tools and their practical application in various domains. In Section 7, model checking languages are described in detail. Section 8 draws conclusion.

2. SYSTEM (PROPERTY/FEATURE) SPECIFICATION

To apply formal methods, first we should know the characteristics of the problem domain and the complexity of their modeling [4. Typically, we can specify the properties of the system using temporal logic and automata. Table 1 list the distinct types of Temporal Logics and their abbreviations.

Table 1. Distinct Types of Temporal Logics.

Temporal-based PSL (Property Specification Languages)		Stochastic Logics
LTL	Linear Temporal Logic	Continuous Stochastic Logic (CSL)
CTL	Computational Temporal Logic	Probabilistic Reward Computation Tree Logic (PRCTL)
CTL*	Combination of LTL and CTL	Continuous Stochastic Reward Logic (CSRL)
PLTL	Probabilistic LTL	Continuous Stochastic Logic (CSL)
PCTL	Probabilistic CTL	Continuous Stochastic Logic (CSL)
TCTL	Timed CTL	Probabilistic Reward Computation Tree Logic (PRCTL)

Temporal logic is a variant of modal logic for expressing temporal modalities and representing propositions qualified in terms of time. Depending upon the system types, we have to select the distinct types of temporal logic. LTL is more suitable for specifying sequential systems. Whenever we have to verify branching cases in some of the states, then CTL is more suitable. When systems are stochastic in nature, during that time Probabilistic CTL and Probabilistic CTL are appropriate PSL. When we need branching CTL and TCTL are more applicable. Stochastic model may be continuous or discrete. Some of the stochastic logics are CSL, PRCTL and CSRL.

3. SYSTEM MODELLING METHODOLOGIES

Inputs to Model Checking are system modelling and system specification. Once we know the system specifications from the requirement document, next step is to model the system. We can model the system using Finite Automata or a graph at the early stages of system development like at architecture or design level. During design or at architecture level, we have to model systems without implementation (code). When we have to verify the source code, then we have to extract the model from source code. Software model checkers are suitable for extracting model from the source code. There are different methods to Model system formally as specified in the Table 2. Some are text based and others are graphics based. Graph based include Petrinets, state chart and Statemate.

Table 2. Distinct Methods to model the systems

Approaches for modelling System	Characteristics	
Finite State Machine [8]	It is an abstract method having finite number of states. The modelled system can be in only one state at any particular instance in time. Here we have timed automata and hybrid automata.	
Labelled Transition System (Kripke Structure)[19]	It is a labelled transition graph that can sufficiently capture the temporal behaviour of reactive systems	
Model extraction from code (Software Model checkers)	Model extraction from some of the programming languages like C, JAVA and .NET and their corresponding tools are listed below.	
	C	CBMC (C Bounded Model Checker), BLAST (Berkeley Lazy Abstraction Software Verification Tool), CPAchecker (Configurable Program Analysis Checker), DSVerifier (Digital Systems Verifier), ESBMC, LLBMC (Low-Level Bounded Model Checker), SATABS (SAT-based Predicate Abstraction for ANSI-C)
	JAVA	JavaPathFinder, BANDERA
	.NET	MoonWalker

Process Algebra	It is a framework to model concurrent systems. Under this framework we have CSP, CCS and ACP.
Petrinets, Statechart, Statemate	Graph based formal specification languages

Table 3 provides input regarding three main semantic models according to untimed, timed and stochastic/probabilistic system category. Once we understand the basic fundamental concepts to model a given system, later it is easy to select the model checking tools. Otherwise selecting the tool is very difficult task. During the design time, if our aim is to check time critical system, then we have to use timed transition system.

Table 3. Types of Semantic model and their purposes

Types of Semantic Model	Purpose	MC	Applications
Label Transition System	• For untimed systems Finite state Machine Process Algebraic based Model Graph Transition Model	NuSMV [19], CADENCE SMC, ARC,DIVINE Edinburgh CWB, GEAR,LTSA, LTSmin	Used whenever we need only the concurrency problem without timing and stochastic features
Timed Transition System	• Timed automata • Timed process Algebra	UPPAAL,RED PAT,MRMC	When precise constraints on the timing of events are needed, timed automata are the high-level model.
Probabilistic Semantic Model	For Markov Decision Process	PRISM[21][22], MRMC, [64],CADP[24], Modest, Toolset[25] Mobius [26]	Used to model various sources of uncertainty [20].

4. SYSTEM VERIFICATION

According to ISO/IEC/IEEE 15288, system verification includes set of activities that compares a system or system elements against the requirements, architectures [29] and design characteristics and other properties to be verified including deadlock freeness, safety, fairness and aliveness. System verification is used to establish that the design, product or system under consideration possesses the requisite properties.

The system verification methods can be broadly classified into three categories based on the system components or implementation, namely: (a) hardware verification [34] (b) software verification, and (c) System Integration verification. Quality expectations/predications are very high need in hardware systems due to higher fabrication cost and testing involved. Formal Verification uses mathematical reasoning to guarantee the absence of errors. It is an effective bug hunting technique [20]. Testing checks that system behaves correctly under a finite number of test cases, whereas formal verification is designed to be exhaustive.

Different types of Formal Verification (FV) Techniques

- Model Checker (e.g.: SPIN, UPPAAL, PAT, FDR, NuSMV)
- Theorem Proving (e.g.: PVS, HOL [35])

Theorem Proving is used for system with infinite number of states. Commercial use of automated theorem proving is in integrated circuit design and verification. Some of the companies like Intel and AMD use automated theorem proving to verify that division and other operations are correctly implemented in their processor design. But in this paper our aim is describe model checking techniques.

FV is used in various domains for distinct purposes as below.

- Development of Integrated Circuits
- For verifying Electronic Design Automation (EDA) tools
- Stringent regulations for certifications
- To check the correctness of device drivers, cryptographic and communication protocols
- Embedded Control systems [14].
    - Medical devices such as pacemakers and sensors
    - Communication networks
- Safety critical system
    - Avionic Industry, Nuclear energy, Process control, Robotics, Transport, Medical.

5. TYPES OF MODEL CHECKING

Model Checkers **(MC)** are classified as Modern Model Checkers (Software MC) and Traditional Model Checkers [1]. Another category is based on types of verification algorithms as explained below.

- Explicit Model Checker
- Symbolic Model Checker
- Bounded Model Checker
    - SAT (Propositional Satisfiability): In BMC, Model checking complexity is reduced to a propositional satisfiability problem that can be solved with SAT (Satisfiability) solvers. So the size of state space will decrease and increases the performance (speed).
    - SMT (Satisfiability Modulo Theories) [27]: It is an extension of propositional satisfiability (SAT) which is the most well know constraint-satisfaction problem. It generalizes Boolean satisfiability by adding equality reasoning, arithmetic, fixed-size bit vector, arrays, quantifiers and other useful first-order theories.

- On-line Model checkers [28]
- Parallel Model checkers
- Software Model checkers

Depending upon the choices of the system model [FSM, Process Algebra, GTS] and property specification methods [temporal Logic, automata], different Model checking approaches like Explicit MC, Symbolic MC, Bounded MC and On-line MC (OMC). Parallel model checkers are used in Semiconductor Industry for verifying multicore processor.These distinct model checker are typically classified by how states are stored and manipulated. Table 4 gives brief knowledge about the characteristics of the different model checkers and listed reduction methods used in different types of model checking tools.

Table 4. Model checkers types and their characteristics

Explicit Model Checker	**Symbolic Model Checker**	**Bounded Model Checker**	**On-line Model checkers**
States are indexed directly	Second generation MC. Here state of the system is represents by Boolean Functions.	Bounded because only states reachable within a bounded number of times. If length of the path can't be found at a given length K, then the search is continued for higher than K.	Here parameters of models are continually adjusted to remedy possible modelling faults [18] [17].
Graph algorithms are used to explore the state space starting from the initial state	To resolve the problem of state space explosion by enumerating states symbolically	It is a successor of propositional SAT solvers	It drops the need for models to be accurate far into the future.
It construct State Transition Graph by reclusively generating successor of the Initial state	Boolean formulas are represented using the data structure BDD or OBD to improve the efficiency	Used in semiconductor industry	OMC offers safety assurances for short time frames only and renews these assurances continually during operation
Graph is created using DFS (Depth First Search), BFS (Breadth First Search) or in Heuristic manner		Applicable to system level software	It is used in Medical domain because patient model is like to be inaccurate as the physiology of the human body is complex and differs between individuals.
Methods: Here Partial Order Reduction (POR) is used for state-space exploration problem		Best to find shallow bugs [programs without deep loops]	Example: Heart rate and oxygen in the blood depend on the patient's condition
It depends on extensive search through explicit representation of reachable system states [7].		It supports full counterexample trace. SAT based Bounded Model Checking is typically quicker in finding bugs compared to BDDs.	A generalized model will always slip individual characteristics

Table 5, list the tools developed under each model checking type and their limitation. Depending upon the system types and types of properties we have to verify, we have select more than one model checking tools in system development. DIVINE [30[31] is an example for Parallel Model checker. Limitations of parallel Model checkers are communication and load balancing.

Table 5. List of Model Checking Tools and their limitations

Explicit Model Checker	Symbolic Model Checker	Bounded Model Checker	On-line Model checkers
Tools: SPIN, ZING [7][23] PRISM [21][22] DiViNe	**Tools**: SAL, NuSMV BEBOP MOPED SMART MC CadenceSMV,	**Tools**: **SAT-based:**SAL (Symbolic Analysis Laboratory) NuSMV **SMT**-based [27] Z3, CVC4	**Tools**: Java PathExplorer (JPaX) [32]
Limitation: It suffers from the state-space explosion problem due to the exponential growing of the explicit state space	**Limitation**: Human verifier manually adjusts the order of the state variable. This can impact the size of the Binary Decision Diagram (BDD).So performance will reduce. Solution is BMC	**Limitations**: In SAT procedure, the variables must be Boolean type. Due to this it is inexpressive for industrial problems. Example: Computer Programs variables of type other than Boolean must be encoded into Boolean/bit variables which can result in a large formula. **Solution**: Alternative new technique is SMT (Satisfiability Modulo Theories).	
		Why BMC? SMT-based BMC are more expressive than SAT-based BMC.	**Why OMC:** It permits safety assignment at all times and provides means to react before safety violations occur.

Table 6, showcases few software model checkers and their purpose for different programming langauges as well as solutions to the state space explosion and their purpose. SLAM project is used for making reachability analysis for large sequential C programs mainly to Device Driver [1]. BLAST is to prevent all memory safety violations. JPF is used for verification and testing environment for Java programs.

Various abstraction techniques are used depending upon the usage. Some of the abstarction techniques are modular, lazy, Process counter abstraction, parallel modle checking and slicing methods. Few reduction methods are symmetric reduction, and partial order reduction. To achive good performance advanced optimization methods are now a days adapted in various model checking tools.

Table 6. Software model checkers

Year	Tools	Developed by	State-explosion solutions used
2000	(SLAM project) CHESS MODIST	Microsoft	Modular Abstraction BDD
2002	BLAST	University of California Berkeley	Lazy abstraction BDD
2002	JPF (Java PathFinder) [5]	NASA	• Compression techniques is used to handle big states • Partial Order Method • Symmetric Reduction method • Slicing Abstraction • Runtime analysis Techniques

Some of the demerits of Model checking are listed as below.

- Temporal logic specifications are complex.
- Writing specifications is difficult.
- State explosion is a major problem.

6. MODEL CHECKER TOOLS

Depending upon the characteristics of the system, application domain, here are the some of the examples of application of tools for a particular system as shown in the Table 7. The objectives for applying a formal method to a project must be clearly identified and documented. Though we have described in the previous sections about system specification and modelling methods, for detailed understanding of formal languages, we are describing model checking languages in the next section 7. Various model checking tools are used for distinct domain like communication, embedded system, Software Engineering, hardware and healthcare so on. Here we have listed few purposes like to model concurrent software, real-time systems, clock synchronized protocols, synchronous digital logic, asynchronous systems, consistency of software data structure. Table 7, list few model checking tool names and their abbreviation. For SPIN model checker, Promela is the specification language. It is based on the process algebra. In UPPAAL, for formal modelling timed automata is used for system modelling.

Table 7. MC tools and their abbreviation

Tool	Abbreviation	Developed By
SPIN	Simple Promela Interpreter	Bell Labs
UPPAAL	Uppsala and Aalborg	Uppsala and Aalborg University
NuSMV	New Symbolic Model Verifier	Carnegie Mellon University
FDR	Failures Divergences Refinement	University of Oxford
PRISM	Probabilistic Model Checker	University of Birmingham

Table 8. List of model checker tools and their applicability

MC	Purpose	Domain
SPIN	Used to model concurrent software or asynchronous processes.	Communication protocols
UPPAAL	Used to model real-time systems Formal model and analysis of clock-synchronised protocols in sensor networks based on timed automata [36].	Timed systems
NuSMV	Used to model synchronous digital logic.	Digital Circuits
FDR	Used to model asynchronous systems	
Alloy	Used to analyse consistency of software data structures	Requirement Analysis
Simulink Design Verifier	Used to verify models created in Simulink, a data-flow and state-machine simulation	Hardware Circuits
SAT Solvers[19]	Used to in Electronic Design Automation (EDA) community for checking correctness of Hardware designs mainly in synthesis and verification.	Hardware design
PRISM	Formal model of flooding and gossiping protocols for analysing their performance probabilistic properties [35],Cardiac Pacemakers [37] For automatically verify whether STRAC (Spatio-Temporal Access Control based on Reputation, one policy for the IoT) policies conform to security properties [38]	Healthcare

7. MODEL CHECKERS LANGUAGES

There are two different types of specification languages to construct a system model. One is state based and another is Event based languages [2]. Both the methods are specified in Figure 1.

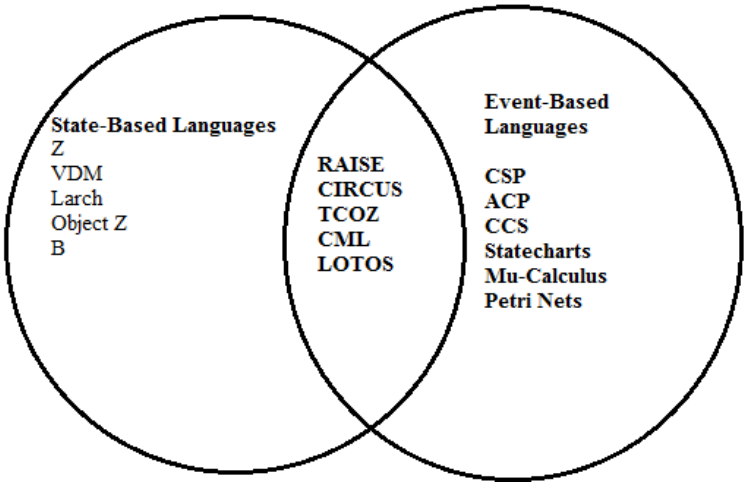

Figure 1. State-based and event-based Specification Languages

Figure 2, lists the important concepts used to develop formal specification languages. Sets, Relations, functions are used in state based and Trees. Graph and automata are used in event based formal specification languages.

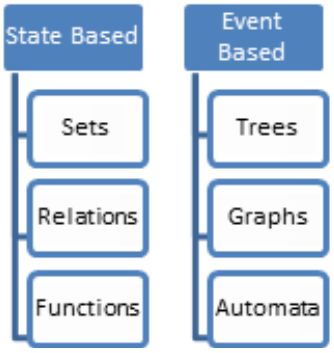

Figure 2. Concepts used in formal specification languages.

Current need is to integrate the distinct specification languages to handle different features of the systems as itemized in Table 8. RAISE is used for handing rich state space. CIRCUS is used for specification, programming, and verification by refinement. Its semantics is grounded on Hoare. It motivated by the need for a notation and techniques to reason about designs and implementations of state-rich reactive processes. Used for reasoning about Safety-critical Java programs (SCJ), avionics control systems, and Systems of Systems. TCOZ supports Object-Oriented principles. CML is a combination of Circus and VDM developed for the modelling of Systems of Systems (SoS).

LOTOS is used for handling concurrency complexities. LOTOS is used for protocol specification in ISO OSI standards LOTOS is an algebraic language that consists of two parts: a part for the description of data and operations, based on abstract data types, and a part for the description of concurrent processes, based on process calculus.

Table 9. Integration of specification languages

State-based Language	**Event-based Language**	**Extension**	**Integrated Specification Language**
VDM	CCS	VDM+CCS	RAISE
Z	CSP		CIRCUS
Object Z	Timed CSP	Object Z +Timed CSP	TCOZ
VDM	CIRCUS	VDM+CIRCUS	CML (COMPASS Modelling Language)
	CSP, CCS	CSP+CCS	LOTOS (Language Of Temporal Ordering Specification) [2]

8. CONCLUSION

As systems get complex, a seamless flow becomes imperative. A formal method helps to reduce errors, cost and ensure that the developed system will meet all expectations. Understanding the role of tools and languages irrespective of the application domain will invariably be a great asset to all concerned. The role of Model Checkers in design and verification of systems with relevant tools and languages is presented in detail. Suitability of Model Checkers tools and languages in various application domains is mainly based on few characteristics of the domain including sequential, parallel, timed, untimed, etc. Novices in the field will get a broad view of role of Model Checkers in system design and verification. Showcasing the usage of respective tool and

language in different domains would be appropriate but is outside the scope of this paper and can be considered for future work.

REFERENCES

[1] Strunk, E. A, Aiello, M. A & Knight, J. C. (2006) "A survey of tools for model checking and model-based development", *University of Virginia*.

[2] Clarke, E. M & Wing, J. M. (1996) "Formal methods: State of the art and future directions", *ACM Computing Surveys (CSUR)*, Vol. *28, No.*4, pp626-643.

[3] Kern, C & Greenstreet, M. R. (1999) "Formal verification in hardware design: a survey", *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Vol. *4,* No.2, pp123-193.

[4] Liu, Y, Sun, J & Dong, J. S. (2011) "Pat 3: An extensible architecture for building multi-domain model checkers",*IEEE 22nd International Symposium on Software Reliability Engineering* pp190-199.

[5] Zhang, H, Aoki, T, & Chiba, Y. (2015) "Yes! You can use your model checker to verify OSEK/VDX applications", *IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)* pp1-10.

[6] Lamsweerde, A. V. (2000) "Formal specification: a roadmap", *Proceedings of the Conference on the Future of Software Engineering* pp147-159. ACM.

[7] Chen, T, Diciolla, M, Kwiatkowska, M, & Mereacre, A. (2012) "Quantitative verification of implantable cardiac pacemakers", In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd* pp263-272

[8] Qadir, J, & Hasan, O. (2015) "Applying Formal Methods to Networking: Theory, Techniques, and Applications", *IEEE Communications Surveys & Tutorials*, Vol. *17. No.*1, pp256-291.

[9] Gaudel, M. C. (1994). "Formal specification techniques", *Proceedings of the 16th international conference on Software engineering*. pp 223-227. IEEE Computer Society Press.

[10] Belinfante, A. (2010) "JTorX: A tool for on-line model-driven test derivation and execution", In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* pp266-270. Springer Berlin Heidelberg.

[11] Medvidovic, N., & Taylor, R. N. (2000) "A classification and comparison framework for software architecture description languages", *IEEE Transactions on software engineering*, Vol *26*. No.1, pp70-93.

[12] Clements, P. C. (1996) "A survey of architecture description languages", *Proceedings of the 8th international workshop on software specification and design* pp16). IEEE Computer Society.

[13] Hermerschmidt, Lars, et al (2015) "Generating Domain-Specific Transformation Languages for Component & Connector Architecture Descriptions", *2nd International Workshop on Model-Driven Engineering for Component-Based Software Systems (ModComp)* Workshop Proceedings.

[14] Jiang, Y., Zhang, H., Li, Z., Deng, Y., Song, X., Gu, M., & Sun, J. (2015), "Design and optimization of multiclocked embedded systems using formal techniques", *IEEE Transactions on Industrial Electronics*, Vol.*62* No.2, pp1270-1278.

[15] Havelund, K., & Pressburger, T. (2000) "Model checking java programs using java pathfinder", *International Journal on Software Tools for Technology Transfer*, pp366-381.

[16] Corbett, J. C, Dwyer, M. B, Hatcliff, J, Laubach, S, Pasareanu, C. S., & Zheng, H. (2000), "Bandera: Extracting finite-state models from Java source code", *Software Engineering, 2000. Proceedings of the 2000 International Conference on* pp439-448. IEEE.

[17] Ammann, P. E, Black, P. E, & Majurski, W (1998), "Using model checking to generate tests from specifications", *Formal Engineering Methods, 1998. Proceedings. Second International Conference on* pp46-54. IEEE.

[18] Utting, M., & Legeard, B. (2010). "*Practical model-based testing: a tools approach*", Morgan Kaufmann.

[19] Baier, C., Katoen, J. P., & Larsen, K. G. (2008) *Principles of model checking*. MIT press.

[20] Norman, G & David P (2014) "Quantitative Verification: Formal Guarantees for Timeliness, Reliability and Performance".

[21] Barnat, J, Brim, L, Ceska, M, & Rockai, P (2010),"Divine: Parallel distributed model checker", *Parallel and Distributed Methods in Verification, 2010 Ninth International Workshop on, and High Performance Computational Systems Biology, Second International Workshop on* pp4-7. IEEE.

[22] Kwiatkowska, M, Norman, G, & Parker, D. (2011), "PRISM 4.0: Verification of probabilistic real-time systems", *International Conference on Computer Aided Verification.* pp. 585-591. Springer Berlin Heidelberg.

[23] Andrews, T, Qadeer, S, Rajamani, S. K, Rehof, J, & Xie, Y. (2004), "Zing: Exploiting program structure for model checking concurrent software", *International Conference on Concurrency Theory.* pp. 1-15. Springer Berlin Heidelberg.

[24] Petri, C. A. (1986)"Concurrency theory", *Advanced Course on Petri Nets",* pp4-24, Springer Berlin Heidelberg.

[25] Markov Reward Model Checker. www.mrmc-tool.org, October 2016

[26] CADP Home page. www.inrialpes.fr/vasy/cadp/, October 2016.

[27] Phan, Q. S., & Malacaria, P. (2015) "All-Solution Satisfiability Modulo Theories: applications, algorithms and benchmarks' *Availability, Reliability and Security (ARES), 2015 10th International Conference on* pp100-109. IEEE.

[28] Jard, C., & Jeron, T. (1989). "On-line model-checking for finite linear temporal logic specifications", *International Conference on Computer Aided Verification,* pp189-196).Springer Berlin Heidelberg.

[29] Shaw, M, & Garlan, D. (1996), *Software architecture: perspectives on an emerging discipline,* Prentice Hall.

[30] Weiqiang, K. O. N. G., Shiraishi, T., Katahira, N., Watanabe, M., Katayama, T., & Fukuda, A. (2011). An smt-based approach to bounded model checking of designs in state transition matrix. *IEICE transactions on information and systems*, Vol.*94* No.5, pp946-957.

[31] Barnat, J., Brim, L., Havel, V., Havlíček, J., Kriho, J., Lenčo, M., ... & Weiser, J. (2013). "DiVinE 3.0–an explicit-state model checker for multithreaded C & C++ programs", *International Conference on Computer Aided Verification* pp. 863-868. Springer Berlin Heidelberg.

[32] The M¨obius Tool. www.mobius.illinois.edu, accessed on October 2016.

[33] PAT Web site http://pat.comp.nus.edu.sg/ accessed on October 2016

[34] Ardakani, H. H, Gharehbaghi, A. M, & Hessabi, S(2007), "A performance and functional assertion-based verification methodology at transaction-level", *2007 Internatonal Conference on Microelectronics* pp133-136. IEEE.

[35] Fehnker, A, & Gao, P(2006), "Formal verification and simulation for performance analysis for probabilistic broadcast protocols", *International Conference on Ad-Hoc Networks and Wireless* pp28-141. Springer Berlin Heidelberg.

[36] Heidarian, F, Schmaltz, J, & Vaandrager, F. (2012), "Analysis of a clock synchronization protocol for wireless sensor networks", *Theoretical Computer Science*, Vol. *413* No.1, pp87-105.

[37] Chen, T, Diciolla, M, Kwiatkowska, M, & Mereacre, A. (2012) "Quantitative verification of implantable cardiac pacemakers", *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd* pp 263-272. IEEE.

[38] Yunchuan, G, Lihua, Y, & Chao, L.(2014) "Automatically verifying STRAC policy", *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on* pp141-142. IEEE.

AUTHORS

Shubha Raj K B She is an Assistant Professor at PESIT-BSC, Bengaluru, Karnataka, India in the Department of Computer Science and Engineering. Currently she is pursuing PhD at VTU Belagavi in Formal Methods. She obtained her MTech degree from RVCE, Bengaluru. Her research interests are in the field of Formal Verification, Model checking, Software Architecture and Architecture Description Languages.

Suryaprasad J He is Director/Principal at PESIT-BSC, Bengaluru, Karnataka, India. He obtained his PhD from Florida Atlantic University, Boca Raton. His research interests are in the field of System level design methodologies, Hardware Software Co-design Embedded System Design, Power Optimal Design, Verification Methodologies and Advanced Programming Metodologies.