SMT-Based Context-Bounded Model Checking for Embedded Systems: Challenges and Future Trends

Lucas C. Cordeiro
Electronic and Information Research Center
Federal University of Amazonas, Brazil
Email: lucascordeiro@ufam.edu.br

Eddie B. de Lima Filho
Science, Technology, and Innovation Center
for the Industrial Pole of Manaus, Brazil
Email: eddie@ctpim.org.br

ABSTRACT

The dependency on the correct functioning of embedded systems is rapidly growing, mainly due to their wide range of applications, such as micro-grids, automotive device control (*e.g.*, airbag control), health care, surveillance, mobile devices, and consumer electronics. Their structures are becoming more and more complex and now require multi-core processors with scalable shared memory, in order to meet increasing computational power demands. As a consequence, reliability of embedded (distributed) software becomes a key issue during system development, which must be carefully addressed and assured. Normally, state-of-the-art verification methodologies for embedded systems generate test vectors (with constraints) and use assertion-based verification and high-level processor models, during simulation; however, other additional challenges have been raised: the need for meeting time and energy constraints, handling concurrent software, dealing with platform restrictions, evaluating implementation-structure choices, and supporting new software architectures and legacy designs (usually written in low-level languages). The present paper discusses challenges, problems, and recent advances to ensure correctness and timeliness regarding embedded systems. Reliability issues, in the development of micro-grids and cyber-physical systems, are then considered, as a prominent (bounded) model checking application.

1. INTRODUCTION

Generally, embedded computer systems perform dedicated functions with high degree of reliability. They are used in a variety of sophisticated applications, which range from entertainment software, such as games and graphics animation, to safety-critical systems, such as nuclear reactors and automotive controllers [1]. Embedded systems are ubiquitous, in modern day information systems, and are also becoming increasingly important in our society, especially in micro-grids, where reliability and carbon emission reduction are of paramount importance [2], and in cyber-physical systems (CPS), which demand short development cycles and again high-level of reliability [3, 4]. As a consequence, human life has also become more and more dependent on the services provided by this type of system and, in particular, their success is strictly related to both service relevance and quality.

Figure 1 shows some examples of embedded systems, which typically consist of a human-machine interface (*e.g.*, keyboard and LCD), a processing unit (*e.g.*, real-time computer system), and an instrumentation interface (*e.g.*, sensor, network, and actuator) [1]. Indeed, many current embedded systems, such as unmanned aerial vehicles (UAVs) [5] and medical monitoring systems [6], become interesting solutions only if they can reliably perform their target tasks. For instance, UAVs are a trend on military missions, due to the absence of pilots; however, an incorrect plan execution may also cost human lives, which is unacceptable. In addition, wrong disease diagnosis or condition-evaluation reports have the potential to compromise patients' health, with serious consequences.

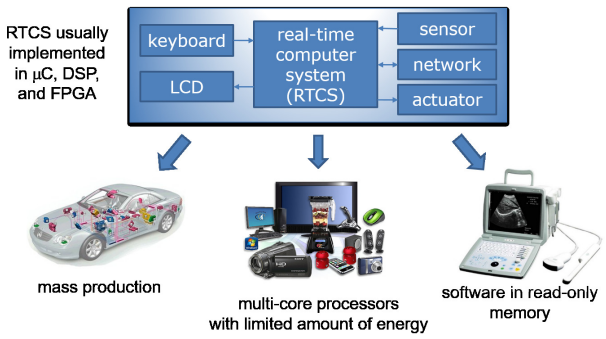

Figure 1: An embedded system is part of a well-specified larger system (intelligent product).

Besides, when physical interaction with the real world is needed, which happens in CPS, additional care must be taken, mainly when human action is directly replaced, as in vehicle driving. Regarding the latter, even human-in-the-loop feedback control can be employed [7], which then raises deeper concerns related to the reliability of human behavior modeling and system implementation. Consequently, it is important to go beyond design correctness and also address behavior correctness, which may be performed by incorporating system models.

A number of distinctive characteristics might influence the embedded system development and verification process, which include: mass production and static structure, functionality determined by software in read-only memory, multi-core processors with scalable shared memory, and limited amount of energy. Additionally, the increasing computational power and decreasing size and cost, which are common to the area of computer processors, are enabling system designers to move more features to software domain, which consequently leads to difficulties in verifying design correctness, since stringent constraints imposed by the underlying hardware (*e.g.*, real-time, memory allocation, interrupts, and concurrency) must be considered during verification [8].

2. BOUNDED MODEL CHECKING (BMC)

Bounded Model Checking (BMC) based on Boolean Satisfiability (SAT) was originally proposed to verify hardware designs [9, 10]. Indeed, Biere *et al.* were able to successfully verify large digital circuits with approximately 9510 latches and 9499 inputs, leading to BMC formulae with 4×10^6 variables and 1.2×10^7 clauses to be checked by a standard SAT solver. BMC based on Satisfiability

Modulo Theories (SMT) [11], in turn, was originally proposed to deal with increasing software verification complexity [12].

In general, BMC techniques aim to check the violation of a given (safety) property at a given system depth, as shown in Figure 2. Indeed, given a transition system M, which is derived from the control-flow graph of a program, a property ϕ, which represents the program correctness and/or the system's behavior, and an iteration bound k, which limits the loop unrolling, data structures, and context-switches, BMC techniques thus unfold the system k times, in order to convert it into a verification condition ψ, expressed in propositional logic or in a decidable-fragment of first-order logic, such that ψ is *satisfiable* if and only if ϕ has a counterexample of depth less than or equal to k.

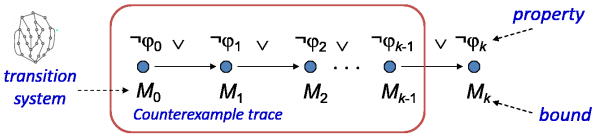

Figure 2: Bounded Model Checking.

From the practical point of view, SAT-based or SMT-based BMC procedures have been successfully applied to verify a large number of hardware and software systems, including digital circuits and single- and multi-threaded programs. Those BMC techniques were able to find subtle bugs in real digital and embedded software systems, as reported in the available literature [13, 14, 15, 16, 17]. Nonetheless, the main criticism with respect to BMC techniques relies on completeness, since they are able to prove system correctness only if an upper bound k is known, *i.e.*, a bound that unfolds all loops and recursive functions to their maximum possible depth.

Due to that limitation, BMC tools are typically susceptible to exhaustion of time or memory limits, when checking complex circuit-implementations or programs with loops, whose bounds are too large or cannot be statically determined.

2.1 Induction-based Verification of C Programs

One promising approach to achieve completeness, in BMC techniques, is to prove that an invariant (assertion) is k-inductive [18, 19]. The main challenge, however, of such an approach relies on computing and strengthening inductive invariants from programs. In particular, loop invariants, which were computed from programs under verification, must be inductive (and not just invariant) for the corresponding verification conditions to be valid, *i.e.*, induction cannot determine the invariance of a non-inductive assertion [20]. As a consequence, even if k-induction procedures successfully compute such assertions, which are indeed invariant, those must be inductive, so that verifiers can automatically prove program correctness.

There are several invariant-generation algorithms that discover linear and polynomial relations among integer and real variables, in order to provide loop invariants and also discover the memory "shape", in programming languages with pointers [21, 22]. The current literature also reports successful applications of k-induction based verification algorithms for hardware and software systems, using invariant generation and strengthening, mostly based on interval analysis.

Novel verification algorithms for proving correctness of (a large set of) C programs, by mathematical induction, in a completely automatic way (*i.e.*, users do not need to provide the loop invariant) were recently proposed [23, 24, 25, 26, 27]. Additionally, k-induction based verification was also applied to ensure that (restricted) C programs (1) do not contain violations related to data races [28], considering the Cell BE processor, and (2) do respect time constraints, which are specified during the system design phase [18]. Apart from that, the k-induction algorithm is also a well-established technique in hardware verification, where it is easily applied, due to the monolithic transition relation present in such designs [18, 19, 29].

Nonetheless, there is still little evidence, in the available literature, that model checking hardware and software systems, using k-induction (and invariants), can be efficiently exploited in embedded-system verification. That happens due to the distinctive characteristics mentioned earlier, which influence the embedded system development and also the employed verification processes. Additionally, there is still a lack of studies for embedded software verifiers to exploit the combination of different invariant generation and strengthening algorithms, including analysis to discover linear inequalities, polynomial equalities and inequalities, and invariants about memory and variable aliasing [20].

2.2 Incorporating System Models to Automated Verification Procedures

It is worth noticing that, currently, SMT-based BMC approaches check code properties in real programs, which basically address programing-language issues and general correctness, without taking into account target applications or system behavior. Such a statement is important, since, as already mentioned, many system features are being moved to software domain, which then requires schemes that do not only check if the source code is correctly written, but also if it will properly respond in real environments or under external problems. For instance, the anti-lock braking system software of a vehicle model can be bug free, but it may not work correctly if a sensor is damaged.

Indeed, research in software verification is now incorporating such considerations, during checking processes, and some schemes already use knowledge about the system to be verified and the underlying hardware. Recently, a verification tool for digital systems was proposed, which is called digital system verifier (DSVerifier) [31] and is able to aid engineers to check overflow, limit cycle, output error, timing, stability, and minimum phase, considering finite word-length (FWL) effects. Additionally, DSVerifier checks closed-loop systems with uncertain models considering FWL effects, which are typically represented as hybrid systems, *i.e.*, the controller is digital but the controlled agent (plant) is a physical and continuous system. Here, the verification procedure has to consider the interaction between a continuous plant and a digital (and sampled) controller with FWL effects, which can be connected using different control system configurations. Indeed, DSVerifier is a useful test tool, which takes into account different realization forms (*e.g.*, direct forms, delta forms, and transposed forms) and other implementation restrictions to explore the design space. Ultimately, if the system requirements are not met with a given configuration, an analysis of the provided error report may suggest another setup.

Scratch is another example of a software model checker, which uses knowledge about the system to be verified and the underlying hardware, for detecting races related to direct memory access (DMA), in the Cell BE processor [28]. That tool also uses BMC, in order to detect DMA races, and BMC with k-induction, which aims to prove the absence of races. If support to other DMA

operations were added, Scratch could be adapted to different architectures, *i.e.*, the same techniques would be employed, but with a different system behavior/knowledge.

3. VERIFICATION CHALLENGES

State-of-the-art verification methodologies for embedded systems generate test vectors (with constraints) and use assertion-based verification and high-level processor models, during simulation [30, 32], as shown in Figure 3.

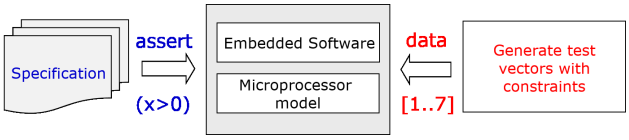

Figure 3: Verification methodologies for embedded systems.

In particular, the main challenges regarding the verification of embedded systems lie on improving coverage, where more system functions are verified, reducing verification time, *i.e.*, pruning the state-space exploration during verification, and incorporating system models, which allow specific checks regarding system behavior and not only code correctness. Additionally, embedded system verification raises additional challenges, such as

1. time and energy constraints;
2. handling of concurrent software;
3. platform restrictions;
4. legacy designs; and
5. support to different programming languages and interfaces;

Indeed, the first two aspects are of extreme relevance in microgrids and cyber-physical systems, in order to ensure reliability, which is a key issue for (smart) cities, industries, and consumers, and the third one is essential in systems that implement device models, such as digital filters and controllers, which present a behavior that is highly dependent on signal inputs and outputs and whose deployment may be heavily affected by hardware restrictions. The fourth aspect is inherent to a large number of embedded systems from telecommunications, control systems, and medical devices. Finally, the last one is related to the evolution of development processes and technologies, which may delay the application of suitable verification approaches.

4. RESEARCH PROBLEM (RP)

This position paper tackles five major problems in computer-aided verification for embedded systems, which are (partially) open in current published research.

(RP1) provide suitable encoding into SMT [11], which may extend the background theories typically supported by SMT solvers, with the goal of reasoning accurately and effectively about realistic embedded programs.

(RP2) exploit SMT techniques to leverage bounded model checking of multi-threaded software, in order to mitigate the state-explosion problem due to thread interleaving.

(RP3) prove correctness and timeliness of embedded systems, by taking into account stringent constraints imposed by hardware.

(RP4) incorporate knowledge about system purpose and associated features, which aims to detect system-level and behavior failures.

(RP5) provide tools and approaches capable of addressing different programming languages and application interfaces, with the goal of reducing the time needed to adapt current verification techniques to new developments and technologies.

Section 5 outlines contributions toward these research problems.

5. CURRENT ACHIEVEMENTS AND FUTURE TRENDS

(RP1) Cordeiro, Fischer, and Marques-Silva proposed the first SMT-based BMC for full C programs, called Efficient SMT-Based Context-Bounded Model Checker (ESBMC) [17], which was later extended to support C++98 programs [36], CUDA programs [37], and Qt-based consumer electronics applications [38]. This approach was also able to find undiscovered bugs related to arithmetic overflow, buffer overflow, and invalid pointer, in standard benchmarks, which were later confirmed by the benchmarks' creators (*e.g.*, NOKIA, NEC, NXP, and VERISEC) [15, 17]. Other SMT-based BMC approaches have also been proposed and implemented [14], but the coverage and verification time of all existing ones are still limited to specific classes of programs, especially for those that contain intensive floating-point arithmetic and dynamic memory allocation [33, 34]. One possible research direction is to bridge the gap between BMC tools and SMT solvers to propose background theories and develop more efficient decision procedures, in order to handle specific classes of programs.

(RP2) The SMT-based BMC approach proposed by Cordeiro, Fischer, and Marques-Silva was further developed to verify correct lock acquisition ordering and the absence of deadlocks, data races, and atomicity violations in multi-threaded software based on POSIX and CUDA libraries [15, 37], considering monotonic partial-order reduction [49] and state-hashing techniques, in order to prune the state-space exploration [39]. Recent advances for verifying multi-threaded C programs have been proposed to speed up the verification time, which significantly prune the state-space exploration [35, 40]; however, the class of concurrent programs (*e.g.*, CUDA, OpenCL, and MPI) that can be verified is still very limited. One possible research direction is to further extend BMC of multi-threaded C programs via Lazy Sequentialization [35], in order to analyze unsatisfiability cores [41] with the goal to remove redundant behaviour or to analyze interpolants [42] to prove non-interference of context switches.

(RP3) Novel approaches to model check embedded software using k-induction and invariants were proposed and evaluated in the literature, which demonstrate its effectiveness in some real-life embedded-system applications [23, 25, 26, 28]; however, the main challenge still remains open, *i.e.*, to compute and strengthen loop invariants to prove program correctness and timeliness in a more efficient and effective way, in order to be competitive with other model-checking approaches. In particular, invariant-generation algorithms have substantially evolved over the last years, with the goal of discovering inductive invariants of programs [21, 22] or continuously refine them during verification [24]; however, there is still a lack of studies for exploiting the combination of different invariant-generation algorithms (*e.g.*, interval analysis, linear inequalities, polynomial equalities and inequalities) and how to

strengthen them during verification, in order to ensure system robustness w.r.t. implementation aspects.

(RP4) State-of-the-art SMT-based context-BMC approaches were extended to verify overflow, limit cycle, time constraints, stability, and minimum phase, in digital systems. Indeed, digital filters and controllers [31, 43, 44] were tackled, in order to specify system-level properties of those systems, using linear-time temporal logic [45]. In particular, a specific UAV application was tackled, with the goal to verify its attitude controllers [46]. In general, however, there is still a lack of studies to verify system-level properties related to embedded systems; emphasis should be given to micro-grids [2] and cyber-physical systems [4], which require high-dependability requirements for computation, control, and communication. Additionally, the application of automated fault detection, localization, and correction techniques to digital systems represents an important research direction to make BMC tools useful for embedded systems engineers [50].

(RP5) Although ESBMC [17] was extended to support C/C++ and some variants, new application interfaces and programming languages are often developed, which require suitable software verification tools. Indeed, it would be interesting if a new programming language model could be loaded, which along with a BMC core could check different programs. Some work towards that was already presented by Sousa, Cordeiro, and Filho [51], which employed operational models for checking Qt-based programs from consumer electronics. In summary, the BMC core (in that case, ESBMC) is not changed, but instead an operational model, which implements the behavior and features of Qt libraries, is used for providing the new code structure to be checked. Such research problem is closely related to the first one **(RP1)** and has the potential to devise a new paradigm in software verification.

Lastly, yet importantly, BMC tools like CBMC [13], ESBMC [47, 48], and LLBMC [14] represent the most prominent approaches for verifying C programs, as observed in the Intl. Competition on Software Verification [33, 34], where verifiable (correctness and violation) witnesses are of extreme importance for evaluating software verifiers [52, 53].

6. CONCLUSIONS

This paper presented the main challenges related to the verification of design correctness, in embedded systems, and also raised some important side considerations. In particular, it emphasizes that stringent constraints imposed by the underlying hardware (*e.g.*, real-time, memory allocation, interrupts, and concurrency), along with system behavior models, must be considered during verification. Additionally, there is little evidence that model checking embedded software using k-induction (and invariants), which extends BMC-based approaches from falsification to verification, can be applied to formally verify correctness and timeliness of embedded systems.

Given that software complexity has significantly increased in embedded products, there are still some (recent) advances to stress and exhaustively cover the system state space, in order to verify low-level properties that have to meet the application's deadline, access memory regions, handle concurrency, and control hardware registers. Besides, there is a trend towards incorporating knowledge about the system to be verified, which may take software verification one step further, where not only code correctness will be addressed, but also full system reliability. Finally, it seems interesting to provide behavioral models when new application interfaces or programming language features are used, in order to extend the capabilities of current verification tools, without changing the core BMC module.

As future work, the main goal of this research is to extend BMC as a design and verification tool for achieving correct-by-construction embedded system implementations. Special attention will be given to cyber-physical systems and modern micro-grids, considering small-scale versions of a distributed system, so that reliability and other system-level properties (*e.g.*, carbon emission reduction in smart cities) are amenable to automated verification, probably through behavior models.

Acknowledgment

The authors thank M. Dangl for reviewing a draft version of this paper. This research was supported by CNPq 475647/2013-0 grant.

7. REFERENCES

[1] H. Kopetz: *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Real-Time Systems Series, Springer, ISBN 978-1-4419-8236-0, pp. 1–376, 2011.

[2] Xua X., Jiaa H., Wanga D., Yub D., Chiangc H.: *Hierarchical energy management system for multi-source multi-product microgrids* In: Renewable Energy, v. 78, pp. 621–630, 2015.

[3] Lee E.: *Cyber-physical Systems: Design Challenges.* In: International Symposium on Object Oriented Real-Time Distributed Computing, pp. 363–369, 2008.

[4] Lee E.: *The Past, Present and Future of Cyber-Physical Systems: A Focus on Models*. In: Sensors 15(3): pp. 4837–4869, 2015.

[5] Groza A., Letia I., Goron A., Zaporojan S.: *A formal approach for identifying assurance deficits in unmanned aerial vehicle software* In: Progress in Systems Engineering, Springer, pp. 233–239, 2015.

[6] Cordeiro L., Fischer B., Chen H., Marques-Silva J.: *Semiformal Verification of Embedded Software in Medical Devices Considering Stringent Hardware Constraints.* In: International Conference on Embedded Software and Systems, pp. 396–403, 2009.

[7] Munir S., Stankovic J. A., Liang C.-J. M., Lin S.: *Cyber Physical System Challenges for Human-in-the-Loop Control.* In: 8th International Workshop on Feedback Computing, pp. 1–4, 2013.

[8] Kroening D., Liang L., Melham T., Schrammel P., Tautschnig M.: *Effective Verification of Low-Level Software with Nested Interrupts*. In: Design, Automation and Test in Europe, pp. 229–234, 2015.

[9] Biere A., Cimatti A., Clarke E., Zhu Y.: *Symbolic Model Checking without BDDs*. In: Tools and Algorithms for Construction and Analysis of Systems, LNCS 1579, pp. 193–207, 1999.

[10] Biere A., Heule M., van Maaren H., Walsh T., eds.: *Handbook of Satisfiability*. Volume 185 of Frontiers in Artificial Intelligence and Applications., IOS Press, 2009.

[11] Barrett C., Sebastiani R., Seshia S.A., Tinelli C.: *Satisfiability Modulo Theories*. In: Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 825–885, 2009.

[12] Armando A., Mantovani J., Platania L.: *Bounded Model Checking of Software Using SMT Solvers Instead of SAT Solvers*. In: SPIN Workshop on Model Checking Software, LNCS 3925, pp. 146-162, 2006.

[13] Clarke E., Kroening D., Lerda F.: *A Tool for Checking ANSI-C Programs.* In: Tools and Algorithms for the

Construction and Analysis of Systems. LNCS 2988, Springer Berlin Heidelberg, pp. 168–176, 2004.

[14] Merz F., Falke S., Sinz C.: *LLBMC: Bounded Model Checking of C and C++ Programs using a Compiler IR.* In: International Conference on Verified Software: Theories, Tools, Experiments. LNCS 7152, pp. 146–161, 2012.

[15] Cordeiro L., Fischer B.: *Verifying Multi-threaded Software using SMT-based Context-Bounded Model Checking.* In: International Conference on Software Engineering. pp. 331–340, 2011.

[16] Ivanicic F., Shlyakhter I., Gupta A., Ganai, M.K.: *Model Checking C Programs using F-Soft.* In: International Conference on Computer Design: VLSI in Computers and Processors, pp. 297–308, 2005.

[17] Cordeiro L., Fischer B., Marques-Silva J.: *SMT-based Bounded Model Checking for Embedded ANSI-C Software.* IEEE Trans. Software Eng. **38**(4), pp. 957–974, 2012.

[18] Eén, N., Sörensson, N.: *Temporal Induction by Incremental SAT Solving.* Electronic Notes in Theoretical Computer Science **89**(4), pp. 543 – 560, 2003.

[19] Sheeran M., Singh S., Stålmarck G.: *Checking Safety Properties using Induction and a SAT-solver.* In: International Conference on Formal Methods in Computer-Aided Design. Springer-Verlag, pp. 108–125, 2000.

[20] Bradley A., Manna Z.: *The calculus of computation - decision procedures with applications to verification.* In: Springer, pp. I-XV, pp. 1–366, 2007.

[21] ParisTech M.: *PIPS: Automatic Parallelizer and Code Transformation Framework.* Accessed 21 February 2016

[22] Henry J., Monniaux D., Moy M.: *PAGAI: A Path Sensitive Static Analyser.* In: Electronic Notes in Theoretical Computer Science, pp. 15–25, 2012.

[23] Gadelha M., Ismail H., Cordeiro L.: *Handling Loops in Bounded Model Checking of C Programs via k-Induction.* In: International Journal on Software Tools for Technology Transfer (to appear), 2015. http://dx.doi.org/10.1007/s10009-015-0407-9

[24] Beyer D., Dangl M., Wendler P.: *Boosting k-Induction with Continuously-Refined Invariants.* In: International Conference on Computer-Aided Verification, LNCS 9206, pp. 622–640, 2015.

[25] Brain M., Joshi S., Kroening D., Schrammel P.: *Safety Verification and Refutation by k-Invariants and k-Induction.* In: International Symposium on Static Analysis, LNCS 9291, pp. 145–161, 2015.

[26] Rocha H., Ismail H., Cordeiro L., Barreto R.: *Model Checking Embedded C Software using k-Induction and Invariants.* V Brazilian Symposium on Computing Systems Engineering, pp. 90–95, 2015.

[27] Donaldson A., Haller L., Kroening D., Rümmer, P.: *Software Verification using k-Induction.* In: International Symposium on Static Analysis. LNCS 6887, pp. 351–368, 2011.

[28] Donaldson A., Kroening D., Rümmer P.: *SCRATCH: A Tool for Automatic Analysis of DMA Races.* In: ACM Symposium on Principles and Practice of Parallel Programming. ACM, pp. 311–312, 2011.

[29] Große D., Le H., Drechsler R.: *Induction-based Formal Verification of SystemC TLM Designs.* In: International Workshop on Microprocessor Test and Verification, pp. 101–106, 2009.

[30] Behrend J., Lettnin D., Gruenhage A., Ruf J., Kropf T., Rosenstiel W.: *Scalable and Optimized Hybrid Verification of Embedded Software.* In: J. Electronic Testing 31(2): pp. 151–166, 2015.

[31] Ismail H., Bessa I., Cordeiro L., Lima Filho E., Chaves Filho J.: *DSVerifier: A Bounded Model Checking Tool for Digital Systems.* In: International SPIN Symposium on Model Checking of Software, LNCS 9232, pp. 126–131, 2015.

[32] Lettnin D., Nalla P. K., Behrend J., Ruf J., Gerlach J., Kropf T., Rosenstiel W., Schönknecht V., Reitemeyer S.: *Semiformal Verification of Temporal Properties in Automotive Hardware Dependent Software.* In: Design, Automation and Test in Europe, pp. 1214–1217, 2009.

[33] D. Beyer Status report on software verification - (competition summary SV-COMP). In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS 8413, pp. 373–388, 2014.

[34] D. Beyer: *Software Verification and Verifiable Witnesses - (Report on SV-COMP 2015).* In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS 9035, pp. 401-416, 2015.

[35] Inverso O., Tomasco E., Fischer B., La Torre S., Parlato G.: *Bounded Model Checking of Multi-threaded C Programs via Lazy Sequentialization.* In: International Conference on Computer-Aided Verification, LNCS 8559, pp. 585-602, 2014.

[36] Ramalho M., Lopes M., Sousa F., Marques H., Cordeiro L., Fischer B.: *SMT-Based Bounded Model Checking of C++ Programs* . In: International Conference on Engineering of Computer-Based Systems, pp. 147–156, 2013.

[37] Pereira P. Albuquerque H., Marques H., Silva I., Carvalho C., Santos V., Ferreira R., Cordeiro L.: *Verifying CUDA Programs using SMT-Based Context-Bounded Model Checking.* In: ACM Symposium on Applied Computing, Software Verification and Testing Track, pp. 1648-1653, 2016.

[38] Sousa F., Cordeiro L., Lima Filho E.: *Bounded Model Checking of C++ Programs Based on the Qt Framework.* In: Global Conference on Consumer Electronics, pp. 179–180, 2015.

[39] J. Morse. *Expressive and Efficient Bounded Model Checking of Concurrent Software.* University of Southampton, PhD Thesis, 2015.

[40] Zheng M, Rogers M, Luo Z, Dwyer M, Siegel S. *CIVL: Formal Verification of Parallel Programs.* In: International Conference on Automated Software Engineering, pp. 830–835, 2015.

[41] Grumberg O., Lerda F., Strichman O., Theobald M.: *Proof-guided underapproximation-widening for multi-process systems.* In: Symposium on Principles of Programming Languages, pp. 122–131, 2005.

[42] K. McMillan: *Widening and Interpolation.* In: International Symposium on Static Analysis, LCNS 6887, pp. 1, 2011.

[43] Bessa I., Abreu R., Cordeiro L., Filho J.: *SMT-Based Bounded Model Checking of Fixed-Point Digital Controllers).* In: Annual Conference of the Industrial Electronics Society, pp. 295–301, 2014.

[44] Abreu R., Cordeiro L., Filho E.: *Verifying Fixed-Point Digital Filters using SMT-Based Bounded Model Checking.* In: XXXI Brazilian Symposium on Telecommunications, 2013. http://dx.doi.org/10.14209/sbrt.2013.57

[45] Morse J., Cordeiro L., Nicole D., Fischer B.: *Model Checking LTL Properties over ANSI-C Programs with Bounded Traces.* In: Software and System Modeling 14(1): pp. 65–81, 2015.

[46] Bessa I., Ismail H., Cordeiro L., Chaves Filho J.: *Verification of Fixed-Point Digital Controllers Using Direct and Delta Forms Realizations.* In: Design Automation for Embedded Systems (to appear), 2016.

[47] Morse J., Cordeiro L., Nicole D., Fischer B.: *Handling unbounded loops with ESBMC 1.20.* In: Tools and Algorithms for the Construction and Analysis of Systems. LNCS 7795, Springer Berlin Heidelberg, pp. 619–622, 2013.

[48] Morse J., Ramalho M., Cordeiro L., Nicole D., Fischer B.: *ESBMC 1.22.* In: Tools and Algorithms for the Construction and Analysis of Systems. LNCS 8413, Springer Berlin Heidelberg, pp. 405–407, 2014.

[49] Kahlon V., Wang C., Gupta A.: *Monotonic Partial Order Reduction: An Optimal Symbolic Partial Order Reduction Technique.* In: International Conference on Computer-Aided Verification, LNCS 5643, pp. 398–413, 2009.

[50] Alves E. , Cordeiro L., Lima Filho E.: *Fault Localization in Multi-Threaded C Programs using Bounded Model Checking.* In: Brazilian Symposium on Computing Systems Engineering, pp. 96-101, 2015.

[51] Garcia M. , Sousa F., Cordeiro, L., Lima Filho E.: *$ESBMC^{QtOM}$: A Bounded Model Checking Tool to Verify Qt Applications.* In: International SPIN symposium on Model Checking of Software (to appear), 2016.

[52] Beyer D., Dangl M., Dietsch D., Heizmann M., Stahlbauer A.: *Witness validation and stepwise testification across software verifiers.* In: ESEC/SIGSOFT Foundations of Software Engineering, pp. 721–733, 2015.

[53] Rocha H., Barreto R., Cordeiro L., Dias Neto A.: *Understanding Programming Bugs in ANSI-C Software Using Bounded Model Checking Counter-Examples.* In: International Conference on Integrated Formal Methods, LNCS 7321, pp. 128-142, 2012.